PicXml

Version : 1.0
Date : 19/08/2006
Author : ¨Patrick Germain Placidoux

SUMMARY

1 INTRODUCTION

Picxml is a Jython/Python tools for xml handling.

It is an xml parser and a lite xpath like xml request engine to ease your life manipulating complexe xml files.

Most of the time, dealing with xml file, we need to address this specific node with this tag and this specific set of attributes/values.

Less likely we need to address this node with this tag and/or maybe this other one with eventually this set of atrributes/values matching this regular expression or not.

If it is the last kind of request you ae expecting support for, please switch to a more advanced feature like full xpath supporting tools.

Picmlx is straigthforward either as a command or as an inline jython/python module.

It use a human syntax you wont forget.

2 PREREQUISITES

Oses (every where python works)

Linux, AIX ®, Windows ®

Langages

Python >= 2.4 < 3
Jython >= 2.5

3 INSTALLATION

Download picxml at www.sourceforge.net

Unzip the file picxml_#.##.zip in the directory of your choice.

Put the path <PICXML_INSTALL_PATH>/bin in your path environment variable.

Type :

> picxml on Windows
or
> picxml.sh on Unixes
(dont forget to run chmod ugo+x *.sh in the bin directory)

4 QUICK VIEW

4.1 USING COMMAND LINE PXQUERY

4.1.1 *Top down request*

Syntax
TAG[/TAG]@ATTR
**TAG[/TAG]@ATTR[,ATTR]\n' + **
TAG[/TAG]@*'

Each elements (except the last one) of the patern is a tag name,
the last element is the returned attribute name, a list of returned attributes, or * for all attributes.

Given this xml file:

```
<tag1 attr1="a" attr2="b">

        <tag2 attr1="c"  attr2="d"  attr3="e"/>
        <tag3 attr1="f"  attr2="g" attr3="A value with spaces !"/>

        <tag2 attr1="h"  attr2="i"  attr3="j"  attr4="k">
                <tag4>
                        <tag5 attr1="l"  attr2="m"  attr3="n"/>
                </tag4>
        </tag2>

        <tag2 attr1="o"  attr2="p"/>
</tag1>
```

> picxml test.xml -t tag1@attr1
a

> picxml test.xml -t tag1@attr2
b

> picxml test.xml -t tag1/tag3@attr2
g

> picxml test.xml -t tag1/tag2@attr1
c,h,o

Returning a list of attributes

> picxml test.xml -t tag1/tag2@attr1,@attr2
attr1:c attr2:d,attr1:h attr2:i,attr1:o attr2:p

Returning all attributes

> picxml test.xml -t tag1/tag2@*
attr1:c attr2:d attr3:e,attr1:h attr2:i attr3:j attr4:k,attr1:o attr2:p

4.1.2 Top down complete request

Syntax
TAG[@ATTR=VALUE[,@ATTR=VALUE]],@ATTR Tag sep:/
TAG[@ATTR=VALUE[,@ATTR=VALUE]],@ATTR[,@ATTR]
TAG[@ATTR=VALUE[,@ATTR=VALUE]],@*

This syntax let you to specify attribute/values for intermediate tags.

Each "/" separates a tag level.
Each "/" is follown by a tag name.
A tag name is follown by a "/" than another tag,
or is follown by "@".
Each "@" is followed by an attribute name.
Intermediates attribute names are folown by "=" then
their value.
The expression always finishes by an attribute name,
or a list of attributes to retreive or a "*".

