polib Documentation

Release 1.0.0

David Jean Louis <izimobil@gmail.com>

February 09, 2013

CONTENTS

This documentation covers the latest release of polib.

polib is a library to manipulate, create, modify gettext files (pot, po and mo files). You can load existing files, iterate through it's entries, add, modify entries, comments or metadata, etc. or create new po files from scratch.

polib supports out of the box any version of python ranging from 2.4 to latest 3.X version.

polib is pretty stable now and is used by many *opensource projects*.

polib is completely free and opensource, the license used is the MIT license. It was developed back in 2006 by David Jean Louis and it is still actively maintained.

To get up and running quickly, consult the *quick-start guide*, which describes all the necessary steps to install and use polib. For more detailed information about how to install and how to use polib, read through the documentation listed below.

Contents:

CHAPTER

ONE

QUICK START GUIDE

1.1 Installing polib

polib requires python 2.5 or superior.

There are several ways to install polib, this is explained in *the installation section*.

For the impatient, the easiest method is to install polib via pip, just type:

```
pip install polib
```

1.2 Some basics about gettext catalogs

A gettext catalog is made up of many entries, each entry holding the relation between an original untranslated string and its corresponding translation.

All entries in a given catalog usually pertain to a single project, and all translations are expressed in a single target language. One PO file entry has the following schematic structure:

```
#  translator-comments
#. extracted-comments
#: reference...
#, flag...
msgid untranslated-string
msgstr translated-string
```

A simple entry can look like this:

```
#: lib/error.c:116
msgid "Unknown system error"
msgstr "Error desconegut del sistema"
```

polib has two main entry points for working with gettext catalogs:

- the `pofile()` and `mofile()` functions to **load** existing po or mo files,
- the `POFile` and `MOFile` classes to **create** new po or mo files.

References * Gettext Manual * PO file format * MO file format

1.3 Loading existing catalogs

1.3.1 Loading a catalog and detecting its encoding

Here the encoding of the po file is auto-detected by polib (polib detects it by parsing the charset in the header of the pofile):

```
import polib
po = polib.pofile('path/to/catalog.po')
```

1.3.2 Loading a catalog and specifying explicitly the encoding

For some reason you may want to specify the file encoding explicitely (because the charset is not specified in the po file header for example), to do so:

```
import polib
po = polib.pofile(
    'path/to/catalog.po',
    encoding='iso-8859-15'
)
```

1.3.3 Loading an mo file

In some cases you can be forced to load an mo file (because the po file is not available for example), polib handles this case:

```
import polib
mo = polib.mofile('path/to/catalog.mo')
print mo
```

As for po files, mofile also allows to specify the encoding explicitely.

1.4 Creating po catalogs from scratch

polib allows you to create catalog from scratch, this can be done with the POFile class, for exemple to create a simple catalog you could do:

```
import polib

po = polib.POFile()
po.metadata = {
    'Project-Id-Version': '1.0',
    'Report-Msgid-Bugs-To': 'you@example.com',
    'POT-Creation-Date': '2007-10-18 14:00+0100',
    'PO-Revision-Date': '2007-10-18 14:00+0100',
    'Last-Translator': 'you <you@example.com>',
    'Language-Team': 'English <yourteam@example.com>',
    'MIME-Version': '1.0',
    'Content-Type': 'text/plain; charset=utf-8',
    'Content-Transfer-Encoding': '8bit',
}
```

This snippet creates an empty pofile, with its metadata, and now you can add you entries to the po file like this:

```
entry = polib.POEntry(
    msgid=u'Welcome',
    msgstr=u'Bienvenue',
    occurrences=[('welcome.py', '12'), ('anotherfile.py', '34')]
)
po.append(entry)
```

To save your file to the disk you would just do:

```
po.save('/path/to/newfile.po')
```

And to compile the corresponding mo file:

```
po.save_as_mofile('/path/to/newfile.mo')
```

1.5 More examples

1.5.1 Iterating over entries

Iterating over **all** entries (by default POFiles contains all catalog entries, even obsolete and fuzzy entries):

```
import polib

po = polib.pofile('path/to/catalog.po')
for entry in po:
    print entry.msgid, entry.msgstr
```

