# CONTENTS

1 Credits and history  .............................................. 3

2 Introduction  ...................................................... 5
   2.1 Background .................................................. 5

3 Getting started  .................................................... 7
   3.1 Designing corpora ............................................. 7
   3.2 Building corpora .............................................. 7
   3.3 Serving corpora ............................................... 8
   3.4 Browsing corpora ............................................. 9

4 Designing corpora  ................................................ 11
   4.1 Preparing DjVu sources ...................................... 11
   4.2 Creating a tagset ............................................. 12
   4.3 Preparing metadata .......................................... 12
   4.4 Next steps .................................................... 13

5 Building corpora - basic tagset  .............................. 15
   5.1 Preparing DjVu sources ...................................... 15
   5.2 Creating intermediate files ................................. 16
   5.3 Creating the binary corpus .................................. 17
   5.4 Augmenting the corpus ..................................... 18
   5.5 Configuring Poliqarp ....................................... 19
   5.6 Next steps .................................................... 19

6 Building corpora - custom tagset  .......................... 21

7 Serving corpora  .................................................... 23
   7.1 Running the Poliqarp daemon .............................. 23
   7.2 Configuring and running Marasca ......................... 24
   7.3 Query logging ............................................... 26
   7.4 Next steps .................................................... 27

8 Browsing corpora  ................................................ 29
   8.1 Web interface ............................................... 29
   8.2 Client application .......................................... 29
   8.3 Default Poliqarp client .................................... 30

9 File formats  ....................................................... 31
   9.1 DjVu .......................................................... 31
   9.2 hOCR ......................................................... 31
   9.3 XCES .......................................................... 31
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.4</td>
<td>Corpus configuration file</td>
<td>32</td>
</tr>
<tr>
<td>9.5</td>
<td>bpng configuration file</td>
<td>32</td>
</tr>
<tr>
<td>9.6</td>
<td>Corpus header</td>
<td>32</td>
</tr>
<tr>
<td>9.7</td>
<td>Structure file</td>
<td>32</td>
</tr>
</tbody>
</table>
This documentation is intended for users and administrators of Poliqarp for DjVu, a set of software tools for creating corpora from scanned documents in DjVu format, such as digitalized dictionaries.

Throughout the instructions it's assumed that you're using a GNU/Linux system. In theory it should be possible to run all the described software on Windows, but it might require some additional effort, and it's not discussed here.

If you find any bugs in the documentation or you want to suggest any additions, feel free to mail the author.

Contents:
POLIQARP FOR DJVU WAS DEVELOPED BY JAKUB WILK BASED ON IDEAS AND SPECIFICATION BY JANUSZ S. BIEŃ. DEVELOPMENT WAS SUPPORTED BY THE POLISH MINISTRY OF SCIENCE AND HIGHER EDUCATION’S GRANT NO. N N519 384036.

SOME OF THE MATERIAL IS BASED ON PREVIOUS INSTRUCTIONS CREATED IN POLISH BY JANUSZ S. BIEŃ, KRZYSZTOF SZAFRAN, AND TOMASZ OLEJNIČZAK. NUMEROUS COMMENTS AND VALUABLE ADDITIONS WERE CONTRIBUTED BY JANUSZ S. BIEŃ AND JAKUB WILK.

CONFIGURATION SAMPLES FOR MARASCA ARE BASED ON JAKUB WILK’S ORIGINAL INSTALLATION FOR THE VIRTUAL LEXICOGRAPHICAL LIBRARY HOSTED BY THE DEPARTMENT OF FORMAL LINGUISTICS AT THE UNIVERSITY OF WARSAW.
CHAPTER TWO

INTRODUCTION

Poliqarp for DjVu (a.k.a. Marasca ¹) is a set of tools for building corpora from scanned documents, such as historical dictionaries. The tools can be roughly divided into two groups.

The first one includes scripts for augmenting binary corpora with information about graphical coordinates of corpus segments in DjVu documents. This makes it possible to display graphical concordances for the corpus query results. In other words, the user can see the results of their query in the digital version of the source document. The second group allows serving such augmented corpora over the network and browsing the graphical concordances.

Poliqarp for DjVu is available under the terms of the GNU General Public License v. 2. The current version is stable, but new features are still under development. The upcoming additions will make it possible to create corpora with more sophisticated tagsets.

If you want to try out Poliqarp for DjVu, the best place to start is the Virtual Lexicographical Library of the Formal Linguistics Department of the University of Warsaw. The library gives access to several corpora built from digitalized versions of historical Polish dictionaries.

If you like what you see there and want to learn more about how the software works, you can download a virtual machine which comes with a preconfigured version of Poliqarp for DjVu. It can be used to try out the software and to learn about the technical aspects of its configuration.