Given the same xml file (repeated for convenience):

```
<tag1 attr1="a" attr2="b">

        <tag2 attr1="c"  attr2="d"  attr3="e"/>
        <tag3 attr1="f"  attr2="g" attr3='"A value with spaces !"/>

        <tag2 attr1="h"  attr2="i"  attr3="j"  attr4="k">
                <tag4>
                        <tag5 attr1="l"  attr2="m"  attr3="n"/>
                </tag4>
        </tag2>

        <tag2 attr1="o"  attr2="p"/>
</tag1>
```

> picxml test.xml -T tag1@attr1 (<=> -t tag1@attr1)
a

> **picxml test.xml -T tag1@attr2**
b

> picxml test.xml -T tag1/tag3@attr2 (<=> -t tag1.tag3.attr2)
g

> picxml test.xml -T tag1/tag2@attr1 (<=> -t tag1.tag2.attr1)
c,h,o

>picxml test.xml -T tag1/tag2@attr2=p,@attr1
o

More interesting:

> picxml test.xml -T tag1/tag2@attr2=d,@attr1
c

> picxml test.xml -T tag1/tag2@attr2=p,@attr1
o

> picxml test.xml -T tag1/tag2@attr2=i/tag4/tag5@attr2
m

Returning a list of attributes

picxml test.xml -T tag1/tag2@attr2=i/tag4/tag5@attr1,@attr2
attr1:l attr2:m

Returning all attributes

picxml test.xml -T tag1/tag2@attr2=i/tag4/tag5@*
attr1:l attr2:m attr3:n

picxml test.xml -T tag1/tag2@attr2=p,@*
attr1:o attr2:p

4.1.3 Working with more than one file

With this guiven text file (test.txt) :
test.xml
test1.xml
test2.xml

> picxml test.xml test1.xml test2.xml -t tag1/tag2@attr1,@attr2
test.xml:attr1:c attr2:d,attr1:h attr2:i,attr1:o attr2:p
test1.xml:attr1:c1 attr2:d1,attr1:h1 attr2:i1,attr1:o1 attr2:p1
test2.xml:attr1:c2 attr2:d2,attr1:h2 attr2:i2,attr1:o2 attr2:p2

4.1.4 Working with pipes

On Windows

> type test.txt | picxml -t tag1/tag2@attr1,@attr2

test.xml
test.xml:attr1:c attr2:d,attr1:h attr2:i,attr1:o attr2:p
test1.xml
test1.xml:attr1:c1 attr2:d1,attr1:h1 attr2:i1,attr1:o1 attr2:p1
test2.xml
test2.xml:attr1:c2 attr2:d2,attr1:h2 attr2:i2,attr1:o2 attr2:p2

On Unixes

> cat test.txt | picxml -t tag1/tag2@attr1,@attr2

test.xml:attr1:c attr2:d,attr1:h attr2:i,attr1:o attr2:p
test1.xml:attr1:c1 attr2:d1,attr1:h1 attr2:i1,attr1:o1 attr2:p1
test2.xml:attr1:c2 attr2:d2,attr1:h2 attr2:i2,attr1:o2 attr2:p2

4.1.5 Working with separators

Default attribute separator is " ".

Attribute separator

>picxml test.xml -T tag1/tag2@attr2=i/tag4/tag5@attr1,@attr2,@attr3 -s " "
attr1:l attr2:m attr3:n

>picxml test.xml -T tag1/tag2@attr2=i/tag4/tag5@attr1,@attr2,@attr3 -s %
attr1:l%attr2:m%attr3:n

Node separator

Default node separator is ","

> picxml test.xml -t tag1/tag2@attr2 -S ;
d;i;p

> picxml test.xml -t tag1/tag2@attr1,@attr2 -S !
attr1:c attr2:d!attr1:h attr2:i!attr1:o attr2:p

4.1.6 Working with spaces

> picxml test.xml -T "tag1/tag3@attr3=A value with spaces !,@attr1"
f

> picxml test.xml -T "tag1/tag3@attr1=f,@attr3"
A value with spaces !

4.1.7 Restrictions

In command mode each node must be found once per level except the last one that can be multiple.
e.g.:

-t tag1/tag2/tag3/tag4@attr3

Only one instance of tag1, tag2 and tag3 must be found, otherwise an exception is thrown.
(to avoid this use tdc instead to make the difference on attritbutes)
Multiple instances of tag4 is allowed.

