wordaxe Hyphenation

Henning von Bargen, 26 Oct 2008, Release 0.3.0

Overview

The wordaxe library was formerly called "deco-cow", which is an abbreviation for "decomposition of compound words".

The library consists of three program parts:

1) an (easily extendable) class library adding hyphenation support to Python programs.

2) a special hyphenation algorithm, based on the decomposition of compound words (the implementation is only for the German language).

3) a hyphenation extension for the ReportLab PDF library.

Where to get it

The wordaxe library is hosted at SourceForge (http://deco-cow.sourceforge.net).

The current stable Release of the software can be downloaded from the SourceForge download page. There's also a subversion repository where you can get the current development code.

Licence

The wordaxe hyphenation library is dual-licensed. You are permitted to use and distribute wordaxe under one of these open source licenses:

"Apache 2.0 License" or "2-Clauses BSD-License".

For details, see the file license.txt contained in the library.

Regarding the licenes for the pyHnj library from Danny Yoo (HNJ hyphenation, contained in the wordaxe installation), for the ReportLab PDF library and the pyhyphen library please visit the corresponding web sites.

The dictionary files with the suffix .dic have been taken from the OpenOffice distribution, they are licences under the GNU LGPL.

Installation

ReportLab 2.1 or 2.2

wordaxe Release 0.3.0 has been tested with Python 2.5 and ReportLab 2.2, but it should work equally well with Python 2.4, since AFAIK none of the new features of 2.5 are used.

ReportLab 2.2 can be obtained from www.reportlab.org. The installation is easy (though not with easy-install ;-) and is not described here.

wordaxe works with ReportLab 2.1, too.

Notes on even older ReportLab versions:

Though untested, wordaxe Release 0.3.0 may work with ReportLab 2.0. Otherwise you can use 0.2.2 in this case, but the installation is a little harder (because some more files in the ReportLab installation had to be overwritten and the installation guide was not correct).

For ReportLab 1.19 please use Release 0.1.1 (not recommended). When upgrading from ReportLab 1.x to 2.x you probably have to change existing code (independently from wordaxe), since for 2.x all paragraph input text has to be unicode or UTF8 encoded.

Installating wordaxe step-by-step

1. Download the ZIP archive wordaxe-0.3.0.zip from the SourceForge site.

2. Unpack the ZIP-archive wordaxe-0.3.0.zip in the root directory C:\ (inside the archive, all files are in a directory "wordaxe-0.3.0").

The following directory structure will be created:

```
C:
    wordaxe-0.3.0
        doc
        htdocs
            css
            examples
            icons
            images
        wordaxe
            dict
            plugins
            rl
```

3a. On the command line, execute:

```
cd /d c:\wordaxe-0.3.0
setup.py install
```

3b. Alternative:

Make a backup copy of the file reportlab\pdfbase\rl_codecs.py from the ReportLab Installation; then replace the file with the modified version from c:\wordaxe-0.3.0\wordaxe\rl\rl_codecs.py.

Note: This will only change two lines of code in the file, responsible for the "shy hyphen" character SHY.

Add the new library to the Python path, for example by creating a file wordaxe.pth in c:\python25\lib\site-packages, containg only one text line:

```
c:\wordaxe-0.3.0
```

4. Assure that "import wordaxe" doesn't create an error message.

5. ReportLab will work as before; differences may only occur, if *your own* programs or texts use the SHY-character.

Usage

To see the hyphenation with the DCW-Algorithmus (Decomposition of compound words) in action for german language text, run the example script "test_hyphenation.py" in the test subdirectory. It will produce two PDF files, test_hyphenation-plain.pdf and test_hyphenation_styled.pdf.

The document you read now has also be produced with automatic hyphenation (see the buildDoku.py script).

To add hyphenation support to your own programs (here using the DCW algorithm as an example), only very few modifications in your code are necessary:

Add the following lines:

```
from wordaxe import hyphRegistry
from wordaxe.DCWHyphenator import DCWHyphenator
hyphRegistry['DE'] = DCWHyphenator('de',5)
```

Search and replace the following strings:

```
Search                            Replace with
reportlab.platypus.paragraph      wordaxe.rl.paragraph
reportlab.platypus.xpreformatted  wordaxe.rl.xpreformatted
reportlab.lib.styles              wordaxe.rl.styles
```

Enable hyphenation. To do this, set two attributes in your ParagraphStyle:

```
stylesheet = getSampleStyleSheet()
myStyle = stylesheet["BodyText"]
myStyle.language = 'DE'
myStyle.hyphenation = True
```

Using a Hyphenator

Of course the wordaxe hyphenation can be used independent from ReportLab.

