<https://pymotw.com/2/xml/etree/ElementTree/parse.html>

# Parsing XML Documents

Parsed XML documents are represented in memory by ElementTree and Element objects connected into a tree structure based on the way the nodes in the XML document are nested.

## Parsing an Entire Document

Parsing an entire document with parse() returns an ElementTree instance. The tree knows about all of the data in the input document, and the nodes of the tree can be searched or manipulated in place. While this flexibility can make working with the parsed document a little easier, it typically takes more memory than an event-based parsing approach since the entire document must be loaded at one time.

The memory footprint of small, simple documents such as this list of podcasts represented as an [OPML](http://www.opml.org/) outline is not significant:

<?xml version="1.0" encoding="UTF-8"?>

<opml version="1.0">

<head>

<title>My Podcasts</title>

<dateCreated>Sun, 07 Mar 2010 15:53:26 GMT</dateCreated>

<dateModified>Sun, 07 Mar 2010 15:53:26 GMT</dateModified>

</head>

<body>

<outline text="Science and Tech">

<outline text="APM: Future Tense" type="rss"

xmlUrl="http://www.publicradio.org/columns/futuretense/podcast.xml"

htmlUrl="http://www.publicradio.org/columns/futuretense/" />

<outline text="Engines Of Our Ingenuity Podcast" type="rss"

xmlUrl="http://www.npr.org/rss/podcast.php?id=510030"

htmlUrl="http://www.uh.edu/engines/engines.htm" />

<outline text="Science &#38; the City" type="rss"

xmlUrl="http://www.nyas.org/Podcasts/Atom.axd"

htmlUrl="http://www.nyas.org/WhatWeDo/SciencetheCity.aspx" />

</outline>

<outline text="Books and Fiction">

<outline text="Podiobooker" type="rss"

xmlUrl="http://feeds.feedburner.com/podiobooks"

htmlUrl="http://www.podiobooks.com/blog" />

<outline text="The Drabblecast" type="rss"

xmlUrl="http://web.me.com/normsherman/Site/Podcast/rss.xml"

htmlUrl="http://web.me.com/normsherman/Site/Podcast/Podcast.html" />

<outline text="tor.com / category / tordotstories" type="rss"

xmlUrl="http://www.tor.com/rss/category/TorDotStories"

htmlUrl="http://www.tor.com/" />

</outline>

<outline text="Computers and Programming">

<outline text="MacBreak Weekly" type="rss"

xmlUrl="http://leo.am/podcasts/mbw"

htmlUrl="http://twit.tv/mbw" />

<outline text="FLOSS Weekly" type="rss"

xmlUrl="http://leo.am/podcasts/floss"

htmlUrl="http://twit.tv" />

<outline text="Core Intuition" type="rss"

xmlUrl="http://www.coreint.org/podcast.xml"

htmlUrl="http://www.coreint.org/" />

</outline>

<outline text="Python">

<outline text="PyCon Podcast" type="rss"

xmlUrl="http://advocacy.python.org/podcasts/pycon.rss"

htmlUrl="http://advocacy.python.org/podcasts/" />

<outline text="A Little Bit of Python" type="rss"

xmlUrl="http://advocacy.python.org/podcasts/littlebit.rss"

htmlUrl="http://advocacy.python.org/podcasts/" />

<outline text="Django Dose Everything Feed" type="rss"

xmlUrl="http://djangodose.com/everything/feed/" />

</outline>

<outline text="Miscelaneous">

<outline text="dhellmann's CastSampler Feed" type="rss"

xmlUrl="http://www.castsampler.com/cast/feed/rss/dhellmann/"

htmlUrl="http://www.castsampler.com/users/dhellmann/" />

</outline>

</body>

</opml>

To parse the file, pass an open file handle to parse().

from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:

tree = ElementTree.parse(f)

print tree

It will read the data, parse the XML, and return an ElementTree object.

$ python ElementTree\_parse\_opml.py

<xml.etree.ElementTree.ElementTree object at 0x10048cfd0>

## Traversing the Parsed Tree

To visit all of the children in order, use iter() to create a generator that iterates over the ElementTree instance.

from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:

tree = ElementTree.parse(f)

for node in tree.iter():

print node.tag, node.attrib

This example prints the entire tree, one tag at a time.

