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}

Table of Contents

List of Figures

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 th econsole 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 n2-n 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).

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).

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 if not please download it from 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.

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 .

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

The non-interactive run Pepper using the passed parameters and terminates after it. This is very usefull, in case of Pepper is called by a script etc. Just call

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

or

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

where OPTIONS is on of the following:

- workflow-file

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

- list

  A table with information about all available Pepper modules.

- list module-name

A table with information about the passed Pepper module.

- self-test

  Tests if the Pepper framework is in runnable mode 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 XXX.

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

  A table with information about all available Pepper modules.

- list module-name

  A table with information about the passed Pepper module.

- self-test

  Tests if the Pepper framework is in runnable mode or if any problems are detected, either in Pepper itself or in any registered Pepper module.

Chapter 2.

workflow description

A Pepper workflow description can be modeled and persisted in an xml file following a specific notation and having the extension ending '.pepperparams'. As already mentioned, a Pepper conversion process consists of three phases, and the notation of a workflow description file follows this structure. To identify a Pepper module realizing a step, you have to declare that module by triggering its name. Note, that for each Pepper workflow description, the specification of an importer and an exporter is necessary, whereas a manipulator is optional. Example 1 shows an excerpt of a Pepper workflow description file using all types of modules.

```
<?xml version="1.0" encoding="UTF-8"?>
    <pepperParams:PepperParams xmlns:xmi="http://www.omg.org/XMI" xmlns:pepperPa
        <pepperJobParams id="1">
            <importerParams moduleName="ImporterName" sourcePath="..."/>
            <!-- ... -->
            <moduleParams moduleName="ManipulatorName"/>
            <!-- ... -->
            <exporterParams moduleName="ExporterName" destinationPath="..."/>
        </pepperJobParams>
    </pepperParams:PepperParams>
```

If the Pepper module you want to use is an importer or an exporter, you can also specify the module by declaring the format name and the format version of the corpus you want to import or export. The Pepper framework will search for an import or export module handling this format. Example 2 shows the specification of an importer or exporter to be used by mentioning the format name and format version of the corpus.

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

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

Note

The value of the attributes 'sourcePath', 'destinationPath' and 'specialParams' (as we show in the following) 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)
```

Special Parameters

A single step can be parametrized by passing a special parameter file to the corresponding Pepper module. This can be done with the attribute 'specialParams' in the workflow description file. The 'specialParams' attribute can be attached to the element 'importerParams', 'exporterParams' and 'moduleParams' as shown in Example 3.

