4XSLT

Davy Friedrich
Joe Achten
Wim Deprez

Wat is 4XSLT?

4XSLT is een open source applicatie voor het processen van de *XSLT 1.0* implementatie (eXtensible Stylesheet Language Transformations) gedefinieerd door het *World Wide Web Consortium* (het *W3C*: http://www.w3.org). *4XSLT* is een onderdeel van *4Suite* (http://www.4suite.org), een verzameling open source tools voor XML processing en object database management, geschreven in de "Very High Level Language" *Python* (http://www.python.org) en wordt onderhouden door *Fourthought Inc.* (http://www.fourthought.com). Het is een pakket dat, naast *4XSLT*, ook nog componenten voor het gebruik van DOM (in-memory en persistent), XPath, RDF (Resource Description Framework), OGMG object databases, XPointer en XLink (eXtensible Linking) bevat.

De meest gebruikte toepassing van *4XSLT* is het omzetten van XML documenten naar aangepaste en gestilleerde HTML voor de huidige web browsers. *4XSLT* heeft ook nog een krachtige API voor applicaties voor het gebruik van low-level aangepaste omzettingen van XML documenten.

Om voorbeeldapplicaties te testen, is er gebruik gemaakt van ***Python 2.1*** *- #15, Apr 16 2001, 18:25:49* (ftp://ftp.python.org/pub/python/ of ftp://ftp.sourceforge.net/pub/sourceforge/python/Python-2.1.exe), ***PyXML 0.6.6*** (http://sourceforge.net/project/showfiles.php?group_id=6473 of http://prdownloads.sourceforge.net/pyxml/PyXML-0.6.6.win32-py2.0.exe) aangezien versie 0.7.0 niet ondersteund wordt in de laatste stabiele versie van ***4Suite, versie 0.11.1*** (http://4suite.org/download.html#4Suite) voor Windows 98 (2nd edition).

Gebruik van 4XSLT

Er zijn twee soorten gebruik van *4XSLT*, via de command line of de geïmplementeerde Application Program Interface (API) `Processor` die het mogelijk maakt om vanuit een Python programma gebruik te maken van *4XSLT*.

1. Gebruik van 4XSLT via de command line

De command line is de meest eenvoudige manier om *4XSLT* te runnen, het eenvoudigste voorbeeld is een XML bestand aan te maken (`source.xml`) en een XSLT file (`transform.xsl`). Om hierop *4XSLT* toe te passen, is het slechts nodig om het volgende commando te geven:

```
4xslt source.xml transform.xsl
```

Om de volledige command line syntax te weten, is het slechts nodig om *4XSLT* op te starten:

```
Usage:
      4xslt   [--help] [--version] [--validate] [--ignore] [--define=<definition>]
              [--includepath=<baseUri>] [--outfile=<fileName>]
              [--stacktrace-on-error] [--noxinclude] [--trace]
              [--trace-file=<trace-file>] [--debug] source-uri [stylesheet-uri]...
Options:
      -h
      --help                        show detailed help message
      -V
      --version                     Display current version.
      -v
      --validate                    Validate the input file as it is being parsed.
      -i
      --ignore                      Ignore stylesheet processing instructions in the
                                    input file.
      -D<definition>
      --define=<definition>         Bind a top-level parameter, overriding any
                                    binding in the stylesheet. A definition is in
                                    the form of name=value
      -I<baseUri>
      --includepath=<baseUri>       Set a base URI where stylesheet import and
                                    include will search
      -o<fileName>
      --outfile=<fileName>          Specify a filename for the output. This file
                                    will be overwritten if present.
      -e
      --stacktrace-on-error         Display a stack trace trace when an error occurs.
      --noxinclude                  Do no expansions of XInclusions in source
                                    documents or stylesheets.
      --trace                       Output information tracing the progress the
                                    processing as it occurs. Output will be
                                    sent to standard output by default.
      --trace-file=<trace-file>     Specify output file fof execution trace output..
      -d
      --debug                       Execute 4xslt in the command line debugger.

Arguments:
      source-uri     The URI of the source XML document. Note: if you use "-" as
                     the name of the source document, the source will instead be
                     read from standard input.
      stylesheet-uri The URI(s) of the stylesheets to apply to the source XML.

4XSLT Command line application.
```

Het is mogelijk om via de command line verschillende stylesheets op te geven. In deze stylesheets wordt dan gezocht naar een template die matcht met de gegeven source. In de XSLT specficatie staan ook standaard mechanismen om stylesheets te importeren en te includen, natuurlijk zijn deze technieken meer aangeraden dan ze te includen via de command line.

Een interessante optie om verder op in te gaan is de `-D` optie, dewelke gebruikt wordt om een top-level parameter in de stylesheet met een andere waarde te overriden. Hiermee kan men dus een nieuwe waarde aan een globale parameter in het XSLT-bestand geven.

Laat `transform.xsl` bijvoorbeeld de volgende stylesheet bevatten:

```
<?xml version="1.0"?>
  <xsl:transform
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

     <xsl:param name="PARAMETER" select="'Oude
waarde'"/>

     <xsl:template match="/">

               <xsl:value-of select='$PARAMETER'/>

     </xsl:template>
  </xsl:transform>
```

zoals u ziet is dit een zinloze stylesheet, maar hij leent zich goed om de `-D` optie te illustreren en het nut ervan zal dra wel duidelijk worden.