Iterating over **all** entries except obsolete entries:

```
import polib

po = polib.pofile('path/to/catalog.po')
valid_entries = [e for e in po if not e.obsolete]
for entry in valid_entries:
    print entry.msgid, entry.msgstr
```

Iterating over translated entries only:

```
import polib

po = polib.pofile('path/to/catalog.po')
for entry in po.translated_entries():
    print entry.msgid, entry.msgstr
```

And so on... You could also iterate over the list of POEntry objects returned by the following POFile methods:

- `untranslated_entries()`
- `fuzzy_entries()`

1.5.2 Getting the percent of translated entries

```
import polib

po = polib.pofile('path/to/catalog.po')
print po.percent_translated()
```

1.5.3 Compiling po to mo files and reversing mo files to po files

Compiling a po file:

```
import polib

po = polib.pofile('path/to/catalog.po')
# to get the binary representation in a variable:
modata = po.to_binary()
# or to save the po file as an mo file
po.save_as_mofile('path/to/catalog.mo')
```

Reverse a mo file to a po file:

```
mo = polib.mofile('path/to/catalog.mo')
# to get the unicode representation in a variable, just do:
podata = unicode(mo)
# or to save the mo file as an po file
mo.save_as_pofile('path/to/catalog.po')
```

CHAPTER

TWO

INSTALLATION GUIDE

2.1 Requirements

polib requires python 2.5 or higher.

2.2 Installing polib

There are several ways to install polib:

- Automatically, via a package manager.
- Manually, by downloading a copy of the release package and installing it yourself.
- Manually, by performing a Mercurial checkout of the latest code.

2.2.1 Automatic installation via a package manager

Several automatic package-installation tools are available for Python; the most popular are pip and easy_install . Either can be used to install polib.

Using `pip`, type:

```
pip install polib
```

Using `easy_install`, type:

```
easy_install polib
```

It is also possible that your operating system distributor provides a packaged version of polib. Consult your operating system's package list for details, but be aware that third-party distributions may be providing older versions of polib, and so you should consult the documentation which comes with your operating system's package.

2.2.2 Manual installation from a downloaded package

If you prefer not to use an automated package installer, you can download a copy of polib and install it manually. The latest release package can be downloaded from polib's page on the Python Package Index.

Once you've downloaded the package, unpack it, this will create the directory `polib-X-Y-Z`, which contains the `setup.py` installation script. From a command line in that directory, type:

```
python setup.py install
```

Note: On some systems you may need to execute this with administrative privileges (e.g., `sudo python setup.py install`).

2.2.3 Manual installation from a Mercurial checkout

If you'd like to try out the latest in-development code, you can obtain it from the polib repository, which is hosted at Bitbucket and uses Mercurial for version control.

To obtain the latest code and documentation, you'll need to have Mercurial installed, at which point you can type:

```
hg clone http://bitbucket.org/izi/polib/
```

This will create a copy of the polib Mercurial repository on your computer; you can then add the `polib.py` file to your Python import path, or use the `setup.py` script to install as a package.

CHAPTER

THREE

POLIB API

3.1 The `pofile` function

`polib.`**`pofile`**(*pofile*, ***kwargs*)

Convenience function that parses the po or pot file `pofile` and returns a `POFile` instance.

Arguments:

`pofile` string, full or relative path to the po/pot file or its content (data).

`wrapwidth` integer, the wrap width, only useful when the `-w` option was passed to xgettext (optional, default: `78`).

`encoding` string, the encoding to use (e.g. "utf-8") (default: `None`, the encoding will be auto-detected).

`check_for_duplicates` whether to check for duplicate entries when adding entries to the file (optional, default: `False`).

`klass` class which is used to instantiate the return value (optional, default: `None`, the return value with be a `POFile` instance).

3.2 The `mofile` function

`polib.`**`mofile`**(*mofile*, ***kwargs*)

Convenience function that parses the mo file `mofile` and returns a `MOFile` instance.

Arguments:

`mofile` string, full or relative path to the mo file or its content (data).

`wrapwidth` integer, the wrap width, only useful when the `-w` option was passed to xgettext to generate the po file that was used to format the mo file (optional, default: `78`).

`encoding` string, the encoding to use (e.g. "utf-8") (default: `None`, the encoding will be auto-detected).