$ python ElementTree\_dump\_opml.py

opml {'version': '1.0'}

head {}

title {}

dateCreated {}

dateModified {}

body {}

outline {'text': 'Science and Tech'}

outline {'xmlUrl': 'http://www.publicradio.org/columns/futuretense/podcast.xml', 'text': 'APM: Future Tense', 'type': 'rss', 'htmlUrl': 'http://www.publicradio.org/columns/futuretense/'}

outline {'xmlUrl': 'http://www.npr.org/rss/podcast.php?id=510030', 'text': 'Engines Of Our Ingenuity Podcast', 'type': 'rss', 'htmlUrl': 'http://www.uh.edu/engines/engines.htm'}

outline {'xmlUrl': 'http://www.nyas.org/Podcasts/Atom.axd', 'text': 'Science & the City', 'type': 'rss', 'htmlUrl': 'http://www.nyas.org/WhatWeDo/SciencetheCity.aspx'}

outline {'text': 'Books and Fiction'}

outline {'xmlUrl': 'http://feeds.feedburner.com/podiobooks', 'text': 'Podiobooker', 'type': 'rss', 'htmlUrl': 'http://www.podiobooks.com/blog'}

outline {'xmlUrl': 'http://web.me.com/normsherman/Site/Podcast/rss.xml', 'text': 'The Drabblecast', 'type': 'rss', 'htmlUrl': 'http://web.me.com/normsherman/Site/Podcast/Podcast.html'}

outline {'xmlUrl': 'http://www.tor.com/rss/category/TorDotStories', 'text': 'tor.com / category / tordotstories', 'type': 'rss', 'htmlUrl': 'http://www.tor.com/'}

outline {'text': 'Computers and Programming'}

outline {'xmlUrl': 'http://leo.am/podcasts/mbw', 'text': 'MacBreak Weekly', 'type': 'rss', 'htmlUrl': 'http://twit.tv/mbw'}

outline {'xmlUrl': 'http://leo.am/podcasts/floss', 'text': 'FLOSS Weekly', 'type': 'rss', 'htmlUrl': 'http://twit.tv'}

outline {'xmlUrl': 'http://www.coreint.org/podcast.xml', 'text': 'Core Intuition', 'type': 'rss', 'htmlUrl': 'http://www.coreint.org/'}

outline {'text': 'Python'}

outline {'xmlUrl': 'http://advocacy.python.org/podcasts/pycon.rss', 'text': 'PyCon Podcast', 'type': 'rss', 'htmlUrl': 'http://advocacy.python.org/podcasts/'}

outline {'xmlUrl': 'http://advocacy.python.org/podcasts/littlebit.rss', 'text': 'A Little Bit of Python', 'type': 'rss', 'htmlUrl': 'http://advocacy.python.org/podcasts/'}

outline {'xmlUrl': 'http://djangodose.com/everything/feed/', 'text': 'Django Dose Everything Feed', 'type': 'rss'}

outline {'text': 'Miscelaneous'}

outline {'xmlUrl': 'http://www.castsampler.com/cast/feed/rss/dhellmann/', 'text': "dhellmann's CastSampler Feed", 'type': 'rss', 'htmlUrl': 'http://www.castsampler.com/users/dhellmann/'}

To print only the groups of names and feed URLs for the podcasts, leaving out of all of the data in the header section by iterating over only the outline nodes and print the text and xmlUrl attributes.

from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:

tree = ElementTree.parse(f)

for node in tree.iter('outline'):

name = node.attrib.get('text')

url = node.attrib.get('xmlUrl')

if name and url:

print ' %s :: %s' % (name, url)

else:

print name

The 'outline' argument to iter() means processing is limited to only nodes with the tag 'outline'.

$ python ElementTree\_show\_feed\_urls.py

Science and Tech

APM: Future Tense :: http://www.publicradio.org/columns/futuretense/podcast.xml

Engines Of Our Ingenuity Podcast :: http://www.npr.org/rss/podcast.php?id=510030

Science & the City :: http://www.nyas.org/Podcasts/Atom.axd

Books and Fiction

Podiobooker :: http://feeds.feedburner.com/podiobooks

The Drabblecast :: http://web.me.com/normsherman/Site/Podcast/rss.xml

tor.com / category / tordotstories :: http://www.tor.com/rss/category/TorDotStories