4.2 USING PICXML AS A MODULE INTO JYTHON/PYTHON

```
>>> import picxmlp
>>> top=picxmlp.getTopNode('/where/is/my/xml/file')
```

This returns an instance of class Node.

4.2.1 Using the td query method

- **Working with attributes**

The following is quivalent to picxml test.xml -t tag1.tag2.attr1:

```
>>> req='tag1.tag2.attr2'
>>> print top.td(req, checkIsNode=False, checkIsAttr=True)
d,i,p
```

- **Working with nodes**

```
>>> req='tag1.tag2'
>>> nodes=top.td(req, checkIsNode=True, checkIsAttr=False)
>>>for node in nodes:print node.getAttrs().attr2
d
i
p

>>>for node in nodes:print node.getdAttrs()
{'attr2': 'd', 'attr3': 'e', 'attr1': 'c'}
{'attr4': 'k', 'attr2': 'i', 'attr3': 'j', 'attr1': 'h'}
{'attr2': 'p', 'attr1': 'o'}

>>>for node in nodes:print node.getAttrs().attr2
```

Changing Attribute values:

```
>>> nodes[2].setAttr('attr2', 'new_value')
>>> for node in nodes:print node.getAttrs().attr2
d
i
new_value
```

checkIsNode=True: checks that the last item of the request is a node name and returns the (or a list of) node(s).

- **Advanced parameters**

checkIsUnique: (default True)
Each node must be found once per level except the last one that can be multiple.
If True the last level neither can be multiple.
If false a list of matching nodes is returned.

attr_separator (Working with attributes only)
When returned a list of attributes, the attributes separator.

node_separator (Working with attributes only)
When found multiple mathing nodes, the nodes separator.

4.2.2 Using the tdc query method

- **Working with attribute values**

> python
The following is equivalent to picxml test.xml -T tag1/tag2@attr1:
>>> req='tag1/tag2@attr1'
>>> print top.tdc(req, checkIsNode=False, checkIsAttr=True)
c,h,o

- **Working with nodes**

>>> **req='tag1/tag2'**
>>> nodes=top.tdc(req, checkIsNode=True, checkIsAttr=False)
>>>for node in nodes:print node.getAttrs().attr2
d
i
p

>>>for node in nodes:print node.getdAttrs()
{'attr2': 'd', 'attr3': 'e', 'attr1': 'c'}
{'attr4': 'k', 'attr2': 'i', 'attr3': 'j', 'attr1': 'h'}
{'attr2': 'p', 'attr1': 'o'}

Changing Attribute values:

>>> nodes[2].setAttr('attr2', 'new_value')
>>> for node in nodes:print node.getAttrs().attr2
d
i
new_value

- **Advanced parameters**

checkIsUnique: (default True)
Each node must be found once per level except the last one that can be multiple.
If True the last level neither can be multiple.
If false a list of matching nodes is returned.

attr_separator (Working with attributes only)
When returned a list of attributes, the attributes separator.

node_separator (Working with attributes only)
When found multiple mathing nodes, the nodes separator.

4.2.3 Using the geNode method

On a guiven node, returns a list of the child nodes, matching a tag name.

```
>>> nodes=top.getNode('tag2')
>>> for node in nodes:print node.getAttrs().attr2
d
i
new_value
```

The getNode is faster than PxQuery, because there is no need to parse any request line.

4.2.4 Using the get method

On a guiven node, returns a list of the child nodes found at this level, matching a tag name and an optional list of attribute/value pairs.

```
>>> print top.get('tag2', attr3='j', attr4='k')[0].getAttrs().attr1
h
```

The get method is faster than PxQuery, because there is no need to parse any request line.

4.2.5 General rule on Node methods

The Node methods : td, tdc, get, … can be called on any node instance at any level of the tree.
They do not apply only on he top Node.
Actually the current node is considerated as being the top level.

