Trac Macros

Trac macros are plugins to extend the Trac engine with custom 'functions' written in Python. A macro inserts dynamic HTML data in any context supporting WikiFormatting.

Another kind of macros are WikiProcessors. They typically deal with alternate markup formats and representation of larger blocks of information (like source code highlighting).

Using Macros

Macro calls are enclosed in two *square brackets*. Like Python functions, macros can also have arguments, a comma separated list within parentheses.

Getting Detailed Help

The list of available macros and the full help can be obtained using the MacroList macro, as seen below.

A brief list can be obtained via `[[MacroList(*)]]` or `[[?]]`.

Detailed help on a specific macro can be obtained by passing it as an argument to MacroList, e.g. `[[MacroList(MacroList)]]`, or, more conveniently, by appending a question mark (`?`) to the macro's name, like in `[[MacroList?]]`.

Example

A list of 3 most recently changed wiki pages starting with 'Trac':

Wiki Markup

```
[[RecentChanges(Trac,3)]]
```

Display

Apr 24, 2020

- TracWorkflow (diff)
- TracWiki (diff)
- TracUpgrade (diff)

```
[[RecentChanges?(Trac,3)]]
```

`[[RecentChanges]]`

List all pages that have recently been modified, ordered by the time they were last modified.

Wiki Markup	Display

	This macro accepts two ordered arguments and a named argument. The named argument can be placed in any position within the argument list.
	
	The first parameter is a prefix string: if provided, only pages with names that start with the prefix are included in the resulting list. If this parameter is omitted, all pages are included in the list.
	
	The second parameter is the maximum number of pages to include in the list.
	
	The `group` parameter determines how the list is presented:
	
	`group=date`
	The pages are presented in bulleted lists that are grouped by date (default).
	`group=none`
	The pages are presented in a single bulleted list.
	
	Tip: if you only want to specify a maximum number of entries and don't want to filter by prefix, specify an empty first parameter, e.g. `[[RecentChanges(,10,group=none)]]`.

	[[Image]] Embed an image in wiki-formatted text. The first argument is the file ?
	[[InterTrac]] Provide a list of known InterTrac prefixes.
`[[?]]`	**[[InterWiki]]** Provide a description list for the known InterWiki prefixes.
	[[KnownMimeTypes]] List all known mime-types which can be used as WikiProcessors. Can be ? etc.

Available Macros

Note that the following list will only contain the macro documentation if you've not enabled `-OO` optimizations, or not set the `PythonOptimize` option for mod_python.

[[Image]]

Embed an image in wiki-formatted text.

The first argument is the file specification. The file specification may reference attachments in three ways:

- `module:id:file`, where module can be either **wiki** or **ticket**, to refer to the attachment named *file* of the specified wiki page or ticket.
- `id:file`: same as above, but id is either a ticket shorthand or a Wiki page name.

- `file` to refer to a local attachment named 'file'. This only works from within that wiki page or a ticket.

The file specification may also refer to:

- repository files, using the `source:file` syntax (`source:file@rev` works also).
- files, using direct URLs: `/file` for a project-relative, `//file` for a server-relative, or `http://server/file` for absolute location. An InterWiki prefix may be used.
- embedded data using the ?rfc2397 `data` URL scheme, provided the URL is enclosed in quotes.

The remaining arguments are optional and allow configuring the attributes and style of the rendered `<img>` element:

- digits and unit are interpreted as the size (ex. 120px, 25%) for the image
- `right`, `left`, `center`, `top`, `bottom` and `middle` are interpreted as the alignment for the image (alternatively, the first three can be specified using `align=...` and the last three using `valign=...`)
- `link=some TracLinks...` replaces the link to the image source by the one specified using a TracLinks. If no value is specified, the link is simply removed.
- `inline` specifies that the content generated be an inline XHTML element. By default, inline content is not generated, therefore images won't be rendered in section headings and other one-line content.
- `nolink` means without link to image source (deprecated, use `link=`)
- `key=value` style are interpreted as HTML attributes or CSS style indications for the image. Valid keys are:
  - ♦ align, valign, border, width, height, alt, title, longdesc, class, margin, margin-(left,right,top,bottom), id and usemap
  - ♦ `border`, `margin`, and `margin-*` can only be a single number (units are pixels).
  - ♦ `margin` is superseded by `center` which uses auto margins

Examples:

```
[[Image(photo.jpg)]]                # simplest
[[Image(photo.jpg, 120px)]]         # with image width size
[[Image(photo.jpg, right)]]         # aligned by keyword
[[Image(photo.jpg, nolink)]]        # without link to source
[[Image(photo.jpg, align=right)]]   # aligned by attribute
```