Computers and Programming

MacBreak Weekly :: http://leo.am/podcasts/mbw

FLOSS Weekly :: http://leo.am/podcasts/floss

Core Intuition :: http://www.coreint.org/podcast.xml

Python

PyCon Podcast :: http://advocacy.python.org/podcasts/pycon.rss

A Little Bit of Python :: http://advocacy.python.org/podcasts/littlebit.rss

Django Dose Everything Feed :: http://djangodose.com/everything/feed/

Miscelaneous

dhellmann's CastSampler Feed :: http://www.castsampler.com/cast/feed/rss/dhellmann/

## Finding Nodes in a Document

Walking the entire tree like this searching for relevant nodes can be error prone. The example above had to look at each outline node to determine if it was a group (nodes with only a text attribute) or podcast (with both text and xmlUrl). To produce a simple list of the podcast feed URLs, without names or groups, for a podcast downloader application, the logic could be simplified using findall() to look for nodes with more descriptive search characteristics.

As a first pass at converting the above example, we can construct an [XPath](http://www.w3.org/TR/xpath/) argument to look for all outline nodes.

from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:

tree = ElementTree.parse(f)

for node in tree.findall('.//outline'):

url = node.attrib.get('xmlUrl')

if url:

print url

The logic in this version is not substantially different than the version using getiterator(). It still has to check for the presence of the URL, except that it does not print the group name when the URL is not found.

$ python ElementTree\_find\_feeds\_by\_tag.py

http://www.publicradio.org/columns/futuretense/podcast.xml

http://www.npr.org/rss/podcast.php?id=510030

http://www.nyas.org/Podcasts/Atom.axd

http://feeds.feedburner.com/podiobooks

http://web.me.com/normsherman/Site/Podcast/rss.xml

http://www.tor.com/rss/category/TorDotStories

http://leo.am/podcasts/mbw

http://leo.am/podcasts/floss

http://www.coreint.org/podcast.xml

http://advocacy.python.org/podcasts/pycon.rss

http://advocacy.python.org/podcasts/littlebit.rss

http://djangodose.com/everything/feed/

http://www.castsampler.com/cast/feed/rss/dhellmann/

Another version can take advantage of the fact that the outline nodes are only nested two levels deep. Changing the search path to .//outline/outline mean the loop will process only the second level of outline nodes.

from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:

tree = ElementTree.parse(f)

for node in tree.findall('.//outline/outline'):

url = node.attrib.get('xmlUrl')

print url

All of those outline nodes nested two levels deep in the input are expected to have the xmlURL attribute refering to the podcast feed, so the loop can skip checking for for the attribute before using it.

$ python ElementTree\_find\_feeds\_by\_structure.py

http://www.publicradio.org/columns/futuretense/podcast.xml

http://www.npr.org/rss/podcast.php?id=510030

http://www.nyas.org/Podcasts/Atom.axd

http://feeds.feedburner.com/podiobooks

http://web.me.com/normsherman/Site/Podcast/rss.xml

http://www.tor.com/rss/category/TorDotStories

http://leo.am/podcasts/mbw

http://leo.am/podcasts/floss

http://www.coreint.org/podcast.xml

http://advocacy.python.org/podcasts/pycon.rss

http://advocacy.python.org/podcasts/littlebit.rss

http://djangodose.com/everything/feed/

http://www.castsampler.com/cast/feed/rss/dhellmann/

This version is limited to the existing structure, though, so if the outline nodes are ever rearranged into a deeper tree it will stop working.

## Parsed Node Attributes

The items returned by findall() and iter() are Element objects, each representing a node in the XML parse tree. Each Element has attributes for accessing data pulled out of the XML. This can be illustrated with a somewhat more contrived example input file, data.xml:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?xml version="1.0" encoding="UTF-8"?>  <top>  <child>This child contains text.</child>  <child\_with\_tail>This child has regular text.</child\_with\_tail>And "tail" text.  <with\_attributes name="value" foo="bar" />  <entity\_expansion attribute="This &#38; That">That &#38; This</entity\_expansion>  </top> |

The “attributes” of a node are available in the attrib property, which acts like a dictionary.

from xml.etree import ElementTree

with open('data.xml', 'rt') as f:

tree = ElementTree.parse(f)

node = tree.find('./with\_attributes')

print node.tag

for name, value in sorted(node.attrib.items()):

print ' %-4s = "%s"' % (name, value)

The node on line five of the input file has two attributes, name and foo.

