XML in Python

Konrad Hinsen
Centre de Biophysique Moléculaire, CNRS
Rue Charles Sadron; 45071 Orléans
E-Mail: hinsen@cnrs-orleans.fr

9 October 2003

XML

XML is a scheme for flexible yet standardized data storage in text files.

XML is **not** a rigid file format, it is a scheme for **defining** specific file formats that share a common syntax.

Advantage: XML processing software is standardized.

XML/HTML: HTML is an application of SGML, XML is a simplified subset of SGML. XHTML is HTML adapted to XML.

XML norms

World Wide Web Consortium norms and recommendations:

- XML
- XML namespaces: mixing document types
- Document Object Model (DOM): a language-independent object-oriented interface to XML documents
- XPath: references into XML documents
- XLink: links between XML documents
- XML Schema: XML document types in XML

Non-W3C: Simple API for XML (SAX), a standard parser interface

Python support for XML

Standard: non-validating parser, SAX interface, a simple DOM Level 1 implementation

Optional: PyXML package containing a validating parser, a faster non-validating parser, additional DOM implementations including Level 2

This tutorial covers only the standard modules.

An XML "document" file

```
<?xml version="1.0"?>

<!DOCTYPE book PUBLIC
 "-//OASIS//DTD DocBook XML V4.1.2//EN"
 "/usr/share/docbook/docbookx.dtd">

<book id="Hamlet">
<chapter>
<title>To be or not to be</title>
<para>That's the question.</para>
</chapter>
</book>
```

An XML "database" file

```
<?xml version="1.0"?>

<molecule type="water">
<atom type="H" name="H1">
<position units="nm">
<vector>
<x>-0.0757</x> <y>0.</y> <z>-0.0520</z>
</vector>
</position>
</atom>
<atom type="H" name="H2">
```

```
<position units="nm">
<vector>
<x>0.0757</x> <y>0.</y> <z>-0.0520</z>
</vector>
</position>
</atom>
<atom type="O" name="O">
<position units="nm">
<vector>
<x>0.</x> <y>0.</y> <z>0.0066</z>
</vector>
</position>
</atom>
</molecule>
```

XML Parsers

An XML parser fulfills two tasks:

- Check that a file is syntactically correct.
- Hand out the contents in small pieces for further processing.

Two kinds of parsers:

- **Validating** parsers verify compliance with a DTD.
- **Non-validating** parsers merely verify that the file is "well-formed",

XML Processing

Two common approaches to reading XML files:

- **Event driven:** the program hands a list of functions to the XML parser which the latter calls at specific points in the file, e.g. the beginning of a particular kind of data.
- **Document oriented:** the parser builds up a data object that reflect the file content. The application can inspect this object, and even modify it and then write it to an XML file again.

XML in Python: DOM

```
from xml.dom.minidom import parse

document = parse('molecule.xml')

molecule = document.childNodes[0]
print molecule.getAttribute('type')
atoms = molecule.getElementsByTagName('atom')
for a in atoms:
    print a.getAttribute('name')
    x_node = a.getElementsByTagName('x')[0]
    x = float(x_node.childNodes[0].data)
    y_node = a.getElementsByTagName('y')[0]
    y = float(y_node.childNodes[0].data)
    z_node = a.getElementsByTagName('z')[0]
    z = float(z_node.childNodes[0].data)
    print x, y, z
```

XML in Python: SAX

```
from xml.sax.handler import ContentHandler
from xml.sax import parse

class DataNode:

    def __init__(self, name, parent):
        self.name = name
        self.parent = parent
        self.data = None
        self.children = []

    def addChildNode(self, node):
        self.children.append(node)

    def setText(self, text):
        text = text.strip()
        if text: self.data = text

    def __str__(self):
        return self.asString(0)
```

```
    def asString(self, indent):
        s = indent*' '
        if self.data:
            s += "%s: %s\n" % (self.name, self.data)
        else:
            s += self.name + '\n'
        for node in self.children:
            s += node.asString(indent + 2)
        return s


class MyContentHandler(ContentHandler):

    def __init__(self):
        ContentHandler.__init__(self)
        self.object_stack = [DataNode('root', None)]

    def retrieveTree(self):
        return self.object_stack[0]

    def startElement(self, name, attributes):
        parent = self.object_stack[-1]
        node = DataNode(name, parent)
        if parent is not None:
            parent.addChildNode(node)
        self.object_stack.append(node)
        self.text_data = ''

    def endElement(self, name):
        current = self.object_stack.pop()
        current.setText(self.text_data)
        self.text_data = ''

    def characters(self, data):
        self.text_data += data

handler = MyContentHandler()
parse('molecule.xml', handler)
tree = handler.retrieveTree()
print tree
```

DOM or SAX?

DOM:

+ Random access to data
+ Predefined data structures
- Inefficient when only a small part of a file is needed
- Not very "pythonic"

SAX:

+ Efficient (time and memory)
+ Allows optimized data structures
- Only sequential access