`check_for_duplicates` whether to check for duplicate entries when adding entries to the file (optional, default: `False`).

`klass` class which is used to instantiate the return value (optional, default: `None`, the return value with be a `POFile` instance).

3.3 The detect_encoding function

polib.**detect_encoding**(*file*, *binary_mode=False*)

Try to detect the encoding used by the file. The file argument can be a PO or MO file path or a string containing the contents of the file. If the encoding cannot be detected, the function will return the value of default_encoding.

Arguments:

file string, full or relative path to the po/mo file or its content.

binary_mode boolean, set this to True if file is a mo file.

3.4 The escape function

polib.**escape**(*st*)

Escapes the characters \\, \t, \n, \r and " in the given string st and returns it.

3.5 The unescape function

polib.**unescape**(*st*)

Unescapes the characters \\, \t, \n, \r and " in the given string st and returns it.

3.6 The POFile class

class polib.**POFile**(**args*, ***kwargs*)

Po (or Pot) file reader/writer. This class inherits the _BaseFile class and, by extension, the python list type.

fuzzy_entries()

Convenience method that returns the list of fuzzy entries.

merge(*refpot*)

Convenience method that merges the current pofile with the pot file provided. It behaves exactly as the gettext msgmerge utility:

- comments of this file will be preserved, but extracted comments and occurrences will be discarded;
- any translations or comments in the file will be discarded, however, dot comments and file positions will be preserved;
- the fuzzy flags are preserved.

Keyword argument:

refpot object POFile, the reference catalog.

obsolete_entries()

Convenience method that returns the list of obsolete entries.

percent_translated()

Convenience method that returns the percentage of translated messages.

save_as_mofile(*fpath*)
Saves the binary representation of the file to given `fpath`.

Keyword argument:

fpath string, full or relative path to the mo file.

translated_entries()
Convenience method that returns the list of translated entries.

untranslated_entries()
Convenience method that returns the list of untranslated entries.

3.7 The `MOFile` class

class `polib.`**MOFile**(**args*, ***kwargs*)
Mo file reader/writer. This class inherits the `_BaseFile` class and, by extension, the python `list` type.

fuzzy_entries()
Convenience method to keep the same interface with POFile instances.

obsolete_entries()
Convenience method to keep the same interface with POFile instances.

percent_translated()
Convenience method to keep the same interface with POFile instances.

save(*fpath=None*)
Saves the mofile to `fpath`.

Keyword argument:

fpath string, full or relative path to the file.

save_as_pofile(*fpath*)
Saves the mofile as a pofile to `fpath`.

Keyword argument:

fpath string, full or relative path to the file.

translated_entries()
Convenience method to keep the same interface with POFile instances.

untranslated_entries()
Convenience method to keep the same interface with POFile instances.

3.8 The `POEntry` class

class `polib.`**POEntry**(**args*, ***kwargs*)
Represents a po file entry.

merge(*other*)
Merge the current entry with the given pot entry.

translated()
Returns `True` if the entry has been translated or `False` otherwise.

3.9 The `MOEntry` class

class `polib.`**`MOEntry`**(**args*, ***kwargs*)
Represents a mo file entry.

CHAPTER

FOUR

CONTRIBUTING TO POLIB

You are very welcome to contribute to the project! The bugtracker, wiki and mercurial repository can be found at the project's page.

New releases are also published at the cheeseshop.

4.1 How to contribute

There are various possibilities to get involved, for example you can:

- Report bugs preferably with patches if you can;
- Enhance this documentation
- Fork the code, implement new features, test and send a pull request

4.2 Running the test suite

To run the tests, just type the following on a terminal:

```
$ cd /path/to/polib/
$ ./runtests.sh
```

If you want to generate coverage information:

```
$ pip install coverage
$ ./runtests.sh
$ coverage html
```

CHAPTER

FIVE

PROJECTS USING POLIB

polib is used by many opensource projects, here are some of them:

- Mercurial
- Transifex
- Launchpad ubuntu translator tools
- Django-rosetta
- The evergreen library system
- Qooxdoo
- http://www.linux.rk.edu.pl/tra/list/

If you are using polib and wish to be listed here (or not) let me know.