You can use an image from a wiki page, ticket or other module.

```
[[Image(OtherPage:foo.bmp)]]    # from a wiki page
[[Image(base/sub:bar.bmp)]]     # from hierarchical wiki page
[[Image(#3:baz.bmp)]]           # from another ticket
[[Image(ticket:36:boo.jpg)]]    # from another ticket (long form)
[[Image(source:/img/bee.jpg)]]  # from the repository
[[Image(htdocs:foo/bar.png)]]   # from project htdocs dir
[[Image(shared:foo/bar.png)]]   # from shared htdocs dir (since 1.0.2)
```

Adapted from the Image.py macro created by Shun-ichi Goto <gotoh@?>

[[InterTrac]]

Provide a list of known InterTrac prefixes.

[[InterWiki]]

Provide a description list for the known InterWiki prefixes.

[[KnownMimeTypes]]

List all known mime-types which can be used as WikiProcessors.

Can be given an optional argument which is interpreted as mime-type filter.

[[MacroList]]

Display a list of all installed Wiki macros, including documentation if available.

Optionally, the name of a specific macro can be provided as an argument. In that case, only the documentation for that macro will be rendered.

Note that this macro will not be able to display the documentation of macros if the `PythonOptimize` option is enabled for mod_python!

[[PageOutline]]

Display a structural outline of the current wiki page, each item in the outline being a link to the corresponding heading.

This macro accepts four optional parameters:

- The first is a number or range that allows configuring the minimum and maximum level of headings that should be included in the outline. For example, specifying "1" here will result in only the top-level headings being included in the outline. Specifying "2-3" will make the outline include all headings of level 2 and 3, as a nested list. The default is to include all heading levels.
- The second parameter can be used to specify a custom title (the default is no title).
- The third parameter selects the style of the outline. This can be either `inline` or `pullout` (the latter being the default). The `inline` style renders the outline as normal part of the content, while `pullout` causes the outline to be rendered in a box that is by default floated to the right side of the other content.
- The fourth parameter specifies whether the outline is numbered or not. It can be either `numbered` or `unnumbered` (the former being the default). This parameter only has an effect in `inline` style.

[[RecentChanges]]

List all pages that have recently been modified, ordered by the time they were last modified.

This macro accepts two ordered arguments and a named argument. The named argument can be placed in any position within the argument list.

The first parameter is a prefix string: if provided, only pages with names that start with the prefix are included in the resulting list. If this parameter is omitted, all pages are included in the list.

The second parameter is the maximum number of pages to include in the list.

The `group` parameter determines how the list is presented:

`group=date`
: The pages are presented in bulleted lists that are grouped by date (default).

`group=none`
: The pages are presented in a single bulleted list.

Tip: if you only want to specify a maximum number of entries and don't want to filter by prefix, specify an empty first parameter, e.g. `[[RecentChanges(,10,group=none)]]`.

[[RepositoryIndex]]

Display the list of available repositories.

Can be given the following named arguments:

format
: Select the rendering format:
  ◊ *compact* produces a comma-separated list of repository prefix names (default)
  ◊ *list* produces a description list of repository prefix names
  ◊ *table* produces a table view, similar to the one visible in the *Browse View* page

glob
: Do a glob-style filtering on the repository names (defaults to '*')

order
: Order repositories by the given column (one of "name", "date" or "author")

desc
: When set to 1, order by descending order

(*since 0.12*)

[[SubscriberList]]

Display a list of all installed notification subscribers, including documentation if available.

Optionally, the name of a specific subscriber can be provided as an argument. In that case, only the documentation for that subscriber will be rendered.

Note that this macro will not be able to display the documentation of subscribers if the `PythonOptimize` option is enabled for mod_python!

[[TicketQuery]]

Wiki macro listing tickets that match certain criteria.

This macro accepts a comma-separated list of keyed parameters, in the form "key=value".

If the key is the name of a field, the value must use the syntax of a filter specifier as defined in TracQuery#QueryLanguage. Note that this is *not* the same as the simplified URL syntax used for `query:` links starting with a `?` character. Commas (`,`) can be included in field values by escaping them with a backslash (`\`).

Groups of field constraints to be OR-ed together can be separated by a literal `or` argument.

In addition to filters, several other named parameters can be used to control how the results are presented. All of them are optional.

The `format` parameter determines how the list of tickets is presented:

- **list** -- the default presentation is to list the ticket ID next to the summary, with each ticket on a separate line.
- **compact** -- the tickets are presented as a comma-separated list of ticket IDs.