Het bronbestand `source.xml` mag ieder willekeurig bestand zijn volgens de XML-specificatie:

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

     ...

  </bla>
```

Moest men op deze XML de voorgaande stylesheet toepassen, dan krijgt men de volgende output:

```
<?xml version='1.0' encoding='UTF-8'?>
Oude waarde
```

hetgeen vrij voorspelbaar is, maar als men nu de `-D` optie gaat gebruiken, zoals in volgend voorbeeld (case sensitive!):

```
4xslt -D "PARAMETER=Wat zeg je daarvan?" source.xml
transform.xsl
```

ziet men dat de output aangepast is tot:

```
<?xml version='1.0' encoding='UTF-8'?>
Wat zeg je daarvan?
```

Dit is natuurlijk een overgesimplificeerd voorbeeld, maar hiermee is de toepassing van de `-D` optie wel duidelijk geworden.

2. De 4XSLT API `Processor` voor Python

De Application Program Interface (API) van *4XSLT* bestaat uit het object `Processor` dat het hele XSLT processing omvat. Via de method `appendStylesheetUri` wordt er een URI***** (een Universal Resource Identifier) meegegeven. De URI kan naar een lokale file verwijzen. Er zijn nog andere mogelijkheden, zoals `appendStylesheetString`, maar al deze methods nemen dus (een verwijzing naar) een stylesheet aan, die dan kan toegepast worden op een XML document.

***** Python objecten moeten eerst class objecten op het Web worden, de manier waarop dat gebeurd is door ze een URI te geven.
(een **URL** is het meest bekende voorbeeld van een URI).

Laat ons wederom de twee vorige voorbeeldbestanden nemen (`source.xml` en `transform.xsl`) en via Python de volgende code ingeven:

```
from xml.xslt.Processor import Processor
p = Processor()
p.appendStylesheetUri("transform.xsl")
result = p.runUri("source.xml")
print result
```

De `runUri` method neemt het XML document en geeft het resultaat van de stylesheet daarop terug. Dan is de output (zoals reeds bekend):

```
<?xml version='1.0' encoding='UTF-8'?>
Oude waarde
```

De code die hierboven beschreven is, kan men ook opslaan in een .py bestand (`voorbeeld.py`) en dan vanaf de command line runnen met:

```
python voorbeeld.py
```

Voor de volledigheid nog een voorbeeld van het gebruik van `appendStylesheetstring`, hierbij wordt er gebruik gemaakt van `runString` om `source.xml` te processen:

```
transform = """<?xml version="1.0"?>
  <xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

          <xsl:param name="PARAMETER" select="'Oude waarde'"/>

          <xsl:template match="/">

                        <xsl:value-of select='$PARAMETER'/>

          </xsl:template>
  </xsl:transform>"""

source = """<?xml version="1.0"?>
<bla></bla>"""

from xml.xslt import Processor
processor = Processor.Processor()
processor.appendStylesheetString(transform)
result = processor.runString(source)
print result
```

OPGELET! `<?xml version="1.0"?>` moet meteen volgen op `"""` of het betreft hier geen geldig XML document / string!!!

Ook bij het uitvoeren van deze code krijgt men het zelfde resultaat. Maar wat met de opties die bij de command line syntax gebruikt konden worden?
Deze opties kan men op gelijkaardige manier doorgeven aan `Processor`, voor de `-D` optie in command line syntax is dat bijvoorbeeld:

```
from xml.xslt.Processor import Processor
p = Processor()
p.appendStylesheetUri("transform.xsl")
result = p.runUri("source.xml", topLevelParams={
      ("", "PARAMETER"): "Wat zeg je daarvan?"     })
print result
```

Met dit commando wordt dus de waarde van de top-level parameter `PARAMETER` overschreven, zodat men het zelfde resultaat bekomt als met de command line syntax. Merk op dat de naam van de parameter in de *dictionary* (`topLevelParams` is van het type *dictionary* en deelt tupels in in de vorm van (namespace-uri, local-name) die top-level parameters voorstellen) is voorgesteld als een tupel. Dit is nodig omdat de variabelen en de parameters een local name en een namespace URI hebben die bij een prefix horen. De parameter `PARAMETER` in het voorbeeld heeft geen enkele namespace, dus moet er een lege string als eerste argument van het element worden doorgegeven.

Andere opties die (naast `topLevelParams`) nog kunnen doorgegeven worden zijn onder andere:

- `ignorePis` *(integer)*

  Als deze integer 0 is, dan wordt iedere stylesheet processing instruction (PI: een W3C nota die de standaard manier beschrijft om een stylesheet voor het gebruik van een XML-document te bepalen. *4XSLT* leest dergelijke instructies automatisch in en gebruikt de gegeven stylesheets.) die vastgelegd is in de input file toegepast, anders worden die PI's genegeerd. Dit is vergelijkbaar met de `-i` of de `--ignore` optie van de command line syntax.

- `writer` *(Class volgens het protocol van `xml.xslt.TextWriter.TextWriter`)*

  De output handler die gebruikt moet worden (`xml.xslt.TextWriter.TextWriter` als default).

- `outputStream` *(Python FileObject interface)*

  De stream waar de output naar toe moet worden gestuurd (`sys.stdout` als default).

voor de method `runUri`. Voor `runString` komt daar nog bij:

- `baseUri` *(string)*

  De basis URI van de gegeven XML code

Dankzij de API bij *4XSLT* kan men dus in Python code gerust XSLT processing toepassen.

Extra mogelijkheden van 4XSLT

Hetgeen *4XSLT* nog extra aanbiedt naast de standaard ondersteuning van de *XSLT 1.0* specificatie van het *W3C*, is onder andere de `-D` optie om een top-level parameter in de stylesheet te overriden. Dit werd in het voorgaande punt uitvoerig uitgelegd.

Daarnaast onderteunt *4XSLT* ook nog de *EXSLT:node-set*. *EXSLT* (http://www.exslt.org/) is een gemeenschappelijk innitiatief dat (open) extensies voor de *XSLT* specificatie bepaalt. Die extensies zijn opgedeeld in een aantal modules: *Common* (de basis extensie elementen), *Math* (voor wiskundige bewerkingen), *Sets* (voor bewerkingen op verzamelingen), *Functions* (deze extensie elementen en functies laten de gebruiker toe om zelf functies voor eigen gebruik te definieren in XSLT expressies), *Date and Times* (voor alles in verband met de tijd of de datum), *Strings* (voor bewerkingen op stings), *Regular Expressions* en *Dynamic* (voor de evaluatie van *XPath* expressies).

De *EXSLT* namespace is als een extensie namespace gedeclareerd met het `extension-element-prefixes` attribuut in het `xsl:transform` element. Hier volgt de stylesheet `transform2.xsl`:

```
<?xml version="1.0"?>
  <xsl:transform
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:exslt="http://exslt.org/common"
   extension-element-prefixes="exslt"
   version="1.0"
  >

    <xsl:template match="file">
      <exslt:document href="{generate-id()}.txt">
        <xsl:text>Dit is file #</xsl:text>
        <xsl:value-of select="generate-id()"/>
<xsl:text>
</xsl:text>
        <xsl:value-of select="inhoud"/>
      </exslt:document>
    </xsl:template>

  </xsl:transform>
```

Hierin wordt er gebruik gemaakt van de `common` namespace. De `exslt:document` instructie zorgt er voor dat de output van deze stylesheet naar een bestand wordt geschreven met de naam die vermeld wordt in het `href`-attribuut*. Als deze stylesheet nu wordt toegepast (via de command line syntax of de API) op volgend XML document:

```
<?xml version="1.0"?>
<bla>
  <file>
    <inhoud>Dit is de inhoud van de eerste file
</inhoud>
  </file>
  <file>
    <inhoud>Dit is de inhoud van de tweede file
</inhoud>
  </file>
</bla>
```

dan is de output twee txt-files (`id14460860.txt` en `id14464796.txt`) die elk twee regels tekst bevatten, voorgegaan door `<?xml version='1.0' encoding='UTF-8'?>`.

Dit is niet de enige extension set die *4XSLT* bevat, *Fourthought Inc.* heeft zelf ook een set gedefinieerd (http://xmlns.4suite.org/ext). Mocht dit nog niet genoeg zijn, dan biedt *4XSLT* ook de mogelijkheid aan om zelf uitbreidingselementen te schrijven. Het doel van volgend voorbeeld is om een extensie te schrijven om de juiste tijd en datum als output te krijgen. Daarvoor moet men eerst in Python de volgende code schrijven:

---

* Merk op dat de accolades in het `href`-attribuut erop wijzen dat hier gebruik wordt gemaakt van een zo genaamd *attribute value template* (AVT) en de inhoud hiervan wordt behandeld als een *XPath* expressie.

```
#!/usr/bin/env python

import time
import xml.dom.ext
from xml.xslt import XsltElement, XSL_NAMESPACE, AttributeValueTemplate

class CurrentDateElement(XsltElement):

    def __init__(self, doc, uri='http://foo.org/namespaces/date-ext',
                 localName='make-date', prefix='xsl', baseUri=''):
        XsltElement.__init__(self, doc, uri, localName, prefix, baseUri)

    def setup(self):
        self.__dict__['_nss'] = xml.dom.ext.GetAllNs(self)
        self.__dict__['_format'] = AttributeValueTemplate.AttributeValueTemplate(
                                               self.getAttributeNS('', 'format'))

    def instantiate(self, context, processor):
        origState = context.copy()
        context.setNamespaces(self._nss)
        format = self._format.evaluate(context)
        processor.writers[-1].text(time.strftime(format, time.localtime()))
        context.set(origState)
        return (context,)

ExtElements = {
    ('http://foo.org/namespaces/date-ext', 'make-date'): CurrentDateElement
}
```

Noem dit `DateExtension.py` en plaats het in de directory onder Python waar Python zoekt naar modules (in de `Lib`-map voor Windows gebruikers, of in de directory `site-packages` voor Linux gebruikers). Vanaf nu kan men dus met deze Python versie "`make-date`" gebruiken in een stylesheet, om de huidige datum en/of de tijd te laten uitschrijven als men het formaat waarin dit moet gebeuren meegeeft als attribuut. Gebruik van `DateExtension.py` via de API van *4XSLT*:

```
sheet = """<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="http://foo.org/namespaces/date-ext"
  extension-element-prefixes="date"
  version="1.0">

<xsl:template match="example">
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="current-date">
<date:make-date format="{@format}"/>
</xsl:template>

</xsl:stylesheet>
"""
```

```
source = """<?xml version="1.0"?>
<example>
Dit is gewoon wat tekst.
Generated on <current-date format="%a, %d %b %Y %H:%M:%S"/>
</example>
"""

from xml.xslt import Processor
processor = Processor.Processor()
processor.registerExtensionModules(['DateExtension'])
processor.appendStylesheetString(sheet)
result = processor.runString(source)
print result
```

met dank aan Geert-Jan Van den Bogaerde

De `registerExtensionModules(['DateExtension'])` dient om de uitbreiding `DateExtension` in te laden, zodat deze kan toegepast worden. De format string `"%a, %d %b %Y %H:%M:%S"` volgt het formaat gedefinieerd in http://www.python.org/doc/current/lib/module-time.html, zie de documentatie voor de `strftime` functie.

Na het uitvoeren van deze Python code, zou een soortgelijk resultaat bekomen moeten worden:

```
<?xml version='1.0' encoding='UTF-8'?>

Dit is gewoon wat tekst.
Generated on Mon, 18 Mar 2002 14:27:45
```

zoniet, dan is de `DateExtension.py` waarschijnlijk foutief geplaatst.

Een laatste extra mogelijkheid van *4XSLT* die hier besproken wordt, is het gebruik van de `BETA_DOMLETTE` environment variabele, deze kan `true` gezet worden (`1`) of helemaal niet.

4Suite is namelijk vanaf versie 0.12.0a1 aan het overschakelen naar een *DOM* implementatie die op C is gebaseerd. Deze *DOM* implementatie wordt gebruik voor de interne processen, waaronder *XSLT*. Momenteel wordt standaard nog steeds de Python implementatie gebruikt, die veel trager is. Want dat is het grootste nadeel van *4XSLT* en *4Suite* in het algemeen: door zijn implementatie in Python is deze tool trager dan zijn soortgenoten die meestal in C, C++ of zelfs in Java zijn geschreven.

Op dit moment bestaan beide implementaties (respectievelijk `cDomlette` en `pDomlette` genoemd) naast mekaar en om te veranderen van *DOM* implementatie moet men de `BETA_DOMLETTE` environment variabele gebruiken. Op intern niveau gebruikt men namelijk een wrapper class `Domlette` die naar de `BETA_DOMLETTE` environment variabele kijkt die in de shell geset kan worden voordat Python of *4XSLT* wordt opgestart:

```
set BETA_DOMLETTE=1
```

Staat de variabele op `true` (`1`), dan gebruikt *4Suite* `cDomlette`. `pDomlette` is stabiel maar traag, `cDomlette` is snel maar onstabiel.

Code uit de class `Domlette.py`:

```
#Because of the state of CDomlette (not done) we are forcing pDomlette unless
specified.

CDOMLETTE = os.environ.has_key("BETA_DOMLETTE")

if CDOMLETTE:
    sys.stderr.write( "::: Using cDomlette\n")
    from cDomlette import DEFAULT_VALIDATING_READER,
DEFAULT_NONVALIDATING_READER
    from cDomlette import XmlStrStrip, XmlStrRStrip, XmlStrLStrip
    from cDomlette import implementation, Node

    #NOTE, there is no print ro GetAllNs in C yet so use the python versions
    from Lib.Nss import GetAllNs

    #Note, there is no Print and PrettyPrint in c yet so use the python
versions
    from Lib.Print import Print, PrettyPrint

else:
    sys.stderr.write( "::: Using pDomlette\n")
    from pDomlette import DEFAULT_VALIDATING_READER,
DEFAULT_NONVALIDATING_READER
    from pDomlette import XmlStrStrip, XmlStrRStrip, XmlStrLStrip
    from pDomlette import implementation, Node

    from Lib.Nss import GetAllNs
    from Lib.Print import Print, PrettyPrint


##PDOMLETTE = os.environ.has_key("FT_FORCE_PDOMLETTE")
##CDOMLETTE = os.environ.has_key("FT_FORCE_CDOMLETTE")
```

De software architectuur van 4XSLT

4XSLT, als deel van *4Suite* is een verzameling van gekoppelde componenten, dus de beste manier om de code te verstaan is om het stukje bij beetje uit te pluizen. *4Suite* is een dynamisch project, dus naarmate het groeit is het vrij moeilijk om te vinden waar te beginnen. `Processor.py` is de spil waarrond *4XSLT* draait. `Stylesheet.py` bepaalt de structuur van een stylesheet tijdens het processen en `StylesheetReader.py` maakt die structuur aan. Van de `Element.py`'s is `TemplateElement.py` de meest belangrijke, al de andere zijn specifieke *XSLT* instructies. Daarnaast is er nog `NullWriter.py` die de output genereert, de ander `Writer.py`'s zijn daar afgeleiden van. Voor de extension elements bij *4XSLT* zijn er `Exslt.py`, `BuiltInExtElements.py` en `XsltFunctions.py`.

1. Processor.py

Hierin wordt de class `Processor` aangemaakt, die de API van *4XSLT* controleert. `Processor` is het hart van *4XSLT*. De functies die het eigenlijk XML document gaan parsen zijn:

- *RunStream*
- *RunUri*
- *RunString*
- *RunNode*

Het verschil zit hem in de manier waarop het document doorgegeven wordt. Dit gebeurt respectievelijk via een “file-like object” *****, het meegeven van een URI, het expliciet intypen van de XML-file (of via een string variabele) en het verwerken van een inwendige DOM-tree. Al deze functies laten het eigenlijke parsen echter over aan de functie `Execute`. Deze ziet er als volgt uit :