$ python ElementTree\_node\_attributes.py

with\_attributes

foo = "bar"

name = "value"

The text content of the nodes is available, along with the “tail” text that comes after the end of a close tag.

from xml.etree import ElementTree

with open('data.xml', 'rt') as f:

tree = ElementTree.parse(f)

for path in [ './child', './child\_with\_tail' ]:

node = tree.find(path)

print node.tag

print ' child node text:', node.text

print ' and tail text :', node.tail

The child node on line three contains embedded text, and the node on line four has text with a tail (including any whitespace).

$ python ElementTree\_node\_text.py

child

child node text: This child contains text.

and tail text :

child\_with\_tail

child node text: This child has regular text.

and tail text : And "tail" text.

XML entity references embedded in the document are conveniently converted to the appropriate characters before values are returned.

from xml.etree import ElementTree

with open('data.xml', 'rt') as f:

tree = ElementTree.parse(f)

node = tree.find('entity\_expansion')

print node.tag

print ' in attribute:', node.attrib['attribute']

print ' in text :', node.text

The automatic conversion mean the implementation detail of representing certain characters in an XML document can be ignored.

$ python ElementTree\_entity\_references.py

entity\_expansion

in attribute: This & That

in text : That & This

## Watching Events While Parsing

The other API useful for processing XML documents is event-based. The parser generates start events for opening tags and end events for closing tags. Data can be extracted from the document during the parsing phase by iterating over the event stream, which is convenient if it is not necessary to manipulate the entire document afterwards and there is no need to hold the entire parsed document in memory.

iterparse() returns an iterable that produces tuples containing the name of the event and the node triggering the event. Events can be one of:

start

A new tag has been encountered. The closing angle bracket of the tag was processed, but not the contents.

end

The closing angle bracket of a closing tag has been processed. All of the children were already processed.

start-ns

Start a namespace declaration.

end-ns

End a namespace declaration.

from xml.etree.ElementTree import iterparse

depth = 0

prefix\_width = 8

prefix\_dots = '.' \* prefix\_width

line\_template = '{prefix:<0.{prefix\_len}}{event:<8}{suffix:<{suffix\_len}} {node.tag:<12} {node\_id}'

for (event, node) in iterparse('podcasts.opml', ['start', 'end', 'start-ns', 'end-ns']):

if event == 'end':

depth -= 1

prefix\_len = depth \* 2

print line\_template.format(prefix=prefix\_dots,

prefix\_len=prefix\_len,

suffix='',

suffix\_len=(prefix\_width - prefix\_len),

node=node,

node\_id=id(node),

event=event,

)

if event == 'start':

depth += 1

By default, only end events are generated. To see other events, pass the list of desired event names to iterparse(), as in this example:

$ python ElementTree\_show\_all\_events.py

start opml 4299786128

..start head 4299786192

....start title 4299786256

....end title 4299786256

....start dateCreated 4299786448

....end dateCreated 4299786448

....start dateModified 4299786640

....end dateModified 4299786640

..end head 4299786192

..start body 4299787024

....start outline 4299787088

......start outline 4299787152

......end outline 4299787152

......start outline 4299787216

......end outline 4299787216

......start outline 4299787280

......end outline 4299787280

....end outline 4299787088

....start outline 4299787344

......start outline 4299787472

......end outline 4299787472

......start outline 4299787408

......end outline 4299787408

......start outline 4299787536

......end outline 4299787536

....end outline 4299787344

....start outline 4299787600

......start outline 4299787728

......end outline 4299787728

......start outline 4299787920

......end outline 4299787920

......start outline 4299787856

......end outline 4299787856

....end outline 4299787600

....start outline 4299788048

......start outline 4299788112

......end outline 4299788112

......start outline 4299788176

......end outline 4299788176

......start outline 4299792464

......end outline 4299792464

....end outline 4299788048

....start outline 4299792592

......start outline 4299792720

......end outline 4299792720

....end outline 4299792592

..end body 4299787024

end opml 4299786128

The event-style of processing is more natural for some operations, such as converting XML input to some other format. This technique can be used to convert list of podcasts from the earlier examples from an XML file to a CSV file, so they can be loaded into a spreadsheet or database application.

