swark Documentation

Release v1.2.7

Jan 06, 2020

Contents

1 Operators **3**

1.1 add_view_parameters . . . 3
1.2 array_search . . . 4
1.3 arsort . . . 4
1.4 asort . . . 5
1.5 charset . . . 5
1.6 clear_object_cache . . . 6
1.7 cookie . . . 6
1.8 current_layout . . . 7
1.9 current_siteaccess . . . 7
1.10 debug_attributes . . . 7
1.11 debug . . . 8
1.12 is_post_request . . . 9
1.13 json_encode . . . 9
1.14 krsort . . . 10
1.15 ksort . . . 10
1.16 ltrim . . . 11
1.17 modify_view_parameter . . . 11
1.18 preg_match . . . 12
1.19 preg_replace . . . 13
1.20 range . . . 13
1.21 redirect . . . 14
1.22 remove_array_element . . . 15
1.23 return . . . 16
1.24 rsort . . . 16
1.25 rtrim . . . 17
1.26 serialize . . . 17
1.27 server . . . 18
1.28 set_array_element . . . 19
1.29 shortenw . . . 19
1.30 shuffle . . . 20
1.31 sort . . . 20
1.32 split_by_length . . . 21
1.33 strpos . . . 22
1.34 str_replace . . . 22
1.35 strrpos . . . 23

1.36 substr . . . 23
1.37 unserialize . . . 24
1.38 uri_path_segment . . . 25
1.39 user_id_by_login . . . 25
1.40 variable_names . . . 26
1.41 embed_design_file . . . 26
1.42 ezselection_content . . . 27

2 Workflow event types **29**
2.1 autopriority . . . 29
2.2 defertocron . . . 29

3 Custom operators **31**
3.1 Constructor . . . 31
3.2 Execute . . . 32

Authors Jan Kudlicka, Seeds Consulting AS, http://www.seeds.no/

Aplia AS, https://www.aplia.no

Table of Contents

- *Swark*
  - *Operators*
  - *Workflow event types*
  - *Custom operators*

The Swark extension implements a set of highly needed template operators and workflow event types which are sorely missing from eZ publish legacy.

CHAPTER 1

Operators

1.1 add_view_parameters

1.1.1 Summary

Returns the input path extended by the view parameters.

1.1.2 Usage

```
input|add_view_parameters( view_parameters )
```

1.1.3 Parameters

Name	Description	Required	Default
view_parameters	View parameters to add to the input path	Yes	

1.1.4 Examples

```
{$node.url|add_view_parameters( $view_parameters )|ezurl}
```

In a node's full view template reconstructs the request URL. See also the modify_view_parameter operator.

1.2 array_search

1.2.1 Summary

Searches the input array for a given value and returns the corresponding key.

1.2.2 Usage

```
input|array_search( search [, not_found_key] )
```

1.2.3 Parameters

Name	Description	Required	Default
search	Searched value	Yes	
not_found_key	Value to return in the case the searched value is not found	No	-1

1.2.4 Examples

```
{array( 'apple', 'pear', 'pineapple', 'orange' )|array_search( 'pear' )}
```

Returns 1.

```
{array( 'apple', 'pear', 'pineapple', 'orange' )|array_search( 'carrot' )}
```

Returns -1.

```
{array( 'apple', 'pear', 'pineapple', 'orange' )|array_search( 'carrot', 'not_found'␣
↪)}
```

Returns 'not_found'.

```
{hash( 'apple', 'red',
       'pear', 'green',
       'pineapple', 'yellow',
       'orange', 'orange' )|array_search( 'yellow' )}
```

Returns 'pineapple'.

1.3 arsort

1.3.1 Summary

Sorts and returns the input array in reverse order, maintaining index associations.

1.3.2 Usage

```
input|arsort
```

1.3.3 Parameters

None.

1.3.4 Examples

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|arsort}
```

Returns array(‘d’ => 15, ‘a’ => 10, ‘b’ => 8, ‘e’ => 6, ‘c’ => 2).

1.4 asort

1.4.1 Summary

Sorts and returns the input array, maintaining index associations.

1.4.2 Usage

```
input|asort
```

1.4.3 Parameters

None.

1.4.4 Examples

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|asort}
```

Returns array(‘c’ => 2, ‘e’ => 6, ‘b’ => 8, ‘a’ => 10, ‘d’ => 15).

1.5 charset

1.5.1 Summary

Returns the current charset.

1.5.2 Usage

```
charset()
```

1.5.3 Parameters

None.

1.6 clear_object_cache

1.6.1 Summary

Clears the content objects memory cache.

1.6.2 Usage

```
clear_object_cache()
```

1.6.3 Parameters

None.

1.6.4 Examples

```
{clear_object_cache()}
```

Using this operator might be useful when processing huge number of content objects. Instead of fetching all objects at once and risking to exhaust all available memory, use a cycle to fetch a small slice of objects and to process them. Clear the memory cache using the operator before continuing with a next iteration.

