The xmllib module

This module provides a simple XML parser, using regular expressions to pull the XML data apart. The parser does basic checks on the document, such as checking that there is only one top-level element, and checking that all tags are balanced.

You feed XML data to this parser piece by piece (as data arrives over a network, for example). The parser calls methods in itself for start tags, data sections, end tags, and entities, among other things.

If you're only interested in a few tags, you can define special **start_tag** and **end_tag** methods, where **tag** is the tag name. The **start** functions are called with the attributes given as a dictionary.

Example: Using the xmllib module to extract information from an element

```
# File: xmllib-example-1.py

import xmllib

class Parser(xmllib.XMLParser):
    # get quotation number

    def __init__(self, file=None):
        xmllib.XMLParser.__init__(self)
        if file:
            self.load(file)

    def load(self, file):
        while 1:
            s = file.read(512)
            if not s:
                break
            self.feed(s)
        self.close()

    def start_quotation(self, attrs):
        print "id =>", attrs.get("id")
        raise EOFError

try:
    c = Parser()
    c.load(open("samples/sample.xml"))
except EOFError:
    pass


$ python xmllib-example-1.py
id => 031
```

The second example contains a simple (and incomplete) rendering engine. The parser maintains an element stack (**__tags**), which it passes to the renderer, together with text fragments. The renderer looks the current tag hierarchy up in a style dictionary, and if it isn't already there, it creates a new style descriptor by combining bits and pieces from the style sheet.

Example: Using the xmllib module to render XML

```
# File: xmllib-example-2.py

import xmllib
import string, sys

STYLESHEET = {
    # each element can contribute one or more style elements
    "quotation": {"style": "italic"},
    "lang": {"weight": "bold"},
    "name": {"weight": "medium"},
}

class Parser(xmllib.XMLParser):
    # a simple styling engine

    def __init__(self, renderer):
        xmllib.XMLParser.__init__(self)
        self.__data = []
        self.__tags = []
        self.__renderer = renderer

    def load(self, file):
        while 1:
            s = file.read(8192)
            if not s:
                break
            self.feed(s)
        self.close()

    def handle_data(self, data):
        self.__data.append(data)

    def unknown_starttag(self, tag, attrs):
        if self.__data:
            text = string.join(self.__data, "")
            self.__renderer.text(self.__tags, text)
        self.__tags.append(tag)
        self.__data = []

    def unknown_endtag(self, tag):
        self.__tags.pop()
        if self.__data:
            text = string.join(self.__data, "")
            self.__renderer.text(self.__tags, text)
        self.__data = []

class DumbRenderer:

    def __init__(self):
        self.cache = {}

    def text(self, tags, text):
        # render text in the style given by the tag stack
        tags = tuple(tags)
        style = self.cache.get(tags)
        if style is None:
            # figure out a combined style
            style = {}
            for tag in tags:
                s = STYLESHEET.get(tag)
                if s:
                    style.update(s)
            self.cache[tags] = style # update cache
        # write to standard output
        sys.stdout.write("%s =>\n" % style)
        sys.stdout.write("  " + repr(text) + "\n")

#
# try it out

r = DumbRenderer()
c = Parser(r)
c.load(open("samples/sample.xml"))
```

```
$ python xmllib-example-2.py
{'style': 'italic'} =>
  'I\'ve had a lot of developers come up to me and\012say,
  "I haven\'t had this much fun in a long time. It sure
  beats\012writing '
{'style': 'italic', 'weight': 'bold'} =>
  'Cobol'
{'style': 'italic'} =>
  '" -- '
```

```
{'style': 'italic', 'weight': 'medium'} =>
  'James Gosling'
{'style': 'italic'} =>
  ', on\012'
{'weight': 'bold'} =>
  'Java'
{'style': 'italic'} =>
  '.'
```