4.3 USING PICXML MODULE AS A REGULAR (BUT RICH) XML PARSER

- An exemple walking througth nodes and childs and printing the xml content

```
> python
>>>import picxmlp
>>>top=picxmlp.getTopNode('/where/is/my/xml/file')
>>>def prt(node,  indent=0):
    attrs=node.getdAttrs()
    print ' '*indent,  'tag:', node.getTag()
    for attr in attrs.keys():print ' '*(indent+2),  'attr :', attrs[attr]
    for child in node.getNodes():prt(child, indent=indent+4)
>>>prt(top)
```

- An exemple creating a node tree from scratch

```
>>>import picxmlp

>>> d=[
    {'_tag_': 'tag', 'attr1':'a', 'attr2':'b',
        '_childs_': [
            {'_tag_': 'tag2', 'attr1':'c', 'attr2':'d', 'attr3':'e'},
            {'_tag_': 'tag3', 'attr1':'f', 'attr2':'g',
                '_childs_': [
                    {'_tag_': 'tag2', 'attr1':'h', 'attr2':'i', 'attr3':'j', 'attr4':'k',
                        '_childs_': [
                            {'_tag_': 'tag4',
                                '_childs_': [{'_tag_': 'tag5', 'attr1':'l', 'attr2':'m', 'attr3':'n'}]
                            }
                        ]
                    }
                ]
            },
            {'_tag_': 'tag2', 'attr1':'o',  'attr2':'p'}
        ]
    }
]


>>> def newNode(childs, parent_node=None):

    for node_attrs in childs:
        attrs={} # Protects the og dict firts.
        attrs.update(node_attrs)
```

```
        tag=attrs['_tag_'] # Retreives the tag name.
        del attrs['_tag_']
        this_childs=None
        if attrs.has_key('_childs_'):  # Retreives the child nodes.
            this_childs=attrs['_childs_']
            del attrs['_childs_']

        ## Making a new Node.
        # Reminding attributes in attrs are the tag attributes.
        if parent_node==None:new_node=picxmlp.Node(tag, attrs=attrs)
        else:new_node=parent_node.newNode(tag, **attrs)

        # If has childs recurse on them.
        if this_childs!=None:newNode(this_childs, parent_node=new_node)

    return new_node # Returns the top node.
```

>>> top=newNode(d)
>>>print top.printXml().getvalue()

4.4 CLASS NODE REFERENCE

4.4.1 *Utility methods*

def getTag(self):
Returns this node tag name.

def getTop(self):
Returns the xml top node.

def getNodes(self):
Returns this node child nodes.

def newNode(self, tag, **attrs):
Instanciates a new object of class Node,
adds it to this node and returns it.
tag: a tag name.
**attrs: is a set of pair attr=value keyword parameters.
Ex:

```
top=Node(name='mytop')
top.newNode('mytag', attr1='value1', attr2='value2', attr3='value3')
```

def add(self, node):
Add a new Node object to this node.

def adds(self, nodes):
Adds a python list of new Node objects to this node.

def remove(self, node):
Removes this guiven node from its parent (self).

def getAttrs(self):
Returns an object which attributes are the node's tag attributes.

def getdAttrs(self):
Returns a dict which entries are the node's tag attributes.

def hasAttr(self, attr):
Checks if this node support this tag attribute.

def getAttr(self, attr):
Returns this node's tag attribute value.

def setAttr(self, attr, value):
Set value to this node's tag attribute: attr.

def setAttrs(self, **attrs):
Set value to this node's tag attribute: attr.

**attrs: is a set of pair attr=value keyword parameters.

def hasText(self):
Checks if this node has texts.

def getText(self):
Returns a python list of strings for the texts of this node.

def addText(self, text):
Adds a new text entry to the texts of this node.

def setTexts(self, texts):
Set a python list of strings for the texts of this node.

