Wiki Processors

Processors are Wiki Macros that provide alternative markup formats for the Wiki engine. Processors can be thought of as *macro functions to process user-edited text*.

Wiki processors can be used in any Wiki text throughout Trac, such as:

- syntax highlighting or for rendering text verbatim
- rendering Wiki markup inside a context such as <div> or <span> blocks or within <td> or <th> table cells
- using an alternative markup syntax, like raw HTML and Restructured Text or ?textile

Using Processors

To use a processor on a block of text, first delimit the lines using a Wiki *code block*:

```
{{{
The lines
that should be processed...
}}}
```

Immediately after the `{{{` or on the line just below, add `#!` followed by the *processor name*:

```
{{{
#!processorname
The lines
that should be processed...
}}}
```

This is the "shebang" notation, familiar to most UNIX users.

Besides their content, some Wiki processors can also accept *parameters*, which are then given as `key=value` pairs after the processor name and on the same line. If `value` has to contain space, as it's often the case for the style parameter, a quoted string can be used (`key="value with space"`).

As some processors are meant to process Wiki markup, it's quite possible to *nest* processor blocks. You may want to indent the content of nested blocks for increased clarity, this extra indentation will be ignored when processing the content.

Examples

Wiki Markup	Display
Example 1: Inserting raw HTML	

`{{{`
`#!html`
`<h1 style="color: grey">This is raw HTML</h1>`
`}}}`
	This is raw HTML

Example 2: Highlighted Python code in a <div> block with custom style	

`{{{#!div style="background: #ffd; border: 3px ridge"`
 	
`This is an example of embedded "code" block:`
 	
`  {{{`
`  #!python`
`  def hello():`
	This is an example of embedded "code" block:
	
	`def hello():`
	`    return "world"`

Wiki Markup	Display

```
      return "world"
  }}}

}}}
```

Example 3: Searching tickets from a wiki page, by keywords.

```
{{{
#!html
<form action="/query" method="get"><div>
<input type="text" name="keywords" value="~" size="30"/>
<input type="submit" value="Search by Keywords"/>
<!-- To control what fields show up use hidden fields
<input type="hidden" name="col" value="id"/>
<input type="hidden" name="col" value="summary"/>
<input type="hidden" name="col" value="status"/>
<input type="hidden" name="col" value="milestone"/>
<input type="hidden" name="col" value="version"/>
<input type="hidden" name="col" value="owner"/>
<input type="hidden" name="col" value="priority"/>
<input type="hidden" name="col" value="component"/>
-->
</div></form>
}}}
```

Available Processors

The following processors are included in the Trac distribution:

`#!default`	Present the text verbatim in a preformatted text block. This is the same as specifying *no* processor name (and no `#!`).
`#!comment`	Do not process the text in this section, i.e. contents exist only in the plain text - not in the rendered page.
`#!rtl`	Introduce a Right-To-Left block with appropriate CSS direction and styling. *(since 0.12.2)*
	HTML related
`#!html`	Insert custom HTML in a wiki page.
`#!htmlcomment`	Insert an HTML comment in a wiki page. (*since 0.12*)
	Note that `#!html` blocks have to be *self-contained*, i.e. you can't start an HTML element in one block and close it later in a second block. Use the following processors for achieving a similar effect.
`#!div`	Wrap wiki content inside a <div> element.
`#!span`	Wrap wiki content inside a <span> element.
`#!td`	Wrap wiki content inside a <td> element. (*since 0.12*)
`#!th`	Wrap wiki content inside a <th> element. (*since 0.12*)
`#!tr`	Can optionally be used for wrapping `#!td` and `#!th` blocks, either for specifying row attributes or better visual grouping. (*since 0.12*)
`#!table`	Can optionally be used for wrapping `#!tr`, `#!td` and `#!th` blocks, for specifying table attributes. One current limitation however is that tables cannot be nested. (*since 0.12*)
	See Wiki Html for example usage and more details about these processors.
	Other Markups
`#!rst`	Trac support for Restructured Text. See Wiki Restructured Text.
`#!textile`	Supported if ?Textile is installed. See ?a Textile reference.

