DocBook to XHTML

COLLABORATORS			
	TITLE : DocBook to XHTML		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jordi Fita	February 6, 2018	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
29081e152caf	2011-05-31	Added the 'notranslate' class to the code's div output in db2html.	jfita
34b7522b4f97	2011-03-28	atangle is now using a new style for directives which don't collide with XML tags. I had to update all games and programs as well in order to use the new directive syntax.	jfita
6cc909c0b61d	2011-03-07	Added the comments section.	jfita
a43774cb5c70	2011-01-25	db2html now takes into account XML idiosyncrasies.	jfita
3afa2eb8824f	2010-11-12	Fixed missing tokens from lexer in db2html.	jfita
2d89308d5f16	2010-11-10	Fixed a problem with double end of line values in db2html's literate programming filter.	jfita
d1e8f7703f36	2010-11-10	Corrected the literate programming directive's regexp to include the dot character.	jfita
8c7d8f36c874	2010-10-30	Fixed a typo.	jfita
a643bad18ca3	2010-10-28	Fixed a typo in db2html.	jfita
ec13c85db550	2010-10-27	Added a missing source style to db2html.txt	jfita

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
30b4b6244050	2010-10-27	Added the filter for atangle's directive to db2html.	jfita
e3241d8e1dc9	2010-10-25	Added the AsciiDoc's homepage's link to db2html.	jfita
05a1b32f8b4a	2010-10-22	The appendix sections now aren't actual appendix when making a book.	jfita
0ab76df46149	2010-10-20	Added the download links.	jfita
9efbebdaa6ab	2010-10-19	Fixed an unused 'tmp' variable in db2html's print_error function.	jfita
1e5328a5ed6b	2010-10-18	Added the credits appendix to db2html.	jfita
c0dd2ea1b7d8	2010-10-18	Added db2html, a python script to convert from DocBook to XHTML using xsltproc's XSL process and adding pygments' syntax highlighting.	jfita

Contents

1 Introduction **1**

1.1 Syntax highlighting implementation in DocBook stylesheets 1

2 Implementation of a custom `highlight` function **1**

2.1 Highlighting Literate Programming Directives 2

3 Applying an XSLT stylesheet **4**

4 Getting the document and XSLT stylesheet **4**

5 Shebang **5**

6 db2html.py **5**

A Using *db2html.py* **5**

B License **6**

C Credits **6**

D Download **6**

1 Introduction

Since version 1.70, the DocBook XSL stylesheets use the standard xslthl as syntax highlighter for elements that support highlighting: `programlisting`, `screen`, and `synopsis`. Unfortunately, the xslthl is very limited in both supported languages as well as in its highlight capabilities. Moreover, its implementation is in Java an therefore it only works with Java XSL processors like Saxon, which is slower.

One possible alternative to xslthl is Pygments, however its is written in Python and thus not supported by any XSL processor.

DocBook to XHTML or `db2html` is a manual implementation of a XSL processor that uses Pygments for syntax highlighting.

1.1 Syntax highlighting implementation in DocBook stylesheets

In these stylesheets, syntax highlighting is performed by the template `apply-highlighting`, defined in `highlighting/common.xsl` in the DocBook XSL distribution, to all elements which support support highlighting (`programlisting`, `screen`, and `synopsis`.) It determines the language to be used for highlighting from the element's `language` attribute, extracts the content nodes and eventually calls the XPath function `highlight` with these parameters. The function is looked up, in order, in three different namespaces:

1. `s6hl` (`http://net.sf.xslthl/ConnectorSaxon6`)
2. `sbhl` (`http://net.sf.xslthl/ConnectorSaxonB`)
3. `xhl` (`http://net.sf.xslthl/ConnectorXalan`)

If it fails to determine a language or to look up `highlight` in these namespaces, it simply copies the contents.

The `highlight` function returns a list of XML and text nodes. XML nodes with the `xslthl` namespace prefix represent tokens from the highlighted source code. For instance, there are `keyword` and `comment` nodes. Refer to the Processing xslthl results section in the xslthl documentation for more information.

These `xslthl` nodes are then transformed into proper output format by the format-specific DocBook highlighting stylesheets. Note that these must explicitly be included in the customization layer.

These observations lead to the conclusion that if we want to use a custom highlighting routine then we must re-implement this `highlight` function and put into any of the mentioned namespace. In Python, we can use the lxml package that provides the same XSLT processing library as *xsltproc* but that can be extended with custom functions:

```
<<define custom highlight function>>=
xhl = etree.FunctionNamespace('http://net.sf.xslthl/ConnectorXalan')
xhl.prefix = 'xhl'
xhl['highlight'] = html_highlight
```

This adds the namespace `http://net.sf.xslthl/ConnectorXalan` with the prefix `xhl` to the global list of functions namespaces maintained by `lxml`. Then, we add the function `html_highlight` into the namespace as `highlight`. Thus the stylesheet can now call the XPath function `xhl:highlight`.