4.4.2 Node localisation methods

def hasNode(self, tag):
Checks if this node has any child nodes of this tag.

def getNode(self, tag):
If this node has any child nodes of this tag,
returns a python list of these nodes.

def get(self, tag=None, **keywords):
If this node has any child nodes of this tag, matching this set of keyword attributes, returns a python list of these nodes.
tag: a tag name.
**attrs: is a set of pair attr=value keyword parameters.
Ex:
nodes=top.get(tag='tag1', attr1='attr1', attr2='attr2')

4.4.3 Node localisation/PxQuery methods

def td(self, pxq, checkIsNode=True, checkIsAttr=False, checkIsUnique=False, separator=','):

pxq: syntax: TAG[/TAG][@ATTR]
e.g.:
tag1/tag2/tag3 or
tag1/tag2/tag3@attr1

If last item is a tag and checkIsNode is True returns a python list of all matching nodes.
If last item is an attribute and checkIsAttr is True returns a separated list value(s) of this attribute for the matching nodes.
checkIsNode: If true last item is treated as a tag name and a python list of matching nodes is returned.
checkIsAttr: If true last item is treated as an attribute name and this attribute's value for the matching nodes is returned.
checkIsUnique: checks if the last found nodes are uniques.
Note:the intermediate nodes must be uniques.
separator: allows a custom separator.

def tdc(self, pxq, checkIsNode=True, checkIsAttr=False, checkIsUnique=False, separator=','):

PxQuery top down complete method.
pxq: syntax: TAG[@ATTR=VALUE[,@ATTR=VALUE]],@ATTR Tag sep:/

e.g.:
To retreive a node:
servers@type=backend,@site=london/server
or to retreive an attribute:
servers@type=backend,@site=london/server@host=axane,@ip

If last item is a tag and checkIsNode is True returns a python list of all matching nodes.
If last item is an attribute and checkIsAttr is True returns a separated list value(s) of this attribute for the matching nodes.
checkIsNode: If true last item is treated as a tag name and a python list of matching nodes is returned.
checkIsAttr: If true last item is treated as an attribute name and this attribute's value for the matching nodes is returned.
checkIsUnique: checks if the last found nodes are uniques.
Note:the intermediate nodes must be uniques.
separator: allows a custom separator.

4.4.4 Print methods

def printText(self, doChild=True, indent='', step=4, doLBBeforTag=True, noDft=True, noDftRaise=False, _sb=None):

Print a text represention of this node and its children if doChild is True (default).

indent (integer) : if indent is given text is printed respecting this margin.
doChild (bool) : if doChild is True inner subtree nodes are also printed, otherwise only this node is printed.
step (integer) : width between each tree node when doChild is True.
doLBBeforTag (bool) : if True do a line break befor writing tag.
_sb : internal use only.

def printXml(self, doChild=True, indent='', step=4, doSpaceWrapEq=False, doLBBeforTag=True, noDft=True, noDftRaise=False, _nbstep=0, _sb=None, fct_omit=None):

Print an xml represention of this node and its children if doChild is True (default).

indent (integer) : if indent is given text is printed respecting this margin.
doChild (bool) : if doChild is True inner subtree nodes are also printed, otherwise only this node is printed.
doSpaceWrapEq (bool) : if doSpaceWrapEq is True a space will wrap the sign equal in the output, likewise: ATTR_NAME = 'ATTR_VALUE'.
if doSpaceWrapEq is False no space will wrap the sign equal in the output, likewise: ATTR_NAME='ATTR_VALUE'.
step (integer) : width between each tree node when doChild is True.
doLBBeforTag (bool) : if True do a line break befor writing tag.
_nbstep : internal use only.
_sb : internal use only.

5 TRADEMARKS:

AIX is a registred trademarks of International Business Machines Corporation.

Windows is a registred trademark of Microsoft Corporation

Other names may be trademarks of their respective owners.