```
<?xml version="1.0" encoding="UTF-8"?>
    <pepperParams:PepperParams xmlns:xmi="http://www.omg.org/XMI" xmlns:pepperPa
        <pepperJobParams id="1">
            <importerParams moduleName="..."
                        sourcePath="..."
                        specialParams="PATH_TO_PARAMETER"/>
            <!-- ... -->
            <moduleParams   moduleName="..." specialParams="PATH_TO_PARAMETER"/>
            <!-- ... -->
            <exporterParams moduleName="ExporterName"
                        destinationPath="..."
                        specialParams="PATH_TO_PARAMETER"/>
        </pepperJobParams>
    </pepperParams:PepperParams>
```

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).

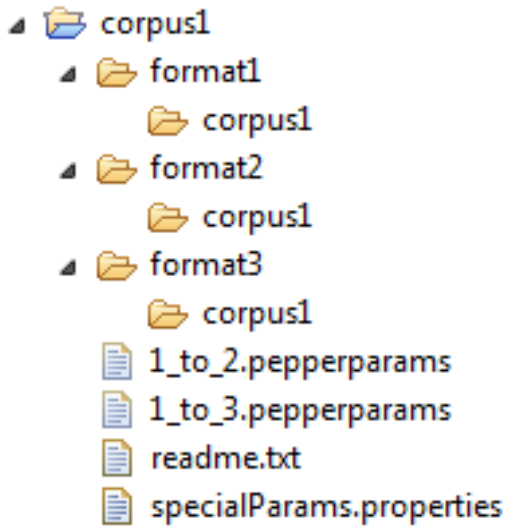
Troubleshooting

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. 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.

Corpus Structuring

For an easier overview, we recommend a specific folder structure for a corpus and its several formats. The last years have shown that when working with a set of corpora, it is not so easy to remember where a specific format of a single corpus can be found. We recommend to try to keep everything as one bundle. Imagine we have a corpus named corpus1, which is available in the formats 'format1', 'format2' and 'format3'. Further imagine that we have workflow descriptions to convert the corpus from 'format1' to 'format2' called '1_to_2.pepperparams' and format1 to 'format3' called '1_to_3.pepperparams'. A module can also take a parametrization given in a special parameter file, let's say one of the modules used in '1_to_3.pepperparams' references the special parameter file 'specialParameter.properties'. Maybe we also have a short description, which describes the corpus and its specifics called '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.

Chapter 3. Pepper as a library

- Pepper is a service infrastructure, therefore more parallel jobs are possible, so one conversion is a job, job is identified by unique id - a job consists of steps (identifying module, carrying customization of step and im-or export path if neccessary) - first show a sample - Pepper pepper= Pepper.createPepperX(); - String id= pepper.createJob() - Job job= pepper.getJob(id) - job.createStep() - ... - job.convert() - than explain what does each step mean and which alternatives are there - create an object of type Pepper (several ways) - via static creation with properties like Pepper.createPepperRESTClient(PepperConfiguration), Pepper.createPepperInOSGi(PepperConfiguration), Pepper.createPepperWithEnvironment(PepperConfiguration) - ecplain PepperConfiguration for different ways - you need a job, to determine what pepper should for instance, if pepper should convert a corpus from paula to annis, or if pepper should merge a corpus from exmaralda, tiger and mmax into paula etc. - a job is identified via a string id - String id= Pepper.createJob(); - a step is one task for instance the import of one format like PAULA, or one manipulation, like merging data is a step, or the export into one format is a step - automatically resolving format and module to a given corpus, gives you a list of possible modules - creating a StepDesc - by module name and module version - or by format name and format version - adding step description to job: three phases, import, manipulation and export, during manipulation order matters! but not in import or export - starting job and result - also possible, to postprocess a SaltProject from your application or to import data to a SaltProject for your application, than use convertTo() or convertFrom(), - in first case, you have to set SaltProject via job.setSaltProject(SaltProject), import step is automatically set to DoNothingImporter() - in second case, you have to define importers, exporter is automatically set to DoNothing, in the end you can use SaltProject with job.getSaltProject() - some (very simple) modules are part of pepper distribution like SaltXML importer and DoNothing to make sure, something is there - others need to be registered - if you don't want to make pathes absolute, you can use the variable $PEPPER_HOME, which will be replaced by the detected pepper home location. Now you can make pathes relative to the pepper home loctation, by adding a relative path after the string '$PEPPER_HOME' for instance imagine the pepper home location is located in /home/pepper/, and your plugins are located in /home/pepper/plugins, you can adopt the plugin property, by the following change in the config file: .plugins= $PEPPER_HOME/plugins This property will be automatically resolved to /home/pepper/plugins.

Chapter 4. Customization properties

Main memory and processing speed

- there are options, to adopt main memory usage of pepepr while conversion - to this via number of SDOcument or SDocumentgraph objects, which are processed at one time - depending on the size of the primary text, the number of tokens, the number of annotations and the deepness of annotation structure, a document can vary in used size of main memory - can cause in some big documents need more memory, than os can give to pepper - solution is to reduce number of sdocuments processed at a time - flag max number of processed dcouments (default value is 10) - main memory and speed often are orthogonal, there - less documents in main memory at the same time will decrease processing speed, so when you have a big number of documents which are very small itself, might be useful, to increase the number of Sdocuments processed at a time - another flag is the intermediate storage policy of pepper - here we provide the flag 'memory policy' and the values thrifty, moderate and greedy - what does this mean? often Pepper modules need different processing time for a document, depending on the task they have to do - therefore, between 2 modules, there is a queue between modules - so if first module is faster then second, a lot of modules will wait in the queue, depending on value in property max number of processed documents - since this can speed down processing time, pepper can store the waiting documents to disk, so that they will not be active any more. - thrifty means each document is stored when completed by one modules - moderate means a document is stored, when no afterwards comming module is waiting for a document - greedy will never store a document