Syntax highlighting with pygments and fop

Table of Contents

Introduction

This is a simple example with all the configuration required to show this method of including syntax highlighting working. Asciidoc and docbook are run with all the default options apart from those required to enable syntax highlighting and a couple of settings to give more space for the listings.

Check everything is installed

First you need to check that the xslfo formatter and pygmentize are installed and available. You should be able to run the following command the see the line containing xslfo among the output lines.

```
$ pygmentize -L formatters
Formatters:
~~~~~~~~~~~
[... omitted output ...]
* xslfo, xsl-fo:
    Format tokens as XSL-FO ``<fo:inline>`` tags.
```

If this is not the case, then the following isn't going to work so go back and make sure everything is installed properly and available in the environment you are running in.

Creating the example

The makefile basically runs the following command

```
a2x -L --xsl-file=fo.xsl --fop \
    --asciidoc-opts='--conf-file=local-docbook45.conf' \
    -a 'source-highlighter=pygments' \
    -a 'pygments-style=default' \
    example.adoc
```

You need the `-L` flag to disable validation, and it is useful to add the `-v` flag too if there are any problems.

There are two files that need to be customised. You will probably already have suitable customisation files already, but if not you can use the exact ones in this directory.

The next sections discuss the files needed.

- fo.xsl - adds a new template for fo:block
- local-docbook45.conf - to set up pygmentize for docbook

The fo.xsl file

You will probably already have a customised fo.xsl file. If so, you just need to add the template from the following complete file.

```
<!--
  Generates single FO document from DocBook XML source using DocBook XSL
  stylesheets.

  See xsl-stylesheets/fo/param.xsl for all parameters.

  NOTE: The URL reference to the current DocBook XSL stylesheets is
  rewritten to point to the copy on the local disk drive by the XML catalog
  rewrite directives so it doesn't need to go out to the Internet for the
  stylesheets. This means you don't need to edit the <xsl:import> elements on
  a machine by machine basis.
-->
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:import href="http://docbook.sourceforge.net/release/xsl/current/fo/docbook.xsl"/>


<!-- This is the only thing that needs to be added for
     the pygments-xslfo-formatter -->
<xsl:template match="fo:inline|fo:block">
    <xsl:copy-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

The local-docbook45.conf file

This file contains the configuration that makes calls to pymentize. It is very much like the configuration for using pygmentize with html, except that you pass the extra argument `-f xslfo`. There is also a configuration parameter `pygments-style` that you can pass on the `a2x` command line as in the example below to change the pygments style that is used.

```
ifdef::basebackend-docbook[]
ifeval::["{source-highlighter}"=="pygments"]
[source-filter-style]
# This should be all one line, broken for readability
source-style=template="source-highlight-block",presubs=(),
  postsubs=("callouts",),
  posattrs=("style","language","src_numbered","src_tab"),
  filter="pygmentize -l {language} -f xslfo
            {pygments-style?-Ostyle={pygments-style}} {args=}"

##########################
# Source paragraph styles
##########################
[paradef-default]
template::[source-filter-style]
```

```
[paradef-literal]
template::[source-filter-style]

#########################
# Source block styles
#########################
[blockdef-open]
template::[source-filter-style]

[blockdef-listing]
template::[source-filter-style]

endif::[]
endif::basebackend-docbook[]
```

You can call this file whatever you want, if you already have a asciidoc configuration file, then just add the lines to it.

The example makefile

The makefile shows the command options to pass. Note that you have to pass `-L` to skip validation, since it will fail due to the added fo:block element.

```
A2X=a2x

# Pass source-highlighter=pygments to enable
# Pass eg. pygments-style=tango to change the pygments style to 'tango'
# or whatever you prefer.
PDF_OPTS=\
    --asciidoc-opts='--conf-file=local-docbook45.conf' \
    -a source-highlighter=pygments \
    -a pygments-style=default

all: example.pdf

example.pdf: example.adoc
    $(A2X) -L --xsl-file=fo.xsl --fop $(PDF_OPTS) example.adoc

clean:
    rm -f example.pdf
```

The source code

Here is a version of the source code, which is included here as an extra example of syntax highlighting.

```
# -*- coding: utf-8 -*-
"""
    pygments.formatters.html
    ~~~~~~~~~~~~~~~~~~~~~~~~

    Formatter for XSL-FO output.

    :copyright: Copyright 2014, Steve Ratcliffe
    :license: BSD, see LICENSE for details.
"""

from pygments.formatter import Formatter
from pygments.util import bytes
```

```
__all__ = ['XslfoFormatter']


_escape_html_table = {
    ord('&'): u'&amp;',
    ord('<'): u'&lt;',
    ord('>'): u'&gt;',
    ord('"'): u'&quot;',
    ord("'"): u'&#39;',
    }


def escape_html(text, table=_escape_html_table):
    """Escape &, <, > as well as single and double quotes for HTML."""
    return text.translate(table)


class XslfoFormatter(Formatter):
    r"""
    Format tokens as XSL-FO ``<fo:inline>`` tags.
    """

    name = 'XSL-FO formatter'
    aliases = ['xslfo', 'xsl-fo']
    filenames = []

    def __init__(self, **options):
        Formatter.__init__(self, **options)
        self.title = self._decodeifneeded(self.title)

        # Holds fo attributes that correspond to each style type.
        self.styles = {}
        self.background_colour = self.style.background_color or '#ffffff'

        # Iterate over the style and create a list of fo attributes that describe it.
        for token, style in self.style:
            atts = []
            if style['color']:
                atts.append('color="#%s"' % style['color'])
            if style['bold']:
                atts.append('font-weight="bold"')
            if style['italic']:
                atts.append('font-style="italic"')
            if style['underline']:
                atts.append('text-decoration="underline"')

            # Save the attributes for this style
            self.styles[token] = ' '.join(atts)

    def _decodeifneeded(self, value):
        if isinstance(value, bytes):
            if self.encoding:
                return value.decode(self.encoding)
            return value.decode()
        return value

    def format_unencoded(self, tokensource, outfile):
        """
        Read the token source and write it marked up with fo:inline styles.
```

```
        We join together tokens with the same token type and then surround the
        text with a fo:inline tag with appropriate attributes for the style
        element.

        The whole thing is wrapped in a fo:block tag.
        """

        def write_value(t, val):
            atts = self.styles[t]
            outfile.write('<fo:inline %s>%s</fo:inline>' % (atts, escape_html(val)))

        outfile.write(
            '<fo:block background-color="%s" xmlns:fo="http://www.w3.org/1999/XSL/Format">'
                % self.background_colour)

        lastval = ''
        lasttype = None
        for ttype, value in tokensource:
            while ttype not in self.styles:
                ttype = ttype.parent

            if ttype == lasttype:
                lastval += value
            else:
                if lastval:
                    write_value(lasttype, lastval)
                lastval = value
                lasttype = ttype

        if lastval:
            write_value(lasttype, lastval)

        outfile.write('</fo:block>')
```