In order to be able to use `etree`, first we must import the `lxml` package.

```
<<modules>>=
from lxml import etree
```

2 Implementation of a custom `highlight` function

`highlight` returns special XML nodes which are transformed to proper output by the DocBook stylesheets. Unfortunately, the highlighting capabilities of xslthl are somewhat limited compared to those of Pygments. When using DocBook's HTML stylesheet, it is possible to abandon xslthl and use the Pygments formatter.

```
<<pygments html highlighter>>=
def html_highlight(context, language, code, config):
    """
    Highlight the given ``code`` in the given ``language``.  ``context`` is
    the XPath context in which this function was applied.  ``config`` is
    ignored.

    Return a list of HTML nodes containing the highlighted code.
    """
    if not code:
        code = context.context_node.xpath('.//text()')
    lexer = get_lexer_by_name(language[0].lower())
    <<add atangle filter to lexer>>
    html = highlight(code[0], lexer, HtmlFormatter(nowrap=True))
    highlight_div = fragment_fromstring(html, create_parent=True)
    highlight_div.set('class', 'pygments_highlight notranslate')
    return [highlight_div]
```

This code uses the HTML Formatter to render the source code to HTML. This HTML code is then parsed using lxml.html. As the stylesheets already wrap highlighted elements in `pre` tags, `nowrap` is specified to avoid Pygments wrapping them again. Instead, the returned tokens are wrapped in a simple `div` element.

I don't want Google translator to modify the contents of the code, because otherwise they become too mangled to understand. That is why besides the `pygments_highlight` class, I also added the `notranslate` class to the `div` output.

We also need to import the packages from pygments;

```
<<modules>>=
from pygments import lex, highlight
from pygments.formatters import HtmlFormatter
from pygments.lexers import get_lexer_by_name
from pygments.token import Token
```

As well as the fragment_fromstring from the `lxml` package.

```
<<modules>>=
from lxml.html import fragment_fromstring
```

2.1 Highlighting Literate Programming Directives

When using literate programming to create the source code blocks to highlight, besides the regular code in the language specified in the `language` attributes, there are special *directives* used for tangle programs such as atangle to extract these code blocks and write a complete source code module.

Obviously, these directives aren't part of the target language, otherwise the tangle program would confuse them for directives and instead of outputting the source code it would try to satisfy the reference. Nevertheless, `Pyments` is unable to detect that particular use and tries to highlight the directives using the language lexer. Occasionally, that means that the labels gets highlighted as *keywords* — such as when using `for` or `while` inside the directive — or as *errors*.

Fortunately, `Pygments` has a method to attach filters to the lexer and perform additional modification to the lexer's output. Usually this filters are used to complement the lexer by highlighting additional keywords or special strings inside comments, such as `TODO`, `XXX`, etc.

In this case I'll use the filter architecture to detect `atangle` directives and inform the formatter to render them as *labels* instead of whatever the lexer believed it to be.

The filter, then, needs to look line by line and check whether a line is either a regular source code or an `atangle` directive. Given the nature of `atangle` directives, this can be accomplished using a regular expression.

```
<<atangle regex>>=
self.directive = r'''^\s*<<(\*|[-\w\s\.]+)>>=?\s*$'''
```

To use regular expressions the script needs to import the `re` module.

```
<<modules>>=
import re
```

This directive matches both the *declaration directive*, that starts a new `atangle` code snippet, as well as *reference directives*. Inside the filter, thus, we need to build an string until we match the end of line, either `\n` or `\r`, and then check against this regular expression. If the expression matches, then return the string as a label token. Otherwise, output whatever the lexer gave to the filter.

In some cases, Pygmens passes an string with double end of line characters, such as when it find an invalid syntax. To get these cases into account, I look for a value whose first character is either \n or \r instead of looking at the whole string.

Also, in some languages, notably XML, since the < and > characters are part of the language itself, Pygmens also tends to give the = as a separate value. Then, I also need to check for this character at first value's position.

```
<<atangle filter>>=
class AtangleFilter(Filter):
    def __init__(self, **options):
        Filter.__init__(self, **options)
        <<atangle regex>>

    def filter(self, lexer, stream):
        lexer_input = []
        line = ""
        for ttype, value in stream:
            if len(value) > 0 and (value[0] == '\n' or value[0] == '\r' or value[0] == '=') ↩
                :
                if re.match(self.directive, line):
                    yield Token.Name.Label, line
                else:
                    for original_ttype, original_value in lexer_input:
                        yield original_ttype, original_value
                # The end of line also needs to be there.
                yield ttype, value
                # start with the next line.
                lexer_input = []
                line = ""
            else:
                lexer_input.append((ttype, value))
                line = line + value;
```

It is also necessary to yield any buffered values from the lexer. This is to avoid losing tokens when there is no line that starts with a newline character and thus the original tokens would never be yielded.

```
<<atangle filter>>=
        for ttype, value in lexer_input:
            yield ttype, value
```