This documentation is primarily for administrators and corpus creators. It assumes some proficiency with Linux, especially working with the console. Because the software is based on several third-party tools, the documentation contains many references to external sources to avoid duplicating information and instructions that are already available elsewhere.

2.1 Background

The idea behind Poliqarp for DjVu combines the advantages of two technologies: the Poliqarp corpus query engine and the DjVu document format. Poliqarp for DjVu operates by augmenting a standard Poliqarp corpus with information about the bounding box coordinates of the segments in DjVu documents.

Poliqarp was originally intended for linguistic corpora, but some of its features can be used successfully to facilitate access to large digitalized documents. These features include:

- a powerful query language,
- support for two-level annotation and polyinterpretation.

The latter means that Poliqarp corpora store not only the surface (i.e. orthographic) representation of words, but also the base forms of words that result from morphological analysis. Since automatic morphological analysis often cannot produce unambiguous results, all possible interpretations are stored in the corpus - hence polyinterpretation.

¹ In the remainder of this document, I’ll use the term Poliqarp for DjVu to refer to the whole set of tools necessary to build and serve DjVu-based corpora, and the term Marasca to refer to the Web client which allows querying the corpora and displaying results in the graphical layer of the source documents.
This feature can be useful for digitalized and OCR-ed documents as well, for example in the context of historical documents in which some characters might map into several possible characters in modern orthography.

The DjVu format is useful for scanned documents for three main reasons:

- the possibility to create “unbundled” (a.k.a. single-page) documents, which can speed up displaying query results for large documents (e.g. dictionaries), because only a single page with a particular result needs to be downloaded;
- support for hidden text, which makes it possible to store the results of OCR together with the digital (i.e. graphical) representation of the original document;
- support for highlighting regions in the documents determined by graphical coordinates, which makes it possible e.g. to jump directly to specific words in the graphical representation of the documents.

When combined, Poliqarp and DjVu can be used to build corpora that allow efficient searching in large digitalized documents, using a powerful query language and optionally displaying search results in the graphical representation of source documents.

There are several possible workflows for creating a new Poliqarp for DjVu corpus and making it accessible to the public. Some of them are described in the following chapters.

The most basic workflow consists of the following steps:

1. Preparing the DjVu sources, which includes optical character recognition (OCR) and unbundling the documents.
2. Creating an hOCR document from DjVu sources.
3. Transforming the hOCR into XCES.
4. Creating a binary Poliqarp corpus from XCES.
5. Augmenting the binary corpus with graphical coordinates of the words (segments) in the DjVu sources.

See also:


Poliqarp for DjVu source code repository

Poliqarp for DjVu instance at the Formal Linguistics Department of the University of Warsaw

A virtual machine with a working instance of Poliqarp for DjVu
Using Poliqarp for DjVu can be roughly divided into four main areas:

- Designing a corpus
- Building a corpus from DjVu materials
- Serving a corpus locally or over the Internet
- Browsing a corpus served on the local or on a remote host

The following sections contain tips to get you started as quickly as possible and describe the prerequisites for all the scenarios. More detailed instructions are provided in other chapters of the documentation.

### 3.1 Designing corpora

The design phase consists of specifying the tagset and the metadata for the corpus, as well as preparing the source DjVu material. This step does not require any special tools, but a general understanding of Poliqarp is needed.

See also:

* Designing corpora
* Man pages for `bpng`
* Daniel Janus’s MSc thesis (in Polish, contains valuable information about Poliqarp internals)

### 3.2 Building corpora

To create corpora, you need a set of tools installed on your system: Poliqarp, DjVuLibre, and Jakub Wilk’s scripts (distributed as part of Marasca).

#### 3.2.1 Installing Poliqarp

Poliqarp is available from [this website](#). If you’re running a Debian-based Linux distribution, you can download and install one of the binary packages. Otherwise, download the source package and perform the following steps to compile Poliqarp and its corpus building utilities:

```bash
tar xzf poliqarp_X.Y.ZZ.tar.gz
cd poliqarp_X.Y.ZZ
./configure
make
```
At this point, you can use `make install` to install both the corpus building tools, and the Poliqarp server and client. If you only plan to create corpora and not query them on your local machine, you can just copy the `bpng` binary from `poliqarp_X.Y.ZZ/bin` to a directory in your `PATH`. However, it's still recommended to run a local Poliqarp server to be able to query the new corpus for validation and debugging purposes.

### 3.2.2 Installing DjVuLibre

DjVuLibre is a library and a set of utilities built around it for manipulating DjVu files. If you're building a corpus from DjVu files, you're very likely to need some of these utilities. DjVuLibre is packaged in all popular Linux distributions, so use your package manager to find and install it. In general, there should be no need to compile DjVuLibre on your own.