import csv

from xml.etree.ElementTree import iterparse

import sys

writer = csv.writer(sys.stdout, quoting=csv.QUOTE\_NONNUMERIC)

group\_name = ''

for (event, node) in iterparse('podcasts.opml', events=['start']):

if node.tag != 'outline':

# Ignore anything not part of the outline

continue

if not node.attrib.get('xmlUrl'):

# Remember the current group

group\_name = node.attrib['text']

else:

# Output a podcast entry

writer.writerow( (group\_name, node.attrib['text'],

node.attrib['xmlUrl'],

node.attrib.get('htmlUrl', ''),

)

)

This conversion program does not need to hold the entire parsed input file in memory, and processing each node as it is encountered in the input is more efficient.

$ python ElementTree\_write\_podcast\_csv.py

"Science and Tech","APM: Future Tense","http://www.publicradio.org/columns/futuretense/podcast.xml","http://www.publicradio.org/columns/futuretense/"

"Science and Tech","Engines Of Our Ingenuity Podcast","http://www.npr.org/rss/podcast.php?id=510030","http://www.uh.edu/engines/engines.htm"

"Science and Tech","Science & the City","http://www.nyas.org/Podcasts/Atom.axd","http://www.nyas.org/WhatWeDo/SciencetheCity.aspx"

"Books and Fiction","Podiobooker","http://feeds.feedburner.com/podiobooks","http://www.podiobooks.com/blog"

"Books and Fiction","The Drabblecast","http://web.me.com/normsherman/Site/Podcast/rss.xml","http://web.me.com/normsherman/Site/Podcast/Podcast.html"

"Books and Fiction","tor.com / category / tordotstories","http://www.tor.com/rss/category/TorDotStories","http://www.tor.com/"

"Computers and Programming","MacBreak Weekly","http://leo.am/podcasts/mbw","http://twit.tv/mbw"

"Computers and Programming","FLOSS Weekly","http://leo.am/podcasts/floss","http://twit.tv"

"Computers and Programming","Core Intuition","http://www.coreint.org/podcast.xml","http://www.coreint.org/"

"Python","PyCon Podcast","http://advocacy.python.org/podcasts/pycon.rss","http://advocacy.python.org/podcasts/"

"Python","A Little Bit of Python","http://advocacy.python.org/podcasts/littlebit.rss","http://advocacy.python.org/podcasts/"

"Python","Django Dose Everything Feed","http://djangodose.com/everything/feed/",""

"Miscelaneous","dhellmann's CastSampler Feed","http://www.castsampler.com/cast/feed/rss/dhellmann/","http://www.castsampler.com/users/dhellmann/"

## Creating a Custom Tree Builder

A potentially more efficient means of handling parse events is to replace the standard tree builder behavior with a custom version. The ElementTree parser uses an XMLTreeBuilder to process the XML and call methods on a target class to save the results. The usual output is an ElementTree instance created by the default TreeBuilder class. Replacing TreeBuilder with another class allows it to receive the events before the Element nodes are instantiated, saving that portion of the overhead.

The XML-to-CSV converter from the previous section can be translated to a tree builder.

import csv

from xml.etree.ElementTree import XMLTreeBuilder

import sys

class PodcastListToCSV(object):

def \_\_init\_\_(self, outputFile):

self.writer = csv.writer(outputFile, quoting=csv.QUOTE\_NONNUMERIC)

self.group\_name = ''

return

def start(self, tag, attrib):

if tag != 'outline':

# Ignore anything not part of the outline

return

if not attrib.get('xmlUrl'):

# Remember the current group

self.group\_name = attrib['text']

else:

# Output a podcast entry

self.writer.writerow( (self.group\_name, attrib['text'],

attrib['xmlUrl'],

attrib.get('htmlUrl', ''),

)

)

def end(self, tag):

# Ignore closing tags

pass

def data(self, data):

# Ignore data inside nodes

pass

def close(self):

# Nothing special to do here

return

target = PodcastListToCSV(sys.stdout)

parser = XMLTreeBuilder(target=target)

with open('podcasts.opml', 'rt') as f:

for line in f:

parser.feed(line)

parser.close()

PodcastListToCSV implements the TreeBuilder protocol. Each time a new XML tag is encountered, start() is called with the tag name and attributes. When a closing tag is seen end() is called with the name. In between, data() is called when a node has content (the tree builder is expected to keep up with the “current” node). When all of the input is processed, close() is called. It can return a value, which will be returned to the user of the XMLTreeBuilder.

