Going Further With Ixml

lxml is an open source third-party library that builds on the popular libxml2 parser. It provides a 100% compatible ElementTree api, then extends it with full XPath 1.0 support and a few other niceties. There are installers available for Windows; Linux users should always try to use distribution-specific tools like yum or apt-get to install precompiled binaries from their repositories. Otherwise you'll need to install lxml manually.

```
from lxml import etree
tree = etree.parse('feed.xml')
root = tree.getroot()
print (root.findall('{http://www.w3.org/2005/Atom}entry'))
#[<Element {http://www.w3.org/2005/Atom}entry at e2b4e0>,
# <Element {http://www.w3.org/2005/Atom}entry at e2b510>,
# <Element {http://www.w3.org/2005/Atom}entry at e2b540>]
```

- ① Once imported, lxml provides the same api as the built-in ElementTree library.
- ② parse() function: same as ElementTree.
- ③ getroot() method: also the same.
- ④ findall() method: exactly the same.

For large XML documents, <code>lxml</code> is significantly faster than the built-in ElementTree library. If you're only using the ElementTree api and want to use the fastest available implementation, you can try to import <code>lxml</code> and fall back to the built-in ElementTree.

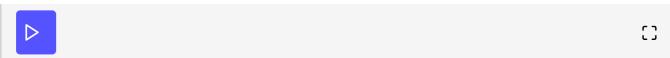
```
try:
    from lxml import etree
except ImportError:
```

```
import xml.etree.ElementTree as etree
```

But lxml is more than just a faster ElementTree. Its findall() method includes support for more complicated expressions.

```
import lxml.etree
tree = lxml.etree.parse('feed.xml')
print (tree.findall('//{http://www.w3.org/2005/Atom}*[@href]') )
#[<Element {http://www.w3.org/2005/Atom}link at eeb8a0>,
# <Element {http://www.w3.org/2005/Atom}link at eeb990>,
# <Element {http://www.w3.org/2005/Atom}link at eeb960>,
# <Element {http://www.w3.org/2005/Atom}link at eeb9co>]

print (tree.findall("//{http://www.w3.org/2005/Atom}*[@href='http://diveintomark.org/']") ) #
#[<Element {http://www.w3.org/2005/Atom}'
print (tree.findall("//{NS}author[{NS}uri]'.format(NS=NS)) )
#[<Element {http://www.w3.org/2005/Atom}author at eeb8a0>,
# <Element {http://www.w3.org/2005/Atom}author at eebba0>]
```



- ① In this example, I'm going to import lxml.etree (instead of, say, from lxml import etree), to emphasize that these features are specific to lxml.
- ② This query finds all elements in the Atom namespace, anywhere in the document, that have an href attribute. The // at the beginning of the query means "elements anywhere (not just as children of the root element)."

 {http://www.w3.org/2005/Atom} means "only elements in the Atom namespace." * means "elements with any local name." And [@href] means "has an href attribute."
- ③ The query finds all Atom elements with an href whose value is http://diveintomark.org/.
- ④ After doing some quick string formatting (because otherwise these compound queries get ridiculously long), this query searches for Atom author elements that have an Atom uri element as a child. This only returns two author elements, the ones in the first and second entry. The author in the last entry contains only a name, not a uri.

Not enough for you? 1xml also integrates support for arbitrary XPath 1.0 expressions. I'm not going to go into depth about XPath syntax; that could be a

whole book unto itself! But I will show you how it integrates into 1xml.

```
import lxml.etree
                                                                                            6
tree = lxml.etree.parse('feed.xml')
NSMAP = {'atom': 'http://www.w3.org/2005/Atom'}
                                                                               #1
entries = tree.xpath("//atom:category[@term='accessibility']/..",
                                                                               #2
     namespaces=NSMAP)
print (entries)
                                                                               #3
#[<Element {http://www.w3.org/2005/Atom}entry at e2b630>]
entry = entries[0]
print (entry.xpath('./atom:title/text()', namespaces=NSMAP))
                                                                               #4
#['Accessibility is a harsh mistress']
                                                                                             []
  \triangleright
```

- ① To perform XPath queries on namespaced elements, you need to define a namespace prefix mapping. This is just a Python dictionary.
- ② Here is an XPath query. The XPath expression searches for category elements (in the Atom namespace) that contain a term attribute with the value accessibility. But that's not actually the query result. Look at the very end of the query string; did you notice the /... bit? That means "and then return the parent element of the category element you just found." So this single XPath query will find all entries with a child element of <category term='accessibility'>.
- ③ The xpath() function returns a list of ElementTree objects. In this document, there is only one entry with a category whose term is accessibility.
- ④ XPath expressions don't always return a list of elements. Technically, the dom of a parsed XML document doesn't contain elements; it contains *nodes*. Depending on their type, nodes can be elements, attributes, or even text content. The result of an XPath query is a list of nodes. This query returns a list of text nodes: the text content (text()) of the title element (atom:title) that is a child of the current element (./).