### 3.2.3 Installing pdf2djvu and ocrodjvu

pdf2djvu is a converter for creating DjVu files from PDF documents. It's available here.

ocrodjvu is a utility for performing optical character recognition (OCR) on DjVu documents. It can be downloaded from the author's website.

### 3.2.4 Installing additional scripts

To create corpora from DjVu, some additional scripts are needed which are part of Marasca (a web interface to Poliqarp which supports displaying results in DjVu files). The easiest way to obtain the scripts is to clone the source repository. Make sure you have Mercurial installed on your machine, and follow the steps below:

1. Visit Marasca's repository.
2. Click the **Clone** button. This will display a clone command for Mercurial.
3. Paste the command into a terminal and execute it. This will copy the content of the repository to your local machine.
4. Finally, use this command to update the cloned repository to the correct branch: `hg update wbl`.

If you're not planning to serve corpora over the Internet, you can ignore most of the contents of Marasca’s repository and only copy the files from `marasca-wbl/misc` into a directory in your `PATH` (make sure to make them executable).

**See also:**

- **Building corpora - basic tagset**
- **Building corpora - custom tagset**

### 3.3 Serving corpora

If you want to serve your corpora locally or over the network, you need to install the Poliqarp daemon (`poliqarpd`). Additionally, if you want create a Web interface to your corpora, you need to install and set up Marasca.
3.3.1 Installing the Poliqarp daemon

To install the Poliqarp daemon, follow the instructions in Installing Poliqarp and use `make install` after compilation.

If you plan to access the daemon remotely, make sure your machine is not blocking connections to Poliqarp’s port (4567 by default).

**Note:** If you perform corpus development activities on the same machine which is used for serving your corpora, it’s useful to run two separate Poliqarp daemons: one for remote users, and one for the local user/administrator.

3.3.2 Prerequisites for installing Marasca

Installing and configuring Marasca is described in detail in the following chapters. To prepare your machine for running Marasca, install the following dependencies from your distribution’s package repository (the names below correspond to Debian/Ubuntu packages, they might be different in your distribution):

- apache2
- django-localeurl
- libapache2-mod-wsgi
- pycairo
- python
- python-django
- python-djvulibre

**See also:**

Serving corpora

3.4 Browsing corpora

Accessing DjVu corpora is possible either through a Web interface provided by Marasca, or through a standalone client (DjView for Poliqarp).

To access the Web interface, all you need is a modern Web browser with a DjVu plugin, such as the one provided with DjView4.

If you rather use a client application, download and install DjView for Poliqarp from its website.

DjVu corpora can also be accessed using the standard Poliqarp GUI. In this case, it’s not possible to display graphical results. On the other hand, statistical queries can be used (they are blocked in Marasca to prevent choking the server with computationally intensive requests by malicious users).

**See also:**

Browsing corpora
This chapter presents basic information about the structure of Poliqarp corpora and considerations that should be taken into account when creating a new corpus. This mainly means preparing the DjVu sources (including performing optical character recognition), designing a tagset, and deciding on the structure of metadata.

4.1 Preparing DjVu sources

The first step to building a new corpus is to prepare the source DjVu files. In general, there are two possible scenarios (which can sometimes co-occur in one corpus): the source documents are either paper publications, or they are digitally-born.

In the former case, DjVu files need to be created by scanning and performing optical character recognition (OCR) on the paper documents. This will not be described in detail here, as there are numerous ways to go about this task. However, some examples will be given in subsequent chapters where real-life examples are discussed.

In the latter case, either DjVu files are already available, or the documents are in another digital format, such as PDF, which can be converted into DjVu. In both cases, it’s crucial that the DjVu files contain the hidden text layer with the results of OCR.

4.1.1 Unbundling DjVu files

If you have a single, multipage DjVu document, split it into single page documents (this is known as “unbundling”). This operation will make querying and displaying the results faster, because instead of downloading the whole, possibly big document, only the relevant page will need to be fetched to display the results.

Unbundling is a simple operation which requires the DjVuLibre package. Let’s assume you have a multiple page DjVu document called source.djvu. To split it into separate pages, use the following command:

```
djvmcvt -i source.djvu ./ index.djvu
```

This will create a set of single page documents in the current directory.

4.1.2 Other DjVu operations

Apart from unbundling, djvulibre makes it possible to manipulate DjVu documents in additional ways. The package comes with several utilities which can be used to, among other things, remove duplicate pages from a DjVu file.

In the example below, `djvm` is used in a simple bash loop to remove pages 1000-1105 from a document:

```
for i in $(seq 1000 1105); do djvm -d document.djvu $i; echo "Removed page $i"; done
```

Refer to DjVuLibre’s documentation for more information and examples.