- **count** -- only the count of matching tickets is displayed
- **rawcount** -- only the count of matching tickets is displayed, not even with a link to the corresponding query (*since 1.1.1*)
- **table** -- a view similar to the custom query view (but without the controls)
- **progress** -- a view similar to the milestone progress bars

The `max` parameter can be used to limit the number of tickets shown (defaults to **0**, i.e. no maximum).

The `order` parameter sets the field used for ordering tickets (defaults to **id**).

The `desc` parameter indicates whether the order of the tickets should be reversed (defaults to **false**).

The `group` parameter sets the field used for grouping tickets (defaults to not being set).

The `groupdesc` parameter indicates whether the natural display order of the groups should be reversed (defaults to **false**).

The `verbose` parameter can be set to a true value in order to get the description for the listed tickets. For **table** format only. *deprecated in favor of the `rows` parameter*

The `rows` parameter can be used to specify which field(s) should be viewed as a row, e.g. `rows=description|summary`

The `col` parameter can be used to specify which fields should be viewed as columns. For **table** format only.

For compatibility with Trac 0.10, if there's a last positional parameter given to the macro, it will be used to specify the `format`. Also, using "&" as a field separator still works (except for `order`) but is deprecated.

`[[TitleIndex]]`

Insert an alphabetic list of all wiki pages into the output.

Accepts a prefix string as parameter: if provided, only pages with names that start with the prefix are included in the resulting list. If this parameter is omitted, all pages are listed. If the prefix is specified, a second argument of value `hideprefix` can be given as well, in order to remove that prefix from the output.

The prefix string supports the standard relative-path notation *when using the macro in a wiki page*. A prefix string starting with `./` will be relative to the current page, and parent pages can be specified using `../`.

Several named parameters can be specified:

- `format=compact`: The pages are displayed as comma-separated links.
- `format=group`: The list of pages will be structured in groups according to common prefix. This format also supports a `min=n` argument, where `n` is the minimal number of pages for a group.
- `format=hierarchy`: The list of pages will be structured according to the page name path hierarchy. This format also supports a `min=n` argument, where higher `n` flatten the display hierarchy
- `depth=n`: limit the depth of the pages to list. If set to 0, only toplevel pages will be shown, if set to 1, only immediate children pages will be shown, etc. If not set, or set to -1, all pages in the hierarchy will be shown.
- `include=page1:page*2`: include only pages that match an item in the colon-separated list of pages. If the list is empty, or if no `include` argument is given, include all pages.
- `exclude=page1:page*2`: exclude pages that match an item in the colon- separated list of pages.

The `include` and `exclude` lists accept shell-style patterns.

[[TracAdminHelp]]

Display help for trac-admin commands.

Examples:

```
[[TracAdminHelp]]                # all commands
[[TracAdminHelp(wiki)]]          # all wiki commands
[[TracAdminHelp(wiki export)]]   # the "wiki export" command
[[TracAdminHelp(upgrade)]]       # the upgrade command
```

[[TracGuideToc]]

Display a table of content for the Trac guide.

This macro shows a quick and dirty way to make a table-of-contents for the Help/Guide. The table of contents will contain the Trac* and WikiFormatting pages, and can't be customized. See the ?TocMacro for a more customizable table of contents.

[[TracIni]]

Produce documentation for the Trac configuration file.

Typically, this will be used in the TracIni page. The macro accepts two ordered arguments and two named arguments.

The ordered arguments are a configuration section filter, and a configuration option name filter: only the configuration options whose section and name start with the filters are output.

The named arguments can be specified:

section
: a glob-style filtering on the section names

option
: a glob-style filtering on the option names

[[Workflow]]

Render a workflow graph.

This macro accepts a TracWorkflow configuration and renders the states and transitions as a directed graph. If no parameters are given, the current ticket workflow is rendered. In WikiProcessors mode the `width` and `height` arguments can be specified.

(Defaults: `width = 800` and `height = 600`)

Examples:

```
[[Workflow()]]

[[Workflow(go = here -> there; return = there -> here)]]
```

```
{{{
#!Workflow width=700 height=700
leave = * -> *
leave.operations = leave_status
leave.default = 1

create = <none> -> new
create.default = 1

create_and_assign = <none> -> assigned
create_and_assign.label = assign
create_and_assign.permissions = TICKET_MODIFY
create_and_assign.operations = may_set_owner

accept = new,assigned,accepted,reopened -> accepted
accept.permissions = TICKET_MODIFY
accept.operations = set_owner_to_self

resolve = new,assigned,accepted,reopened -> closed
resolve.permissions = TICKET_MODIFY
resolve.operations = set_resolution

reassign = new,assigned,accepted,reopened -> assigned
reassign.permissions = TICKET_MODIFY
reassign.operations = set_owner

reopen = closed -> reopened
reopen.permissions = TICKET_CREATE
reopen.operations = del_resolution
}}}
```

Macros from around the world

The ?Trac Hacks site provides a wide collection of macros and other Trac plugins contributed by the Trac community. If you're looking for new macros, or have written one that you'd like to share with the world, please don't hesitate to visit that site.