In the constructor, at least a language code and a minimal word-length have to be supplied. Shorter words will not be considered for hyphenation.

```
from wordaxe.DCWHyphenator import DCWHyphenator
hyphenator = DCWHyphenator('de',5)
```

Now you can hyphenate (unicode) words. The return value will either be None (unknown word) or a HyphenatedWord, that is, a word with hyphenation points and their quality.

```
hword = hyphenator.hyphenate(u"Donaudampfschiffahrt")
print "Possible hyphenations", hword.hyphenations
# Split the word at the second possible hyphenation point:
left,right = hword.split(hword.hyphenations[1])
# returns: (u'Donau\xad', HyphenatedWord(u'dampfschiffahrt'))
# The left part is a unicode object (here: Donau-),
# the right part is the rest of the word (a HyphenatedWord instance again),
# that should go into the next line.
print left
print right
print right.hyphenations
```

Hyphenation Classes

The source code for the classes contains test code that you can examine to find out how to use the class. The test code can be called to see how the corresponding class handles the words given on the command line. Probably you want to supply the -v argument for verbose output, too.

Example

```
c:\python25\python wordaxe\DCWHyphenator.py -v Silbentrennung
```

DCWHyphenator

This class works by splitting compound words into subwords, inspired by the publications of the TU Vienna, see http://www.ads.tuwien.ac.at/research/SiSiSi/.

However, the implementation here isn't in any way related to the closed source product "SiSiSi".

The algorithm works as follows: A given compound word will be decomposed into subwords first, using the file DE_hyph.ini, which contains stems, some of them annotated with properties like NEED_SUFFIX, NO_-SUFFIX etc. Furthermore, possible prefixes and suffixes are defined there.

Due to its complexity, the algorithm is quite slow:

The word will be scanned from left to right. It will be split into a pair (L,R), where different partitions are of possible of course, for example "Trennung": ("T", "rennung"), ("Tr", "ennung"), ("Tre", "nnung"), and so on. For each pair, the algorithm checks if the left part matches a know prefix, stem or suffix from the file DE_-hyph.ini. If yes, the algorithm R continues with the remainder R analogously. Otherwise, or if the combination does not make sense (i.e. prefix + suffix without stem) the algorithm cancels.

In principle, this is a recursive algorithm, although the implementation uses a to-do like list instead.

The properties of the algorithm are:

Only the stems defined in DE_hyph.ini will be detected.

As a consequence, possible some valid hyphenation points may be missed, because unknown words will not be hyphenated at all.

On the other hand, this prevents wrong hyphenations.

If a compound word cannot be decomposed uniquely, only those hyphenation points that exist in all the decompositions. This prevents hyphenations that may be valid but hard to read.

Hyphenation points have a priority, which can be used by the calling program to prefer good hyphenation points (at subword boundaries).

This class supports all features of ExplicitHyphenator as well.

ExplicitHyphenator

This class supports supports all features of BaseHyphenator, plus:

Here you have to explicitly define the hyphenation for every word. Thus this class isn't of much use on its own - only if the dictionary is quite small (for example if more or less only fixed text templates are used).

PyHnjHyphenator

This works like the pattern-based hyphenation in TeX (see also libhnj, pyhnj). The implementation can use the pyhnj C-library or use pure Python, if the argument purePython=True is supplied to the constructor.

wordaxe.plugins.PyHyphenHyphenator

This class also works like the pattern-based hyphenation in TeX, but it uses a different implementation and *works much better*! To use it, you need to have the pyhyphen library installed (see http://pypi.python.-org/pypi/PyHyphen/).

This class supports all features of ExplicitHyphenator as well.

BaseHyphenator

This class should work for all languages. It hyphenates only after the following characters:

```
'-'   minus (45, '\x2D')
'.'   dot (46, '\x2E') (but not if the dot is between digits)
'_'   underscore (95, '\x5F')
'-'   SHY hyphenation character (173, '\xAD')
```

When the argument CamelCase=True is given to the constructor, CamelCase words will be hyphenated, too.

Remarks

Performance

The DCWHyphenator is quite slow, due to the rekursive nature of the algorithm.

Since the word length is bounded, the run time when used with ReportLab is proportional to the number of lines, because only the last word of each line will be handled by the hyphenator.

For the DCWHyphenator you can cache the results instead of using it directly, using the following code:

```
import wordaxe
from wordaxe.DCWHyphenator import DCWHyphenator
hyph = DCWHyphenator("DE")
wordaxe.hyphRegistry ["DE"] = wordaxe.Cached(hyph, 1000)
```

Extensions

Other hyphenation libraries can easily integrated with the help of ctypes or SWIG.

To do this, you have to override the member function "hyphenate", which receives a Unicode-word as input and has to return a HyphenatedWord object or None.

Notes on ReportLab

The code in platypus/paragraph.py is hard to read and therefore it is quite hard to extend the functionality.

Since I was not able to add working hyphenation support to that code in case of paragraph splitting, I decided to start a new Paragraph implementation which a lot of code rewritten from scratch (NewParagraph.py and para_fragments.py) and use that instead.