4.2 Creating a tagset

A tagset is a set of markers which describe the segments (roughly: words) in a corpus. A tagset for a Poliqarp corpus needs to be defined in the corpus’s configuration file (see File formats for a full specification of the configuration file).

There are two groups of information defined in a tagset. First of all, a set of categories, that is attributes of a segment. Secondly, there are classes to which a segment can belong. If you think about linguistic corpora, a category would be something like “gender”, and a class could be a “noun”.

In technical terms, categories are specified by the [ATTR] keyword in the corpus configuration file, and are key/value pairs in which there can be multiple values. Classes are specified by the [POS] keyword and are key/value pairs as well, but:

- the value list might be empty (i.e. the class has no attributes);
- if present, the values correspond to category keys.

Here’s how the noun/gender example would look in a corpus configuration file, with “determiner” added to demonstrate a class with no attributes:

```plaintext
[ATTR]
gender = masculine feminine neuter

[POS]
noun = gender
determiner =
```

Practical examples of how such tagsets can be used to describe Poliqarp for DjVu corpora are provided in the following chapters.

4.3 Preparing metadata

Metadata are used in Poliqarp to store additional information about each document in a corpus, so that it’s possible for example to query only those documents that were published before 1950. There’s no predefined set of metadata in Poliqarp. The corpus designer can create a custom set of fields, which then need to be filled with information for each individual document in the corpus.

Metadata fields are defined in the bpng ¹ configuration file (see File formats for more information) and use XPath syntax, because the actual metadata is then supplied in XML format.

Let’s assume that you want your documents to be described with a set of three metadata fields: the year of publication, the author, and the source of the document. In this case, the metadata section of the bpng configuration file will look like this:

```plaintext
[meta]
name = year
path = meta/year

[meta]
name = author
path = meta/author
```

¹ bpng is a utility for building binary Poliqarp corpora. It’s described in more detail in the following chapters.
This definition corresponds to the following XML skeleton:

```xml
<meta>
    <year></year>
    <author></author>
    <source></source>
</meta>
```

4.4 Next steps

The theory presented in this chapter is shown in practice in the following chapters, where you can find some step-by-step examples of building corpora of varying complexity from scratch.
This chapter provides a step-by-step walkthrough for building a new corpus based on the example of *The Dictionary of 16th Century Polish*.

The input for any corpus building operation should consist of:

- DjVu files,
- a corpus configuration file with a tagset definition, which should be created during the design phase (see *Designing corpora*).

In the case of this example, the tagset is very simple. It’s assumed that all the segments belong to one class (“ign”, for “ignore”) which doesn’t have any attributes. Thus, the configuration file looks like this:

```plaintext
[attr]
[ign ]
[pos]

[named-entity]
entity-orth = orth
entity-base = base
entity-tag = tag
entity-pos = pos
```

See *File formats* for a more detailed specification of the corpus definition file. Examples of corpora with more complex tagsets are provided in the following chapters.

### 5.1 Preparing DjVu sources

The Dictionary of 16th Century Polish is a multivolume work. Work on the early volumes started before the onset of the digital age. Thus, the DjVu sources for the corpus are a mix of scanned paper dictionaries and digitally-born documents.

Details of the OCR process will not be presented here (it was performed by the staff of the Kujawsko-Pomorska Digital Library). The important thing for the purposes of this tutorial is that all the DjVu source files had both the graphic layer and the hidden layer with OCR-ed text.

The actual corpus covers 35 volumes of the dictionary. For the purposes of this demonstration, we’ll work with 3 volumes.

The first thing to do is to create a workspace and arrange the source files. Let’s assume the following directory structure:
The next thing to do is to unbundle the DjVu files, i.e. convert them into single page documents. This can be achieved by running the following command on each DjVu file:

```
djvmcvt -i v01/v01.djvu v01/unbundled/index.djvu
```

For each volume, this will create a set of DjVu files in the unbundled directory, each corresponding to a single page, and an index.djvu file which can be though of as an entry point to the document.

### 5.2 Creating intermediate files

After splitting the source DjVu file into single-page documents, it’s time for some more conversion which will eventually lead to a Poliqarp corpus.

