Source Code Highlight Filter

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
8.6.8	17 July 2012		

Contents

The AsciiDoc distribution includes a *source* filter for highlighting code syntax.

1 DocBook Outputs

AsciiDoc encloses the source code in a DocBook *programlisting* element and leaves source code highlighting to the DocBook toolchain (dblatex has a particularly nice programlisting highlighter). The DocBook programlisting element is assigned two attributes:

1. The *language* attribute is set to the AsciiDoc *language* attribute.
2. The *linenumbering* attribute is set to the AsciiDoc *src_numbered* attribute (*numbered* or *unnumbered*).

2 HTML Outputs

You have the choice of three HTML source code highlighters, your selection is determined by the *source-highlighter* attribute (defaults to *source-highlight*):

Note
Set the *source-highlighter* attribute from the `asciidoc(1)` command-line or in the document header (not in the document body, because the configuration file conditional macros are processed at load time).

2.1 GNU Source Highlight

The default highlighter is the GNU source-highlight which can highlight *html4*, *html5* and *xhtml11* outputs. The GNU source-highlight must be installed and the *source-highlight* command must reside in the shell search *PATH*.

2.2 Highlight

You can use Highlight syntax highlighter for *xhtml11*, *html5* and *html4* outputs (set the *source-highlighter* attribute to *highlighter*).

- The *highlight* command must reside in the shell search *PATH*.
- To make Highlighter your default highlighter put the following line your `~/.asciidoc/asciidoc.conf` file:

```
source-highlighter=highlight
```

- The AsciiDoc *encoding* attribute is passed to Highlighter using the `--encoding` command-line option.

2.3 Pygments

The Pygments syntax highlighter can be used for *xhtml11* and *html5* outputs (set the *source-highlighter* attribute to *pygments*).

- The *pygmentize* command must reside in the shell search *PATH*.
- You can customize Pygments CSS styles by editing `./stylesheets/pygments.css`.
- To make Pygments your default highlighter put the following line your `~/.asciidoc/asciidoc.conf` file:

```
source-highlighter=pygments
```

- The AsciiDoc *encoding* attribute is passed to Pygments using the `-O` command-line option.

3 Block attributes

The following attributes can be included in source code block attribute lists.

- *style* and *language* are mandatory.
- *style*, *language* and *src_numbered* are the first three positional attributes in that order.
- The *args* attribute allows the inclusion of arbitrary (highlighter dependent) command options.

style
: Set to *source*.

language
: The source code language name.

> **Note**
> The language names vary between highlighters — consult the selected highlighter manual.

src_numbered
: Set to *numbered* to include line numbers.

src_tab
: Set tab size (GNU source-highlight only).

args
: Include this attribute value in the highlighter command-line (HTML outputs) or in the `programlisting` element (DocBook).

4 Testing

Test the filter by converting the test file to HTML with AsciiDoc:

```
$ asciidoc -v ./filters/source/source-highlight-filter-test.txt
$ firefox ./filters/source/source-highlight-filter-test.html &
```

5 Examples

5.1 Source code paragraphs

The `source` paragraph style will highlight a paragraph of source code. These three code paragraphs:

```
[source,python]
if n < 0: print 'Hello World!'

:language: python

[source]
if n < 0: print 'Hello World!'

[source,ruby,numbered]
[true, false].cycle([0, 1, 2, 3, 4]) do |a, b|
    puts "#{a.inspect} => #{b.inspect}"
```

Render this highlighted source code:

```
if n < 0: print 'Hello World!'
```

```
if n < 0: print 'Hello World!'
```

```
[true, false].cycle([0, 1, 2, 3, 4]) do |a, b|
    puts "#{a.inspect} => #{b.inspect}"
```

5.2 Unnumbered source code listing

This source-highlight filtered block:

```
 [source,python]
 ---------------------------------------------------------------------
 ''' A multi-line
     comment.'''
 def sub_word(mo):
     ''' Single line comment.'''
     word = mo.group('word')    # Inline comment
     if word in keywords[language]:
         return quote + word + quote
     else:
         return word
 ---------------------------------------------------------------------
```

Renders this highlighted source code:

```
''' A multi-line
    comment.'''
def sub_word(mo):
    ''' Single line comment.'''
    word = mo.group('word')     # Inline comment
    if word in keywords[language]:
        return quote + word + quote
    else:
        return word
```

5.3 Numbered source code listing with callouts

This source-highlight filtered block:

```
 [source,ruby,numbered]
 ---------------------------------------------------------------------
 #
 # Useful Ruby base class extensions.
 #

 class Array

   # Execute a block passing it corresponding items in
   # +self+ and +other_array+.
   # If self has less items than other_array it is repeated.

   def cycle(other_array)   # :yields: item, other_item
     other_array.each_with_index do |item, index|
       yield(self[index % self.length], item)
     end
```

```
  end

end

if $0 == __FILE__                                 # <1>
  # Array#cycle test
  # true => 0
  # false => 1
  # true => 2
  # false => 3
  # true => 4
  puts 'Array#cycle test'                         # <2>
  [true, false].cycle([0, 1, 2, 3, 4]) do |a, b|
    puts "#{a.inspect} => #{b.inspect}"
  end
end
----------------------------------------------------------------------

<1> First callout.
<2> Second callout.
```

Renders this highlighted source code:

```
#
# Useful Ruby base class extensions.
#

class Array

  # Execute a block passing it corresponding items in
  # +self+ and +other_array+.
  # If self has less items than other_array it is repeated.

  def cycle(other_array)  # :yields: item, other_item
    other_array.each_with_index do |item, index|
      yield(self[index % self.length], item)
    end
  end

end

if $0 == __FILE__                                 # ❶
  # Array#cycle test
  # true => 0
  # false => 1
  # true => 2
  # false => 3
  # true => 4
  puts 'Array#cycle test'                         # ❷
  [true, false].cycle([0, 1, 2, 3, 4]) do |a, b|
    puts "#{a.inspect} => #{b.inspect}"
  end
end
```

❶ First callout.

❷ Second callout.

Tip

- If the source *language* attribute has been set (using an *AttributeEntry* or from the command-line) you don't have to specify it in each source code block.
- You should place callout markers inside source code comments to ensure they are not misinterpreted and mangled by the highlighter.