XML processing with Python, Part 2: XPath and XSLT

RE-PACKAGE

XPath and XSLT are technologies for processing and converting XML documents, which are extremely easy to use with a scripting language like Python. Andreas Jung takes a closer look

The author

Andreas Jung lives near Washington D.C. and works for Zope Corporation as part of the Zope core team.

In the first part of our XML discussion we looked at DOM and SAX, both of which allow you to access the structure of XML documents through an API. However, since XML is primarily intended as a central exchange format it is equally important to be able to convert documents into other formats. In this part we will therefore take a look at XPath and XSLT.

XPath is a path-based navigation and processing technology for XML documents. XSLT, on the other hand, describes rule-based XML transformations. Both techniques can turn XML files into HTML or other formats.

In order to demonstrate these techniques we will again be using the XML file pythonbooks.xml in Listing 1 that you may remember from the first part. We are going to transform it into an XHTML table using both XPath and XSLT.

Navigation with XPath

XPath is a W3C standard that enables you to access the elements of XML documents (or, more precisely, their DOM tree) using a path expression. A path expression is similar to the path of a file within a filesystem, as both DOM trees and filesystems are organised hierarchically. An XPath expression references a set of nodes, a boolean value, a floating point number or a character string.

A path expression is always tied to the particular context in which it is used (typically a node of a DOM tree). The most important path expressions and XPath functions are listed in Table 1. Listing 2 (*xpath.py*) shows the transformation of our little Python book database into XHTML.

Listing 1: websync.hs

```
01 from xml.dom.ext.reader.Sax2 import FromXmlStream
02 from xml.xpath import Evaluate
03 fp = open('pythonbooks.xml','r')
04 dom = FromXmlStream(fp)
05 fp.close()
06 print "<table>"
07 print "<tr>"
08 print "<th>Author(s)</th><th>Title</th> <th>Publisher</th>"
09     print "</tr>"
10     for book in Evaluate('book', dom .documentElement):
11         print "<tr>"
12         for item in ['author','title','publisher']:
13             path = '%s/text()' % item
14             print '<td>%s</td>' % Evaluate(path, book)[0].nodeValue
15         print "</tr>"
16     print "</table>"
```

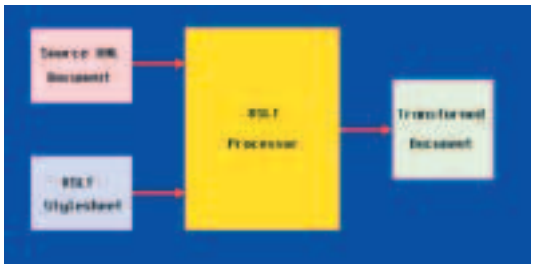

Figure 1: The XSLT processor combines XML documents with XSLT stylesheets to create other types of formats

First we use *From XmlStream()* to create a DOM tree. This is nothing new as we already did it in part 1. The function *Evaluate(xpath, context)* forms the link between DOM and XPath. It requires a path expression as an argument and a DOM node as context. In our example it steps through all document nodes of the type *book* in one iteration. We are using the root of the DOM tree as context.

As soon as we find a *book* node we can use a relative path expression to access the author or other elements directly, always passing the current *book* node as context. XPath's *text()* function returns a set of text nodes.

XPath and DOM are not conflicting methods. On the contrary, XPath builds on DOM and simplifies the handling of XML documents. The DOM API with its multitude of functions and attributes for different types of nodes can often get quite confusing. The XPath API only uses the *Evaluate()* function, while the complexity has been transferred to the syntax of the XPath expressions.

4Suite

If you use Python for XML processing you might expect the PyXML package to be sufficient for any of the simple examples given here. Unfortunately that is not the case. XSLT support in PyXML is based on an older XSLT version of 4Suite from Four Thought, which contains errors.

For the XSLT part we are therefore using the current version 0.12.0a2 of 4Suite. The package is installed with *python setup.py install* using the familiar distutils tool. Like PyXML, 4Suite is a package

for XML processing under Python, but it covers a much larger range of functions than PyXML. Correspondingly, its API is more substantial and complex.

XSLT – transformation with rules

XSLT's approach is fundamentally different from that of XPath on its own. Here, the transformation of XML documents occurs via a number of transformation rules held in an XSLT stylesheet. A stylesheet is itself an XML document. Stylesheet rules use XPath expressions to reference nodes within an XML document.

The transformation is performed by an XSLT processor, which generates the relevant output from the XML file and the stylesheet (see Figure 1). The exact syntax and semantics of XSLT are relatively complex and have already been covered in Linux Magazine a few months ago. Additional information can be found at from the links in the Info box.

To transform the Python book XML file we are using the *transform.xslt* stylesheet. It consists primarily of three rules (*<xsl:template match="...">*) for matching *pythonbooks*, *book* and *author*, *title* and *publisher*. The rule *<xsl:apply-templates select="...">* instructs the XSLT processor to continue with a rule for the specified element. We are using *<xsl:value-of select=".">* to access the content of the text nodes.