```
def execute(self, node, ignorePis=0, topLevelParams=None, writer=None,
                baseUri='', outputStream=None):
        '''
        Run the stylesheet processor against the given XML DOM node with the
        stylesheets that have been registered.  Does not mutate the DOM
        If writer is None, use the XmlWriter, otherwise, use the
        supplied writer
        '''
        topLevelParams = topLevelParams or {}
```

***** File-like objecten zijn objecten die zich gedragen als gewone files. Dit wil zeggen dat ze functies implementeren die een file normalerwijs heeft, zoals read(), readline(), readlines(), write(), flush(), ...

```
        self.attributeSets = {}
        self.keys = {}

        if not self.stylesheet:
            raise XsltException(Error.NO_STYLESHEET)

        self._outputParams = self.stylesheet.outputParams

        if writer:
            self.writers = [writer]
        else:
            self.writers = []
            self.addHandler(self._outputParams, outputStream, 0)

        # Setup the named templates
        self._namedTemplates = self.stylesheet.getNamedTemplates()

        # Initialize any stylesheet parameters
        tlp = topLevelParams.copy()
        self._normalizeParams(tlp)
        self.stylesheet.prime(node, self, tlp, baseUri)

        #Run the document through the style sheets
        self.writers[-1].startDocument()
        context = XsltContext.XsltContext(node, 1, 1, None, processor=self)
        context.documents[baseUri] = node
        context.varBindings = self.stylesheet.getGlobalVariables()

        self.applyTemplates(context, None)

        self.stylesheet.idle(node, self, baseUri)

        self.writers[-1].endDocument()

        Util.FreeDocumentIndex(node)

        result = self.writers[-1].getResult()
        self._lastOutputParams = self.writers[-1]._outputParams

        context.release()

        return result
```

Je kunt zien dat deze de functie `applyTemplates` oproept. Dit is een functie die voor alle kinderen van de huidige context-node zichzelf gaat oproepen (recursief dus). Deze ziet eruit als volgt :