First of all, convert the DjVu files into hOCR documents. In the case of a bundle of single page DjVu files, index.djvu needs to be used as the entry point. Run the following command for each volume:

```
djvu2hocr --word-segmentation=uax29 v01/unbundled/index.djvu > v01/hocr/v01.hocr
```

At this point it’s possible to add structure information to the hOCR files, which can allow only specific parts of the corpus to be queried (such as the original work’s front matter) using the within keyword. In order to add structure, you need to prepare a special configuration file (the File formats chapter contains a full specification). For the 3-volume dictionary in this example, the structure file (structure.txt) looks like this, with each line representing a logical section in the original paper document:

```
1
front, 1,140
intro, 5,140
list, 141,146
body, 147,452
errata, 450,452
back, 453,454
inset, 455,457
back, 458,459

2
front, 1,6
list, 7,13
```
Once you have the structure configuration file prepared, add the structure to the hOCR files with this command:

```
annotate-hocr --in-place v01/hocr/v01.hocr v02/hocr/v02.hocr v03/hocr/v03.hocr <
structure.txt
```

The last conversion step before creating the actual corpus files is to transform the hOCR file into the XCES format, which is understood by Poliqarp’s corpus building utility. You can do this with the following command (repeat for each file):

```
hocr2xces.py < v01/hocr/v01.hocr > v01/xces/v01.xml
```

### 5.3 Creating the binary corpus

Now it’s time to create the corpus files which will be then used by the Poliqarp server. The corpus building utility (bpng) needs the following as input:

- A configuration file which controls the build process
- For each volume, the XCES file created in the previous step, and a header file with metadata information

First of all, arrange the input files into the following structure. Note that you need to rename your XCES files to something uniform (in this example we opted for `data.xml`).

```
|__v01/
 |  |__header.xml
 |  |__data.xml
 |__v02/
 |  |__header.xml
 |  |__data.xml
 |__v03/
 |  |__header.xml
 |  |__data.xml
 |__newcorpus.bp.conf
```

**Warning:** Do not delete the hOCR files from previous section just yet. They’ll be needed in one of the following steps.

**Note:** Note the format of the configuration file name. The first part (`newcorpus`) is the base name of the corpus we’re creating. In real applications, this should be more unique and chosen with some consideration, as it will be used in other configuration files and some of the subsequent steps.
Secondly, prepare the build configuration file. It defines the corpus locale, the file names that `bpng` should expect, and the metadata fields (preparing metadata is described in more detail in *Designing corpora*).

```
[locale]
locale = pl_PL

[filenames]
header = header.xml
morphosyntax = data.xml

[meta]
name = vol
path = /meta/volume

[meta]
name = year
path = /meta/year

[meta]
name = range
path = /meta/range

[meta]
name = orig
path = /meta/orig
```

Thirdly, edit the header file for each volume to make it conform to the metadata structure defined in the configuration file. For example, the header file for volume 1 in our example looks like this:

```
<meta>
  <volume>I</volume>
  <year>1966</year>
  <range>A-Bany</range>
  <orig>scan</orig>
</meta>
```

Now everything is ready to create the corpus. Run the following commands to create the binary files and an index which will speed up getting query results. Depending on the size of your corpus, this might take a while.

```
bpng newcorpus v01/ v02/ v03/
bpindexer -i o newcorpus
```

You should end up with a lot of new files prefixed with the base name of your corpus. At this point, the corpus is almost ready.

### 5.4 Augmenting the corpus

The final step is to augment the corpus with information about the coordinates of the individual segments on the pages of source DjVu files. This makes it possible to jump from query results to the exact place in the scanned source where the match is located.

To augment the corpus, you need the corpus from the previous step, as well as the hOCR files created at the very beginning of the procedure. Run the following command to augment your corpus:

```
augment-djvu-corpus.py newcorpus v01/hocr/v01.hocr v02/hocr/v02.hocr v03/hocr/v03.hocr
```
This should result in creating a `newcorpus.djvu.coordinates` file.

**Note:** It’s a good idea to perform a quick validation step now. The `.poliqarp.corpus.image` file should be 1.5 times bigger than the `.djvu.coordinates` file. If this is not the case, something went wrong and Poliqarp will probably crash when querying your corpus.

The corpus is now ready.

### 5.5 Configuring Poliqarp

Now you should be able to test the new corpus by running the Poliqarp daemon and querying the corpus with either the command line client or the graphical client. To do this, copy the binary corpus files and the corpus configuration file (the file with tagset definition, not the `bpng` configuration file!) to a separate location, e.g. `/srv/corpora/newcorpus`. Next, create a configuration file for the Poliqarp daemon and save it under `/srv/corpora/config/poliqarpd.conf`. For the purposes of this example, it will look like this (see Poliqarp manual pages for a full description of the options):

```plaintext
port = 4567
logging = on
log-file = /srv/corpora/newcorpus/query.log
corpus = newcorpus: /srv/corpora/newcorpus
max-match-length = 100
```

You can now start the Poliqarp daemon with the following command:

```
/usr/local/bin/poliqarpd -c /srv/corpora/config/poliqarpd.conf
```

If the daemon starts successfully, you can try to query the corpus to see if it’s working. See *Browsing corpora* for information on accessing the corpus.