1.7 cookie

1.7.1 Summary

Returns the value of the HTTP cookie identified by the parameter.

1.7.2 Usage

```
cookie( name )
```

1.7.3 Parameters

Name	Description	Required	Default
name	The name of the cookie	Yes	

1.7.4 Examples

```
{cookie( 'login' )}
```

Returns the value of the cookie named login.

1.8 current_layout

1.8.1 Summary

Returns the name of the current layout (set by, for example, the layout/set function) or false for the standard layout.

1.8.2 Usage

```
current_layout()
```

1.8.3 Parameters

None.

1.9 current_siteaccess

Returns the name of the current siteaccess.

1.9.1 Usage

```
current_siteaccess()
```

1.9.2 Parameters

None.

1.10 debug_attributes

1.10.1 Summary

Shows attribute values in the debug output. This operator is equivalent to "attribute(show)" but uses the debug output.

1.10.2 Usage

```
input|debug_attributes( [header] [, depth] )
```

1.10.3 Parameters

Name	Description	Required	Default
header	Header/title of a debug notice	No	Debug attributes operator
depth	Number of levels to show	No	2

For convenience, the order of parameters can be interchanged.

1.10.4 Examples

```
{$node|debug_attributes}
```

Shows attribute values of $node.

```
{$node|debug_attributes( 'Dump of $node' )}
```

Shows attribute values of $node as a notice titled 'Dump of $node'.

```
{$node|debug_attributes( 'Dump of $node', 1 )}
```

or:

```
{$node|debug_attributes( 1, 'Dump of $node' )}
```

Shows attributes values of $node (but not attributes of object attributes) as a notice titled 'Dump of $node'.

1.11 debug

1.11.1 Summary

Adds a notice to the debug output.

1.11.2 Usage

```
input|debug( [ header ] )
```

1.11.3 Parameters

Name	Description	Required	Default
header	Header/title of a notice	No	Debug operator

1.11.4 Examples

```
{concat( 'Node ID =', $node.node_id )|debug}
```

Adds a notice about the node ID of $node to the debug output.

```
{$variable|debug( 'Content of $variable' )}
```

Dumps $variable to the debug output as a notice titled 'Content of $variable'. See also the debug_attributes operator.

1.12 is_post_request

1.12.1 Summary

Returns true if the current request method is POST, false otherwise.

1.12.2 Usage

```
is_post_request()
```

1.12.3 Parameters

None.

1.12.4 Examples

```
This is a {if is_post_request()}POST{else}GET{/if} request.
```

Shows 'This is a POST request' if the current request is POST, 'This is a GET request' otherwise.

1.13 json_encode

1.13.1 Summary

Returns the JSON representation of input.

1.13.2 Usage

```
input|json_encode

json_encode(input)
```

1.13.3 Parameters

None.

1.13.4 Examples

```
{json_encode( 3.1415 )}
```

Returns [“3.1415”].

```
{json_encode( array( hash( 'a', 1, 'b', 2 ), 'Test', false(), 1.2345 ) )}
```

Returns [{“a”:1,”b”:2},”Test”,false,”1.234500”].

1.14 krsort

1.14.1 Summary

Sorts and returns the input array by key in reverse order.

1.14.2 Usage

```
input|krsort
```

1.14.3 Parameters

None.

1.14.4 Examples

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|krsort}
```

Returns array(‘e’ => 6, ‘d’ => 15, ‘c’ => 2, ‘b’ => 8, ‘a’ => 10).

1.15 ksort

1.15.1 Summary

Sorts and returns the input array by key.

1.15.2 Usage

```
input|ksort
```

1.15.3 Parameters

None.

1.15.4 Examples

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|ksort}
```

Returns array(‘a’ => 10, ‘b’ => 8, ‘c’ => 2, ‘d’ => 15, ‘e’ => 6).

1.16 ltrim

1.16.1 Summary

Strips whitespaces (or characters in the parameter, if given) from the beginning of the input string.

1.16.2 Usage

```
input|ltrim( [ charlist] )
```

1.16.3 Parameters

Name	Description	Required	Default
charlist	String containing characters to be removed	No	

1.16.4 Examples

```
{'   -- Hello, world! --   '|ltrim}
```

Returns ‘– Hello, world! – ‘.

```
{'   -- Hello, world! --   '|ltrim( ' -' )}
```

Returns ‘Hello, world! – ‘.

1.17 modify_view_parameter

1.17.1 Summary

Adds, removes or changes a view (user) parameter in the input path.

1.17.2 Usage

```
input|modify_view_parameter( parameter, value )
```

1.17.3 Parameters

Name	Description	Required	Default
parameter	Name of view (user) parameter to add, remove or change	Yes	
value	New value or false to remove the parameter	Yes	

1.17.4 Examples

```
{'/path/to/node/(sort)/name/(offset)/10'|modify_view_parameter( 'sort', 'modified' )}
```

Returns '/path/to/node/(sort)/modified/(offset)/10'.

```
{'/path/to/node/(sort)/name/(offset)/10'|modify_view_parameter( 'sort', false() )}
```

Returns '/path/to/node/(offset)/10'.

```
{'/path/to/node/(sort)/name/(offset)/10'|modify_view_parameter( 'sort_order', 'desc'␣
↪)}
```

Returns '/path/to/node/(sort)/name/(offset)/10/(sort_order)/desc'.