`AtangleFilter` class derives from `Filter` which is defined in the `Pygments` package.

```
<<modules>>=
from pygments.filter import Filter
```

The only thing that remains is to tell the lexer to use this filter. This is done calling `add_filter` with the lexer.

```
<<add atangle filter to lexer>>=
lexer.add_filter(AtangleFilter())
```

3 Applying an XSLT stylesheet

Once we have the custom HTML syntax highlighter function, we now just need to apply an XSLT stylesheet to the DocBook document. We output the transformation's result directly to the standard output using the `print` function. We return the error log, if any, of applying the transformation.

```
<<apply xsl stylesheet>>=
def apply_xslt(stylesheet, document):
    """
    Transform ``document`` using the given ``stylesheet``.  Both
    must be lxml element trees.

    Return the error log of the transformation.
    """
    # Register extension function for highlighting
    <<define custom highlight function>>

    # perform transformation
    transform = etree.XSLT(stylesheet)
    print transform(document)
    return transform.error_log
```

We also need a function to print the error log. This function just prints out each error in a human readable form to the standard error.

```
<<print transformation errors>>=
def print_errors(errors):
    for error in errors:
        if error.type == etree.ErrorTypes.ERR_OK:
            # succes, so just print the message
            tmpl = '{0.message}'
        else:
            # print filename and columns
            tmpl = ('{0.level_name}:{0.filename}:{0.line},{0.column}: '
                    '{0.message} ({0.type_name})')
        print >> sys.stderr, tmpl.format(error)
```

For this to work, we need to `sys` package:

```
<<modules>>=
import sys
```

4 Getting the document and XSLT stylesheet

The last thing we need is to read and parse the actual DocBook document and the XSLT stylesheet. We'll get these two from the user as command line parameters. Following the example of `xsltproc`, we will expect the first parameter to be the XSLT stylesheet and the second the DocBook document. We also need to call the `xinclude()` once we've parsed the DocBook document in order to include any possible referenced XML file as part of the document.

```
<<read and parse documents>>=
def main():
    if len(sys.argv) < 3:
        print >> sys.stderr, 'missing arguments'
        return 1
    elif len(sys.argv) > 3:
        print >> sys.stderr, 'too many arguments'
        return 1
    xslt_file, xml_file = sys.argv[1:]
```

```
    document = etree.parse(xml_file)
    document.xinclude()
    stylesheet = etree.parse(xslt_file)
    print_errors(apply_xslt(stylesheet, document))
```

This `main` function will be the first function called and the one that drives all the transformation. Thus, if this module is not included, we just need to call it.

```
<<read and parse documents>>=
if __name__ == '__main__':
    try:
        sys.exit(main())
    except KeyboardInterrupt:
        pass
```

5 Shebang

In order to allow the Python program to work as an executable file, we must add the traditional shebang line at the beginning.

We assume that the `python` interpreter is installed and accessible from the environment's `PATH`. We also specify that the source code is written in UTF-8.

```
<<shebang>>=
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

6 db2html.py

A simple Python script will incorporate all the elements we defined in the previous sections in the correct order:

```
<<*>>=
<<shebang>>
<<license>>
<<modules>>

<<atangle filter>>

<<pygments html highlighter>>

<<apply xsl stylesheet>>

<<print transformation errors>>

<<read and parse documents>>
```

A Using *db2html.py*

In order to use *db2html.py* we need a customization layer to enable the highlighting as well as to include Pygment's CSS stylesheet:

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

```
 <xsl:import href="http://docbook.sourceforge.net/release/xsl/current/xhtml/docbook.xsl"/>
 <xsl:import href="http://docbook.sourceforge.net/release/xsl/current/xhtml/highlight.xsl"/ ↩
     >
 <xsl:param name="html.stylesheet">highlight.css</xsl:param>
 <xsl:param name="highlight.source" select="1"/>
</xsl:stylesheet>
```

To generate the stylesheet use *pygmentize*:

```
pygmentize -S friendly -f html > highlight.css
```

Invoke *db2html.py* passing the XSLT stylesheet and the DocBook document. In this example, we assume that `xhtml.xsl` is the customization layer and `db2html.xml` the DocBook document:

```
db2html.py xhtml.xsl db2html.xml
```

B License

This program is distributed under the following license:

```
<<license>>=
# Copyright (c) 2009, 2010 Sebastian Wiesner <lunaryorn@googlemail.com>
# Copyright (c) 2010 Jordi Fita <jfita@geishastudios.com>

# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.
```

C Credits

Most of this document's contents and source code is a straight copy with slightly modifications from Pygments as syntax highlighter for DocBook documents written by Sebastian Wiesner.

D Download

The tangled Python source file is available at:

http://www.geishastudios.com/download/db2html.py

Also, for those interested in the AsciiDoc document, the latest version is always available at:

http://dev.geishastudios.com/literate/src/tip/db2html.txt