$ python ElementTree\_podcast\_csv\_treebuilder.py

"Science and Tech","APM: Future Tense","http://www.publicradio.org/columns/futuretense/podcast.xml","http://www.publicradio.org/columns/futuretense/"

"Science and Tech","Engines Of Our Ingenuity Podcast","http://www.npr.org/rss/podcast.php?id=510030","http://www.uh.edu/engines/engines.htm"

"Science and Tech","Science & the City","http://www.nyas.org/Podcasts/Atom.axd","http://www.nyas.org/WhatWeDo/SciencetheCity.aspx"

"Books and Fiction","Podiobooker","http://feeds.feedburner.com/podiobooks","http://www.podiobooks.com/blog"

"Books and Fiction","The Drabblecast","http://web.me.com/normsherman/Site/Podcast/rss.xml","http://web.me.com/normsherman/Site/Podcast/Podcast.html"

"Books and Fiction","tor.com / category / tordotstories","http://www.tor.com/rss/category/TorDotStories","http://www.tor.com/"

"Computers and Programming","MacBreak Weekly","http://leo.am/podcasts/mbw","http://twit.tv/mbw"

"Computers and Programming","FLOSS Weekly","http://leo.am/podcasts/floss","http://twit.tv"

"Computers and Programming","Core Intuition","http://www.coreint.org/podcast.xml","http://www.coreint.org/"

"Python","PyCon Podcast","http://advocacy.python.org/podcasts/pycon.rss","http://advocacy.python.org/podcasts/"

"Python","A Little Bit of Python","http://advocacy.python.org/podcasts/littlebit.rss","http://advocacy.python.org/podcasts/"

"Python","Django Dose Everything Feed","http://djangodose.com/everything/feed/",""

"Miscelaneous","dhellmann's CastSampler Feed","http://www.castsampler.com/cast/feed/rss/dhellmann/","http://www.castsampler.com/users/dhellmann/"

## Parsing Strings

To work with smaller bits of XML text, especially string literals as might be embedded in the source of a program, use XML() and the string containing the XML to be parsed as the only argument.

from xml.etree.ElementTree import XML

parsed = XML('''

<root>

<group>

<child id="a">This is child "a".</child>

<child id="b">This is child "b".</child>

</group>

<group>

<child id="c">This is child "c".</child>

</group>

</root>

''')

print 'parsed =', parsed

for elem in parsed:

print elem.tag

if elem.text is not None and elem.text.strip():

print ' text: "%s"' % elem.text

if elem.tail is not None and elem.tail.strip():

print ' tail: "%s"' % elem.tail

for name, value in sorted(elem.attrib.items()):

print ' %-4s = "%s"' % (name, value)

print

Notice that unlike with parse(), the return value is an Element instance instead of an ElementTree. An Element supports the iterator protocol directly, so there is no need to call getiterator().

$ python ElementTree\_XML.py

parsed = <Element 'root' at 0x100497710>

group

group

For structured XML that uses the id attribute to identify unique nodes of interest, XMLID() is a convenient way to access the parse results.

from xml.etree.ElementTree import XMLID

tree, id\_map = XMLID('''

<root>

<group>

<child id="a">This is child "a".</child>

<child id="b">This is child "b".</child>

</group>

<group>

<child id="c">This is child "c".</child>

</group>

</root>

''')

for key, value in sorted(id\_map.items()):

print '%s = %s' % (key, value)

XMLID() returns the parsed tree as an Element object, along with a dictionary mapping the id attribute strings to the individual nodes in the tree.

$ python ElementTree\_XMLID.py

a = <Element 'child' at 0x100497850>

b = <Element 'child' at 0x100497910>

c = <Element 'child' at 0x100497b50>

See also

Outline Processor Markup Language, [OPML](http://www.opml.org/)

Dave Winer’s OPML specification and documentation.

XML Path Language, [XPath](http://www.w3.org/TR/xpath/)

A syntax for identifying parts of an XML document.

[XPath Support in ElementTree](http://effbot.org/zone/element-xpath.htm)

Part of Fredrick Lundh’s original documentation for ElementTree.

[csv](https://pymotw.com/2/csv/index.html#module-csv)

Read and write comma-separated-value files