### 5.6 Next steps

If the corpus works fine locally, you can configure Marasca, which will make it possible to display query results directly on the pages of the digitalized documents. See *Serving corpora* for details.
CHAPTER
SIX

BUILDING CORPORA - CUSTOM TAGSET

This chapter is a work in progress.
This chapter describes the ways in which you can serve corpora locally or over the network, so that they can be accessed and queried using various clients.

If you’re a regular user and only want to query a corpus, head to the Browsing corpora chapter.

7.1 Running the Poliqarp daemon

The main prerequisite for serving any corpus is to install and run the Poliqarp daemon. The installation process is described in the Getting started chapter.

There’s no single, mandatory way to run the Poliqarp daemon. If you only use it occasionally, you can start it manually like this:

```
poliqarpd -c <path to Poliqarp configuration file>
```

If, on the other hand, you want to run a proper Poliqarp server, some additional effort will be needed. First of all, you should create a separate user for running the Poliqarp daemon. You probably also want to create a startup script and add it to the appropriate run-level in your system, so that you don’t have to start the daemon manually after each reboot.

The content of the configuration file is described in more detail in the Designing corpora and File formats chapters.

7.1.1 Adding corpora to Poliqarp

After you’ve built a corpus (see the Building corpora - basic tagset chapter) and installed the Poliqarp daemon, it’s time to add a corpus to it. Set up a directory for storing corpora files. It’s possible to keep each corpus in a separate directory, but for the purposes of these instructions we’ll take an easier (although a bit more messy) way and assume that all the corpora will be kept in `/srv/corpora`. We’ll also assume that the Poliqarp daemon configuration file is `/srv/poliqarp/poliqarp.conf`.

For each corpus that you’ve built:

1. Copy all its files (the binary corpus files, not the source or intermediate files) to `/srv/corpora`.
2. Add an appropriate line to `/srv/poliqarp/poliqarp.conf`, for example:

   ```
corpus = newcorpus: /srv/corpora/newcorpus
```

3. Restart the Poliqarp daemon (the exact way to do this depends on your setup, how you started the daemon in the first place, etc.).
7.2 Configuring and running Marasca

Note: Marasca is based on the well-known Django framework. The instructions below are not the only way to deploy Marasca. For more information about alternative ways of deployment, refer to Django documentation.

Technically speaking, Marasca is a Poliqarp client, and not a server, but it’s described in this chapter because it’s mostly used for giving users easy access to your Poliqarp server through a Web interface.

The steps presented below are intended for an Ubuntu system. Installing Marasca on other distributions should be very similar, but some details, such as package names, might be different.

First of all, install all the prerequisite software, as described in the Getting started chapter. Secondly, make sure there’s a Poliqarp daemon running on the same host. Then, follow the instructions below:

1. Create a directory from which Marasca files will be served by Apache. For the purposes of these instructions, we assume it will be /srv/marasca.

2. Copy Marasca files from the cloned source code repository to /srv/marasca.

3. Create an entry script (e.g. /srv/django.wsgi) for your application and make it readable for the Web server. The script should look like this:

```python
import os
import sys

os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'

sys.path.append('/srv/marasca/')

import django.core.handlers.wsgi

application = django.core.handlers.wsgi.WSGIHandler()
```

4. Create an Apache configuration file for your Marasca installation and save it as /etc/apache2/sites-available/marasca. For example:

```html
<VirtualHost *:80>
  ServerName marasca.example.com
  Alias /css/ /srv/marasca/media/css/
  Alias /js/ /srv/marasca/media/js/
  Alias /extra/ /srv/marasca/media/extra/
  WSGIDaemonProcess marasca
  WSGIProcessGroup marasca
  WSGIScriptAlias / /srv/django.wsgi
  CustomLog ${APACHE_LOG_DIR}/poliqarp-access.log vhost_combined
</VirtualHost>
```

Note: Change the ServerName directive to match your server.

5. Rename the /srv/marasca/setting/wbl.py file to match your hostname. For example, if your hostname is “foo”, name the file foo.py.

6. Open the file for editing and customize it for your site.
   (a) Make sure the _corpora_dir variable points to the directory where you keep your Poliqarp corpora.
   (b) Edit the list returned by the _get_corpora function to match your corpora.
   (c) Update other variables, such as the administrator’s email, to match your environment.
7. Create information pages for your corpora (see Information pages).

8. After customizing Marasca to your needs, use `a2ensite marasca` to enable the application and restart Apache.

**Note:** Make sure to configure your Web server to serve the corpus’s DjVu files at the location specified by `.djvu.filenames`. The files don’t need to reside on the same server as Marasca – you can host them in a separate location, as long as it’s accessible to Marasca.

### 7.2.1 Information pages

Each corpus in Marasca is accompanied by information pages. The pages are encoded using Django’s template language and can be easily customized.