```
    def applyTemplates(self, context, params=None):
        params = params or {}
        if not self.stylesheet.applyTemplates(context, self, params):
            # No matching templates found, use builtin templates
            if params and not self._builtInWarningGiven:
                self.warning('Built-in template invoked with params that '
                             'will be ignored. This message will only '
                             'appear once per transform.')
                self._builtInWarningGiven = 1
            if context.node.nodeType == Node.TEXT_NODE:
                self.writers[-1].text(context.node.data)
            elif context.node.nodeType in [Node.ELEMENT_NODE,
                                           Node.DOCUMENT_NODE]:
                origState = context.copyNodePosSize()
                node_set = context.node.childNodes
                size = len(node_set)
                pos = 1
                for node in node_set:
                    context.setNodePosSize((node,pos,size))
                    self.applyTemplates(context)
                    pos += 1
                context.setNodePosSize(origState)
            elif context.node.nodeType == Node.ATTRIBUTE_NODE:
                self.writers[-1].text(context.node.value)
        return
```

De functie execute krijgt een parameter `writer` mee. Deze geeft aan welke outputwriter gebruikt moet worden. Als er geen writer wordt meegegeven, wordt er standaard de `XmlWriter` gebruikt. De beschikbare writers zijn :

- *DomWriter* : een writer om van een *XSLT* output Dom DocumentFragments te construeren
- *HtmlWriter* : een HTML output writer voor XSLT processor output
- *RtfWriter* : een simpele writer voor het opvangen van Result-Tree Fragments (RTF)
- *SAXWriter* : de core writer voor XSLT processor output

Deze writers zijn allemaal afgeleid van `NullWriter`, een standaardwriter die alleen maar lege functie-bodies bevat.

2. Stylesheet.py

In de class `StylesheetElement(XsltElement)` wordt de stylesheet (gestructureerd volgens `StylesheetReader.py`) in het geheugen bijgehouden. *XSLT* nodes worden in een dictionary gezet, alle elementen wiens naam niet voorkomen in de dictionary worden weggegooid.

Toepassingen van 4XSLT

De grote toepassing van *XSLT* is natuurlijk het omzetten van een XML document naar een file. Bijna alles is tegenwoordig mogelijk. *XSLT* wordt gebruikt om informatie uit verschillende databases tussen bedrijven uit te wisselen (als de XML documenten gebaseerd op die databanken niet op elkaar afgetuned zijn), voor het creeeren van HTML-bestanden (bijvoorbeeld http://www.opentechnology.org/ of http://www.space-combat.net) om onder andere aan iedere geregistreerde gebruiker een aangepaste visuele opmaak te kunnen aanbieden, om informatie over software producten vanuit een XML document om te zetten naar alle andere mogelijke formaten, gaande van PDF over RTF tot LATEX en ga zo maar door.

4XSLT wordt met *4Suite* voornamelijk gebruikt in de *4Suite Server*, ook een product van *Fourthought Inc.* dat gebruik maakt van standaard XML technologiën voor het transformeren, viewen, query'en, linken en doorzoeken van XML content. Het programma gedraagt zich als een "zwarte doos" en reageert op applicatie requests (locaal of over een netwerk) voor o.a.: *RDF* (Resource Description Framework), ***XSLT*** (eXtensible Stylesheet Language for Transforms),en *XLink* (eXtensible Linking). Het heeft een *CORBA* interface, dus het kan opgeroepen worden vanuit de meeste talen en platforms, waaronder Java, Oracle, Back Office, UNIX en Windows.

Een andere python applicatie die *4XSLT* ondersteund, is *Maki* (http://maki.sf.net), een XML webserving framework.

4XSLT in vergelijking met de W3C specificatie (XSLT 1.0)

Voor het testen van de conformance van *4XSLT* werd er gebruik gemaakt van ***Python 2.1*** *- #15, Apr 16 2001, 18:25:49,* ***PyXML 0.6.6*** en de laatste stabiele versie van ***4Suite, versie 0.11.1*** voor Windows 98 (2nd edition). Het was natuurlijk beter geweest, moest het mogelijk zijn om de allerlaatste versie van dit moment: versie 0.12.0a1 te kunnen testen, maar spijtig genoeg geraakte die niet juist geinstalleerd in onze testomgeving. Door dit toeval is het dan ook onmogelijk om de verschillende `Domlette`'s met `BETA_DOMLETTE` te testen, het geen zeer spijtig is. De `cDomlette` staat nog niet op punt en geeft soms onverwachte resultaten die in het ergste geval zelfs in een crash van het proces zouden kunnen resulteren (afhankelijk van het platform dat gebruikt wordt). Het plan is om uiteindelijk `pDomlette` te elimineren (door zijn traagheid), maar dat zal natuurlijk niet gebeuren voordat `cDomlette` stabiel is.

Als testkit werd de *XSL Testing* (http://xw2k.sdct.itl.nist.gov/xml/page5.html) van *NIST* (National Institute of Standards and Technology - http://www.nist.gov/) toegepast, waarop *4XSLT* toch geen slecht resultaat behaalde: bij "slechts" 17 van de 182 tests was de output niet helemaal correct, waarvan er weer een paar gerelativeerd kunnen worden. Hier volgt een lijst met opmerkingen (zie bijlage voor de code):

1) resultaten van de CoreFunctionTests

- `round()` wordt verkeerd uitgevoerd als het argument "`NaN`" is, in plaats van weer `NaN` (Not a Number) terug te geven, geeft *4XSLT* `0` als resultaat terug (CoreFunctionTest080).
- `round()` met een deling door 0 als argument wordt ook niet volledig correct behandeld. 4XSLT geeft telkens `-1.#IND` als resultaat (of het deeltal nu positief is of niet), maar de specificatie schrijft voor dat er een onderscheid gemaakt moet worden tussen positief en negatief oneindig (CoreFunctionTest084 en CoreFunctionTest085).

2) resultaten van de DataManipulationTests

Geen fouten.

3) resultaten van de ExpressionTests

- Bij een unie van `NodeSet`'s die gebruik maken van `ancestor`, `ancestor-or-self`, `child` of `descendant-or-self` als locatie pad, gaat het mis. Hier werd geen resultaat bekomen (ExpressionTest003, ExpressionTest004, ExpressionTest007 en ExpressionTest010).
- Bij een unie van `NodeSet`'s die gebruik maken van `attribute` als locatie pad, gaat het ook mis. De resultaten werden in omgekeerde volgorde uitgeschreven (ExpressionTest006).
- In plaats van de expressie `2.1 > NaN` als vals aan te nemen, wordt deze als `true` geëvalueerd (ExpressionTest033).

4) resultaten van de OutputTests

Door het gebruik van `UTF-16 encoding` gaat het hier mis onder Windows. De returns en andere speciale tekens worden namelijk anders behandeld, maar de inhoud van de output was wel juist.

5) resultaten van de ResultTreeTests

- Bij het uitschrijven van een aantal attributen door middel van `xsl:copy` of `xsl:element` met `xsl:attribute-set`, worden de attributen in een verkeerde volgorde weggeschreven (ResultTreeTest001, ResultTreeTest003, ResultTreeTest005 en ResultTreeTest008).

6) resultaten van de TemplateTests

- Bij het uitvoeren van TemplateTest003 komt de stylesheet in de laatste template match terecht, hetgeen foutief is (zie code).

Doordat er met versie 0.11.1 gewerkt werd, is het niet onwaarschijnlijk dat een groot deel van deze fouten al zijn weggewerkt.

==========

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: coreFunction080 -->
  <!-- Document: http://www.w3.org/TR/xpath -->
  <!-- DocVersion: 19990922 -->
  <!-- Section: 4.4 Number Functions -->
  <!-- Purpose: Test of 'round' function with NAN as its
       its argument. -->
```

```
<xsl:template match="doc">
 <out>
   <xsl:value-of select="round(NaN)"/>
 </out>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: coreFunction084 -->
  <!-- Document: http://www.w3.org/TR/xpath -->
  <!-- DocVersion: 19990922 -->
  <!-- Section: 4.4 Number Functions -->
  <!-- Purpose: Test of 'round' function with positive infinity
       as the argument (use expression that yields infinity). -->

<xsl:template match="doc">
 <out>
   <xsl:value-of select="round(2.3 div 0)"/>
 </out>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: coreFunction085 -->
  <!-- Document: http://www.w3.org/TR/xpath -->
  <!-- DocVersion: 19990922 -->
  <!-- Section: 4.4 Number Functions -->
  <!-- Purpose: Test of 'round' function with negative infinity
       as the argument (use expression that yields -infinity). -->

<xsl:template match="doc">
 <out>
   <xsl:value-of select="round(-2.3 div 0)"/>
 </out>
</xsl:template>
</xsl:stylesheet>
```