As you can see, the XSLT processor is the central feature of any XSLT application, and 4Suite provides a separate *processor* class for this purpose. For XSLT processing the processor requires a reader that is able

Listing 2: xpath.py

```
01    from xml.dom.ext.reader.Sax2 import FromXmlStream
02    from xml.xpath import Evaluate
03    fp = open('pythonbooks.xml','r')
04    dom = FromXmlStream(fp)
05    fp.close()
06    print "<table>"
07    print "<tr>"
08    print "<th>Author(s)</th><th>Title</th> <th>Publisher</th>"
09    print "</tr>"
10    for book in Evaluate('book', dom .documentElement):
11        print "<tr>"
12       for item in ['author','title','publisher']:
13            path = '%s/text()' % item
14            print '<td>%s</td>' % Evaluate(path, book)[0].nodeValue
15        print "</tr>"
16    print "</table>"
```

Table 1: Important XPath expressions

Xpath syntax	Path expressions
/	The root node
.	Self node
..	Parent node
@attr	All attributes with the name "attr"
@*	All attributes
node	All elements with the name "node"
*	All elements
/node	All child elements with the name "node"
/*	All child elements
Xpath syntax	**Node set and string functions**
local-name()	Returns the local part of the expanded name of the node
name()	Returns the name of an element
string(obj)	Converts an object to a string
concat(s1,s2,..)	Returns the concatenation of its arguments
Xpath syntax	**Node set functions**
last()	Returns a number equal to the context size
position()	Returns a number equal to the context position
count()	Returns the number of selected elements
number(ojb)	Converts its argument to a number
sum(node-set)	Returns the sum, for each node in the argument node-set
string-length(s)	Returns the number of characters in a string
Xpath syntax	**Boolean functions**
startswith(s1,s2)	Returns true if the first string starts with the second string
contains(s1,s2)	Returns true if the first string contains the second string
boolean(obj)	Converts its agrument to Boolean
not(val)	Returns true if its argument is false, and false otherwise
true(),false()	Returns true, returns false

Listing 3: transform.xslt

```
01 <?xml version="1.0" encoding="iso-8859-1" ?>
02 <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
03 <xsl:output method="html" />
04 <xsl:template match="pythonbooks">
05   <table>
06     <tr>
07       <th>Author(s)</th><th>Title</th> <th>Publisher</th>
08         </tr>
09     <xsl:apply-templates select="book">
10       <xsl:sort select="author" />
11     </xsl:apply-templates>
12   </table>
13 </xsl:template>
14 <xsl:template match="book">
15   <tr>
16     <xsl:apply-templates select="author"/>
17     <xsl:apply-templates select="title"/>
18     <xsl:apply-templates select="publisher"/>
19   </tr>
20 </xsl:template>
21 <xsl:template match="author|publisher|title">
22   <td> <xsl:value-of select="."/> </td>
23 </xsl:template>
24 </xsl:stylesheet>
```

to read the stylesheet as well as the file to be transformed from the *InputSource*. An *InputSource* abstracts the input, so that the reader does not need to concern itself with the source.

In our example *xslt.py* (Listing 4) we are using a non-validating parser that is registered with the processor. The XSLT stylesheet is passed to the processor by means of the *appendStylesheet()* call. The processor itself is then started with *run()*. Output in our example is via the standard output.

Conclusion

This part shows that you can convert XML documents without changing the Python program. The Python program merely starts the transformation, which is the responsibility of the XSLT processor. In this, XPath is an integral component of XSLT.

XPath is an interesting alternative to DOM or SAX in cases where you require access to parts or nodes of XML documents without wanting to go to the trouble of writing a parser. In combination with XSLT this gives the developer a very powerful tool for processing XML files.

Listing 4: xslt.py

```
01 import sys, urllib
02 from Ft.Xml import InputSource, Domlette
03 from Ft.Xml.Xslt import Processor
04 xml  = urllib.pathname2url
("pythonbooks.xml")
05 xslt = urllib.pathname2url
("transform.xslt")
06 processor = Processor.Processor()
07 reader = Domlette.NonvalidatingReader
08 processor.setDocumentReader(reader)
09 isrc = InputSource.DefaultFactory.
fromUri(xslt)
10 processor.appendStylesheet(isrc)
11 isrc = InputSource.DefaultFactory.
fromUri(xml)
12 processor.run
(isrc,outputStream=sys.stdout)
```

Info

The Python XML module
http://pyxml.sourceforge.net
XPath recommendation
http://www.w3.org/TR/xpath
Four Thought *http://www.fourthought.com*
4Suite downloads
ftp://ftp.fourthought.com/pub/4Suite
XSLT and XPath tutorial
http://www.vbxml.com/xsl/tutorials/intro/default.asp
XSLT tutorial
http://www.zvon.org/xxl/XSLTutorial/Books/Book1/
C.A. Jones and F.L. Drake, Jr., *Python & XML* (O'Reilly, 2002)