Marasca’s template files are located in the `templates` directory. The files in there are generic error messages, help pages, etc. At least the `about.html` template should be customized before you put Marasca online. The other files should be more or less ready to be used as they are.

The most important templates reside in the `corpora` subdirectory. They are information pages for each corpus served by Marasca. The information page for each corpus can consist of two parts: the main part, and a “suffix” page (the mechanism is slightly similar in nature to Apache’s `HeaderName` and `ReadmeName` directives), each having its own template.

A mapping between a corpus and its templates is achieved through the corpus identifier from the Poliqarp configuration file (see Building corpora - basic tagset). The template name needs to match the identifier, so for example, if the corpus’s identifier is “newcorpus”, the relevant template files should be named `newcorpus.html` and `newcorpus_suffix.html`. If both templates are used for a corpus, the information page will consist of the main part, followed by the query text field, and ended by the “suffix”.

An example of a simple template is presented below. Note the usage of internationalization related tags, which make it possible to create a multilingual interface (see next section for details). Also note how standard HTML tags are used to handle layout and links to external pages.

```
{% load i18n %}

{% blocktrans %}
{% endblocktrans %}
```

**See also:**

Official Django documentation on the template engine.
7.2.2 Localizing Marasca

Marasca follows Django’s mechanisms for localization, so it’s possible to create a multilingual user interface. The prerequisite for that is to include proper markup in your template files (using `trans` or `blocktrans` directives).

When your template files are properly marked up, run the `update-i18n` script distributed with marasca. This will create `.po` files in the `locale` directory. Those files can be used to translate the messages to a target language. The exact format of the `.po` files and instructions for translating them go beyond the scope of this document. Refer to official GNU Gettext documentation (see below for a link to their website).

After translating the strings in the `.po` file, run the following command in Marasca’s root directory to generate the messages in the target language:

```
./manage compilemessages
```

Finally, restart the Web server for the changes to take effect.

**See also:**

Official Django manual on localization.

GNU Gettext manual on PO files.

7.2.3 Using Marasca to access a remote Poliqarp server

**Warning:** This has not been tested, so proceed with caution. If you succeed in setting this up, we’ll be happy to hear from you.

The setup described in the previous section assumes that both the Poliqarp daemon and Marasca are running on the same server. However, in theory nothing should prevent you from running the Poliqarp daemon on one machine, and hosting Marasca on another.

The key thing to achieve this is to add `host` and `port` parameters to a piece of code responsible for establishing a connection to a Poliqarp server. The modification should be made in the `app/views.py` file in Marasca. For example:

```python
class Connection(poliqarp.Connection):
    def __init__(self, request):
        poliqarp.Connection.__init__(self, host="192.168.0.1", port="9876")
```

**See also:**

Marasca Main page for Marasca development efforts.

7.3 Query logging

There are two types of logs that you can collect with Marasca:

- query logs
- Apache access and error logs

Query logs have the following format:
The most important information that can be gleaned from them is the corpus that was queried and the query itself (the last column). Additionally, the logs include the time of the query and the IP address of the client. The location of the query log is configured through the QUERY_LOG variable in Marasca’s configuration file (<hostname>.py). The logs can grow quite large, so it’s a good idea to set up logrotate to clean them up periodically.

Apache logs work on a different level and are mainly useful for debugging problems with Marasca (the error log), as well as for collecting detailed information about the HTTP communication taking place between the client and the server.

See also:

Apache logs documentation

### 7.4 Next steps

Once you have Marasca up and running, the corpus is ready to be used by remote users. See *Browsing corpora* for information on how to access your corpus.
There are two ways to browse DjVu corpora and benefit from the functionality which allows the user to jump from the concordances to the place of their occurrence in the DjVu file. One is Marasca, which means using a Web-based interface, and the other is DjView for Poliqarp, which is a client application.

Whichever way you use, it’s recommended to familiarize yourself with Poliqarp’s query language to fully realize its potential.

### 8.1 Web interface

The Web interface provided by Marasca is the first interface to Poliqarp for DjVu developed as part of the original project. It provides all the functionality necessary for browsing DjVu corpora. It’s main benefit is that it doesn’t require anything to be installed on the user’s computer - all that is needed is a modern Web browser with JavaScript support.

Usage of the Web interface is fairly straightforward. The home page consists of a navigation pane and the main application area. The navigation pane lists all the available corpora, and allows the user to change the interface language and access the settings. The settings aren’t numerous and should be self-explanatory.

**Note:** You need to have cookies enabled, otherwise the settings won’t be saved between sessions.

When any of the corpora links is selected, the main application area will display a short description of the corpus with a query field. After a query is completed, the results will be displayed in the same area. Clicking the result’s number displays the context information and the metadata. Clicking the result itself takes the user to the place where it occurs in the original digital document, on the basis of which the corpus was created. Each result comes with a Bookmark link which allows to save the position of the result’s occurrence in the digital document.