==========

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: expression003.xsl -->
  <!-- Document: http://www.w3.org/TR/Xpath -->
  <!-- Section: 3.3 Node Sets -->
  <!-- Purpose: NodeSet union using the ancestor location path-->
  <!-- Author: Carmelo Montanez -->

 <xsl:template match="/">
   <out>
   <xsl:for-each select="//child1|//child2">
     <xsl:apply-templates select="ancestor::sub1|ancestor::sub2"/>
   </xsl:for-each>
  </out>
 </xsl:template>

 <xsl:template match="*">
   <xsl:value-of select="."/>
 </xsl:template>
</xsl:stylesheet>
```

toegepast op:

```
<?xml version="1.0"?>
 <doc>
   <sub1>
     <child1>descendant number 1</child1>
   </sub1>
   <sub2>
     <child2>descendant number 2</child2>
   </sub2>
 </doc>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: expression004.xsl -->
  <!-- Document: http://www.w3.org/TR/Xpath -->
  <!-- Section: 3.3 Node Sets -->
  <!-- Purpose: NodeSet union using the ancestor-or-self location path-->
  <!-- Author: Carmelo Montanez -->

 <xsl:template match="/">
   <out>
     <xsl:for-each select="//child1|//child2">
```

```
       <xsl:apply-templates select="ancestor-or-self::sub1|ancestor-or-self::sub2"/>
     </xsl:for-each>
   </out>
 </xsl:template>
 <xsl:template match="*">
   <xsl:value-of select="."/>
 </xsl:template>
</xsl:stylesheet>
```

toegepast op:

```
<?xml version="1.0"?>
 <doc>
   <sub1>
     <child1>descendant number 1</child1>
   </sub1>
   <sub2>
     <child2>descendant number 2</child2>
   </sub2>
 </doc>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: expression006.xsl -->
  <!-- Document: http://www.w3.org/TR/Xpath -->
  <!-- Section: 3.3 Node Sets -->
  <!-- Purpose: NodeSet union using the attribute location path-->
  <!-- Author: Carmelo Montanez -->

 <xsl:template match="/">
   <out><xsl:text>
   </xsl:text>
    <xsl:for-each select="doc">
      <xsl:apply-templates select="attribute::attr1|attribute::attr2"/>
    </xsl:for-each>
   <xsl:text>
</xsl:text></out>
 </xsl:template>

 <xsl:template match="*">
   <xsl:value-of select="."/>
 </xsl:template>
</xsl:stylesheet>
```

toegepast op:

```
<?xml version="1.0"?>
 <doc attr1="attribute 1 " attr2="attribute 2">
   <sub1>
     <child1>child number 1</child1>
   </sub1>
   <sub2>
     <child2>child number 2</child2>
   </sub2>
 </doc>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: expression007.xsl -->
  <!-- Document: http://www.w3.org/TR/Xpath -->
  <!-- Section: 3.3 Node Sets -->
  <!-- Purpose: NodeSet union using the child location path-->
  <!-- Author: Carmelo Montanez -->

  <xsl:template match="/">
   <out>
    <xsl:for-each select="doc">
     <xsl:apply-templates select = "child::sub1|child::sub2"/>
    </xsl:for-each>
   </out>
 </xsl:template>

 <xsl:template match="*">
   <xsl:value-of select = "."/>
 </xsl:template>
</xsl:stylesheet>
```

toegepast op:

```
<?xml version="1.0"?>
 <doc>
   <sub1>
     <child1>child number 1</child1>
   </sub1>
   <sub2>
     <child2>child2 number 2</child2>
   </sub2>
```

```
 </doc>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: expression010.xsl -->
  <!-- Document: http://www.w3.org/TR/Xpath -->
  <!-- Section: 3.3 Node Sets -->
  <!-- Purpose: NodeSet union using the descendant-or-self location path-->
  <!-- Author: Carmelo Montanez -->