Developing Custom Macros

Macros, like Trac itself, are written in the ?Python programming language and are developed as part of TracPlugins.

For more information about developing macros, see the ?development resources on the main project site.

Here are 2 simple examples showing how to create a Macro with Trac 0.11.

Also, have a look at ?Timestamp.py for an example that shows the difference between old style and new style macros and at the ?macros/README which provides a little more insight about the transition.

Macro without arguments

To test the following code, you should saved it in a `timestamp_sample.py` file located in the TracEnvironment's `plugins/` directory.

```
from datetime import datetime
# Note: since Trac 0.11, datetime objects are used internally

from genshi.builder import tag
```

```
from trac.util.datefmt import format_datetime, utc
from trac.wiki.macros import WikiMacroBase

class TimeStampMacro(WikiMacroBase):
    """Inserts the current time (in seconds) into the wiki page."""

    revision = "$Rev$"
    url = "$URL$"

    def expand_macro(self, formatter, name, text):
        t = datetime.now(utc)
        return tag.b(format_datetime(t, '%c'))
```

Macro with arguments

To test the following code, you should saved it in a `helloworld_sample.py` file located in the TracEnvironment's `plugins/` directory.

```
from genshi.core import Markup

from trac.wiki.macros import WikiMacroBase

class HelloWorldMacro(WikiMacroBase):
    """Simple HelloWorld macro.

    Note that the name of the class is meaningful:
     - it must end with "Macro"
     - what comes before "Macro" ends up being the macro name

    The documentation of the class (i.e. what you're reading)
    will become the documentation of the macro, as shown by
    the !MacroList macro (usually used in the WikiMacros page).
    """

    revision = "$Rev$"
    url = "$URL$"

    def expand_macro(self, formatter, name, text, args):
        """Return some output that will be displayed in the Wiki content.

        `name` is the actual name of the macro (no surprise, here it'll be
        `'HelloWorld'`),
        `text` is the text enclosed in parenthesis at the call of the macro.
          Note that if there are ''no'' parenthesis (like in, e.g.
          [[HelloWorld]]), then `text` is `None`.
        `args` are the arguments passed when HelloWorld is called using a
        `#!HelloWorld` code block.
        """
        return 'Hello World, text = %s, args = %s' % \
            (Markup.escape(text), Markup.escape(repr(args)))
```

Note that `expand_macro` optionally takes a 4th parameter *`args`*. When the macro is called as a WikiProcessor, it's also possible to pass `key=value` processor parameters. If given, those are stored in a dictionary and passed in this extra `args` parameter. On the contrary, when called as a macro, `args` is `None`. (*since 0.12*).

For example, when writing:

```
{{{#!HelloWorld style="polite" -silent verbose
<Hello World!>
}}}

{{{#!HelloWorld
```

```
<Hello World!>
}}}

[[HelloWorld(<Hello World!>)]]
```

One should get:

```
Hello World, text = <Hello World!> , args = {'style': u'polite', 'silent': False, 'verbose': Tru
Hello World, text = <Hello World!> , args = {}
Hello World, text = <Hello World!> , args = None
```

Note that the return value of `expand_macro` is **not** HTML escaped. Depending on the expected result, you should escape it by yourself (using `return Markup.escape(result)`) or, if this is indeed HTML, wrap it in a Markup object (`return Markup(result)`) with `Markup` coming from Genshi, (`from genshi.core import Markup`).

You can also recursively use a wiki Formatter (`from trac.wiki import Formatter`) to process the `text` as wiki markup, for example by doing:

```
from genshi.core import Markup
from trac.wiki.macros import WikiMacroBase
from trac.wiki import Formatter
import StringIO

class HelloWorldMacro(WikiMacroBase):
        def expand_macro(self, formatter, name, text, args):
                text = "whatever '''wiki''' markup you want, even containing other macros"
                # Convert Wiki markup to HTML, new style
                out = StringIO.StringIO()
                Formatter(self.env, formatter.context).format(text, out)
                return Markup(out.getvalue())
```