### 8.2 Client application

The currently recommended way to access DjVu-based corpora is DjView for Poliqarp, a client application which needs to be installed on your machine. The application is available for Linux and Windows [here](#).

DjView for Poliqarp has several advantages over the Web client:

- it has much better support for graphical concordances,
- it’s not affected by occasional DjVu browser plugin issues.

On the other hand, it currently doesn’t support the bookmarking feature offered by Marasca.

Before using DjView for Poliqarp, it’s necessary to configure the servers it will communicate with to access corpora. This can be done through **Settings → Configure... → Servers.**
Note: DjView for Poliqarp does not query Poliqarp servers (i.e. poliqarpd) directly – it talks to Marasca web applications.

The main window of DjView for Poliqarp consists of two parts. On the left side is the query pane, which allows to:
  • select the Poliqarp server to be used;
  • select the corpus you want to query;
  • enter your query;
  • switch between displaying metadata, textual concordances, and graphical concordances.

The right side displays pages of DjVu documents on which a particular query result appears.

In general, the user interface is rather straightforward. If needed, help is available by pressing F1.

8.3 Default Poliqarp client

Apart from the daemon, the Poliqarp package comes with two default clients, one command-line and another with a graphical user interface in Java. None of them offers the ability to display query results in the graphical layer of the source documents. However, both can be useful for debugging during the corpus-creation stage. In addition, they are currently the only interfaces for using statistical queries, so they might be useful for people performing qualitative linguistic research, who are more concerned with linguistic properties of texts than with their graphical appearance.
Building a Poliqarp corpus involves converting between several data representations or file formats. It’s easy to get confused with all the acronyms, so this section provides a quick reference to all the formats used in a Poliqarp for DjVu pipeline.

### 9.1 DjVu

DjVu is a computer file format designed for efficient representation of scanned documents. It’s the source format for building Poliqarp for DjVu corpora. DjVu’s key feature from the point of view of Poliqarp for DjVu is that documents can contain a hidden text layer. This means that OCR-ed text can be stored along the graphical representation of the original document. Poliqarp corpora are built based on the text layer, and sophisticated Poliqarp queries can return graphical concordances because the items in the text layer are mapped to appropriate regions of the graphical layer or, in other words, the scanned image of the source document.

**See also:**
- DjVu specification from LizardTech
- DjVu resources
- DjVuLibre utilities

### 9.2 hOCR

hOCR is a format for representing OCR output, which combines layout and recognition confidence information with the recognized text. All this data is encoded in a standard HTML document, which makes it easy to process with existing tools.

In Poliqarp for DjVu, hOCR is used as the intermediate format between source DjVu files and XCES. The intermediate format can be annotated with information about the structure of the source document, which can be later used when querying the corpus to limit the query results e.g. only to the front matter or a particular chapter.

**See also:**
- hOCR specification

### 9.3 XCES

XCES is an XML-based format for encoding text corpora. In Poliqarp for DjVu, it’s used as the input for bpng for generating a binary corpus.
See also:
Official XCES website

9.4 Corpus configuration file

The corpus configuration file is used by the Poliqarp daemon. First and foremost, it describes the tagset of the corpus, defining the categories to which the segments can belong and the attributes that categories might take.

See also:
Daniel Janus’s MSc thesis (in Polish)

9.5 bpng configuration file

bpng is the Poliqarp corpus builder. Its configuration file controls which files are included in the corpus and defines the metadata fields.

See also:
bpng man pages

9.6 Corpus header

There are only two requirements for the header file:

• It should be a well-formed XML document.
• It should include the metadata fields defined in the bpng configuration file.

A simple header file, with just two metadata fields, can look like this:

```
<meta>
  <year>1990</year>
  <creator>John Doe</creator>
</meta>
```

9.7 Structure file

The structure file can be used to mark up the corpus with information about the physical layout of the source scanned document. This allows the queries to be limited to particular parts of the document, such as the back matter.

The structure file is a plain-text document which must follow a specific syntax. In short, the file:

• must refer to specific page ranges in the DjVu document;
• must include sequential numbers for each DjVu document that is described;
• can use any names for the individual sections;
• can describe a tree-like structure, i.e. sections can be nested;
• can include comments marked with “#”.
An example is shown below. The file describes the structure of a 200-page document with front matter, body, and back matter.

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td># DjVu document number</td>
</tr>
<tr>
<td>front, 1, 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>acknowledgement, 5, 20</td>
<td></td>
<td># Nested section</td>
</tr>
<tr>
<td>body, 21, 180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>back, 181, 200</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The section identifiers can be used in corpus queries by means of the `within` clause to limit the query results to the particular section.