 <xsl:template match="/">
   <out>
     <xsl:for-each select="doc">
       <xsl:apply-templates select="descendant-or-self::doc|descendant-or-self::doc"/>
     </xsl:for-each>
   </out>
 </xsl:template>

 <xsl:template match="*">
   <xsl:value-of select = "."/>
 </xsl:template>
</xsl:stylesheet>
```

toegepast op:

```
<?xml version="1.0"?>
 <doc attr1="attribute 1 " attr2="attribute 2">
   <sub1>
     <child1>descendant number 1</child1>
   </sub1>
   <sub2>
     <child2>descendant number 2</child2>
   </sub2>
 </doc>
```

==========

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: ResultTree001.xsl -->
  <!-- Document: http://www.w3.org/TR/xslt -->
  <!-- Section: 7.1.4 Named Attribute Sets -->
  <!-- Purpose: Set attributes of an xsl:copy using attribute sets that inherit. -->
  <!-- Author: Carmelo Montanez -->

<xsl:template match="foo">
  <out>
    <xsl:copy use-attribute-sets="set1"/>
  </out>
</xsl:template>

<xsl:attribute-set name="set2" use-attribute-sets="set3">
  <xsl:attribute name="text-decoration">underline</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="set1" use-attribute-sets="set2">
  <xsl:attribute name="color">black</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="set3">
  <xsl:attribute name="font-size">14pt</xsl:attribute>
</xsl:attribute-set>

</xsl:stylesheet>
```

toegepast op:

```
<?xml version="1.0"?>
<test1>
  <foo>a</foo>
</test1>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: ResultTree.xsl -->
  <!-- Document: http://www.w3.org/TR/xslt -->
  <!-- Section: 7.1.4 Named Attribute Sets -->
  <!-- Purpose: Set attributes of element created with xsl:copy with inheritance and
overlappings set name and attribute with xsl:attribute. -->
  <!-- Author: Carmelo Montanez -->

<xsl:template match="foo">
  <out>
    <xsl:copy use-attribute-sets="set1">
      <xsl:attribute name="text-decoration">none</xsl:attribute>
    </xsl:copy>
  </out>
```

```
</xsl:template>

<xsl:attribute-set name="set1">
  <xsl:attribute name="text-decoration">underline</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="set1">
  <xsl:attribute name="color">black</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="set1">
  <xsl:attribute name="font-size">14pt</xsl:attribute>
</xsl:attribute-set>

</xsl:stylesheet>
```

toegepast op:

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

<doc>
  <foo>a</foo>
</doc>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: ResultTree005.xsl -->
  <!-- Document: http://www.w3.org/TR/xslt -->
  <!-- Section: 7.1.4 Named Attribute Sets -->
  <!-- Purpose: Set attributes of an xsl:element using attribute sets that
       inherit. -->
  <!-- Author: Carmelo Montanez -->

<xsl:template match="/">
  <out>
    <xsl:element name="test" use-attribute-sets="set1"/>
  </out>
</xsl:template>

<xsl:attribute-set name="set2" use-attribute-sets="set3">
  <xsl:attribute name="text-decoration">underline</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="set1" use-attribute-sets="set2">
  <xsl:attribute name="color">black</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="set3">
  <xsl:attribute name="font-size">14pt</xsl:attribute>
</xsl:attribute-set>

</xsl:stylesheet>
```

toegepast op:

```
<?xml version="1.0"?>
<doc>
  <foo>a</foo>
</doc>
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: ResultTree008.xsl -->
  <!-- Document: http://www.w3.org/TR/xslt -->
  <!-- Section: 7.1.4 Named Attribute Sets -->
  <!-- Purpose: Set attributes of element created with xsl:element with inheritance and
overlappings set name and attribute with xsl:attribute. -->
  <!-- Author: Carmelo Montanez -->

<xsl:template match="/">
  <out>
    <xsl:element name="element1" use-attribute-sets="set1">
      <xsl:attribute name="text-decoration">none</xsl:attribute>
    </xsl:element>
  </out>
</xsl:template>

<xsl:attribute-set name="set1">
  <xsl:attribute name="text-decoration">underline</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="set1">
  <xsl:attribute name="color">black</xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="set1">
  <xsl:attribute name="font-size">14pt</xsl:attribute>
```

```
</xsl:attribute-set>

</xsl:stylesheet>
```

toegepast op:

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

<doc>
  <foo>a</foo>
</doc>
```

```
                              ================================
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- FileName: template003 -->
  <!-- Document: http://www.w3.org/TR/xslt -->
  <!-- DocVersion: 19991116 -->
  <!-- Section: 5.7 -->
  <!-- Purpose: Test apply-templates for node with a mode
       and moded matching template. -->

<xsl:template match="doc">
  <out>
    <xsl:apply-templates select="node()" mode="mode1"/>
  </out>
</xsl:template>

<xsl:template match="node()" mode="mode1">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="node()">
  This test failed to execute properly.
</xsl:template>
</xsl:stylesheet>
```

toegepast op:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<doc>
  <child1>This is the child number 1.</child1>
</doc>
```

```
                              ================================
```

indien niet anders vermeld, werden deze stylesheets toegepast op:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<doc></doc>
```