Code Highlighting Support

#!c
#!cpp (C++)
#!python
#!perl
#!ruby
#!php
#!asp
#!java
#!js (Javascript)
#!sql
#!xml (XML or HTML)
#!sh (Bourne/Bash shell)

Trac includes processors to provide inline syntax highlighting for source code in various languages.

Trac relies on ?Pygments for syntax coloring.

See Trac Syntax Coloring for information about which languages are supported and how to enable support for more languages.

Since 1.1.2 the default, coding highlighting and MIME-type processors support the argument `lineno` for adding line numbering to the code block. When a value is specified, as in `lineno=3`, the numbering will start at the specified value. When used in combination with the `lineno` argument, the `marks` argument is also supported for highlighting lines. A single line number, set of line numbers and range of line numbers are allowed. For example, `marks=3`, `marks=3-6`, `marks=3,5,7` and `marks=3-5,7` are all allowed. The specified values are relative to the numbered lines, so if `lineno=2` is specified to start the line numbering at 2, `marks=2` will result in the first line being highlighted.

Using the MIME type as processor, it is possible to syntax-highlight the same languages that are supported when browsing source code.

MIME Type Processors

Some examples:

```
{{{#!text/html
<h1>text</h1>
}}}
```

The result will be syntax highlighted HTML code:

```
<h1>text</h1>
```

The same is valid for all other mime types supported.

#!diff has a particularly nice renderer:

```
{{{#!diff
--- Version 55
+++ Version 56
@@ -115,8 +115,9 @@
     name='TracHelloWorld', version='1.0',
     packages=find_packages(exclude=['*.tests*']),
-    entry_points = """
-        [trac.plugins]
-        helloworld = myplugs.helloworld
-    """,
+    entry_points = {
+        'trac.plugins': [
+            'helloworld = myplugs.helloworld',
+        ],
+    },
 )
}}}
```

- **Version**

```
115 115     name='TracHelloWorld', version='1.0',
116 116  packages=find_packages(exclude=['*.tests*']
117         entry_points = """
118            [trac.plugins]
119            helloworld = myplugs.helloworld
120         """,
    117     entry_points = {
    118        'trac.plugins': [
    119            'helloworld = myplugs.helloworld',
    120        ],
    121     },
121 122 )
```

Line numbers can be added to code blocks and lines can be highlighted *(since 1.1.2)*.

```
{{{#!python lineno=3 marks=3,9-10,16
def expand_markup(stream, ctxt=None):
    """A Genshi stream filter for expanding `genshi.Markup` events.

    Note: Expansion may not be possible if the fragment is badly
    formed, or partial.
    """
    for event in stream:
        if isinstance(event[1], Markup):
            try:
                for subevent in HTML(event[1]):
                    yield subevent
            except ParseError:
                yield event
        else:
            yield event
}}}
```

```
Line
  3  def expand_markup(stream, ctxt=None):
  4      """A Genshi stream filter for expanding `genshi.Markup` events.
  5  
  6      Note: Expansion may not be possible if the fragment is badly
  7      formed, or partial.
  8      """
  9      for event in stream:
 10          if isinstance(event[1], Markup):
 11              try:
 12                  for subevent in HTML(event[1]):
 13                      yield subevent
 14              except ParseError:
 15                  yield event
 16          else:
 17              yield event
```

For more processor macros developed and/or contributed by users, visit the ?Trac Hacks community site.

Processors are implemented using the same interfaces as Wiki macros, only the usage syntax differs. To develop a processor, see Wiki Macros#Developing Custom Macros.

See also: Wiki Macros, Wiki Html, Wiki Restructured Text, Trac Syntax Coloring, Wiki Formatting, Trac Guide