1.18 preg_match

1.18.1 Summary

Returns number of matches of a given (Perl-style) regular expression in the input string.

1.18.2 Usage

```
input|preg_match( pattern )
```

1.18.3 Parameters

Name	Description	Required	Default
pattern	Regular expression to search for	Yes	

1.18.4 Examples

```
{'The quick brown fox jumps over the lazy dog'|preg_match( '/the/i' )}
```

Returns 2.

1.19 preg_replace

1.19.1 Summary

Performs a regular expression search and replace.

1.19.2 Usage

```
input|preg_replace( search, replace )
```

1.19.3 Parameters

Name	Description	Required
search	The pattern (Perl-compatible regular expression) to search for (or array of such patterns)	Yes
replace	Replacement string (or array of such strings)	Yes

For more information see http://php.net/preg_replace.

1.19.4 Examples

```
{'Hello, John and Peter!'|preg_replace( array( '/John/', '/Peter/' ), array( 'Jane',
↪'Petra' ) )}
```

returns 'Hello, Jane and Petra!'.

```
{'John Doe'|preg_replace( '/(\w+) (\w+)/', '$2, $1' )}
```

returns 'Doe, John'.

1.20 range

1.20.1 Summary

Returns an array of integers within the given range.

1.20.2 Usage

```
range( min, max [, step] )
```

1.20.3 Parameters

Name	Description	Required	Default
min	The minimum value, inclusive	Yes	
max	The maximum value, inclusive	Yes	
step	Increment between elements	No	1

Note: min and max might be interchanged. The sign of step is not important.

1.20.4 Examples

```
{range( 1, 10 )}
```

Returns array of integers from 1 to 10 inclusive.

```
{range( 1, 12, 3 )}
```

Returns array(1, 4, 7, 10).

```
{range( 5, 2 )}
```

Returns array(5, 4, 3, 2).

```
{range( 9, 2, 2 )}
```

or:

```
{range( 9, 2, -2 )}
```

Returns array(9, 7, 5, 3).

1.21 redirect

1.21.1 Summary

Stops execution and redirects to the given URL.

1.21.2 Usage

```
redirect( url [, status] )
```

or

```
url|redirect
```

1.21.3 Parameters

Name	Description	Required	Default
url	Address to redirect to	Yes	
status	HTTP status code	No	302

1.21.4 Examples

```
{redirect( 'book/article' )}
```

Stops execution and redirects to the node represented by the (eZ Publish) path string book/article.

```
{'http://www.seeds.no'|redirect}
```

Stops execution and redirects to http://www.seeds.no.

```
{redirect( $node.parent.url, 301 )}
```

Stops execution and redirects to the parent node of $node, returning status code 301 (permanent redirection).

1.22 remove_array_element

1.22.1 Summary

Removes an array element in the input array and returns modified array.

1.22.2 Usage

```
input|remove_array_element( key )
```

1.22.3 Parameters

Name	Description	Required	Default
key	Key of the array element to remove	Yes	

1.22.4 Examples

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|remove_array_element( 'c' )}
```

Returns array(‘a’ => 10, ‘d’ => 15, ‘b’ => 8, ‘e’ => 6).

1.23 return

1.23.1 Summary

Terminates execution returning input as the response.

1.23.2 Usage

```
input|return( [ content_type ] )
```

1.23.3 Parameters

Name	Description	Required	Default
content_type	Content type of HTTP response	No	

1.23.4 Examples

```
{$variable|json_encode|return( 'application/json' )}
```

Returns value of $variable as JSON document suitable for processing by Javascript.

1.24 rsort

1.24.1 Summary

Sorts and returns the input array in reverse order.

1.24.2 Usage

```
input|rsort
```

Note that this operator assigns new keys to the output array.

1.24.3 Parameters

None.

1.24.4 Examples

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|rsort}
```

Returns array(15, 10, 8, 6, 2).

1.25 rtrim

1.25.1 Summary

Strips whitespaces (or characters in the parameter, if given) from the end of the input string.

1.25.2 Usage

```
input|rtrim( [ charlist] )
```

1.25.3 Parameters

Name	Description	Required	Default
charlist	String containing characters to be removed	No	

1.25.4 Examples

```
{'   -- Hello, world! --   '|rtrim}
```

Returns ‘ – Hello, world! –’.

```
{'   -- Hello, world! --   '|rtrim( ' -' )}
```

Returns ‘ – Hello, world!’.

1.26 serialize

1.26.1 Summary

Returns a storable representation of the input using PHP function serialize().

1.26.2 Usage

```
input|serialize
```

1.26.3 Parameters

None.

1.26.4 Examples

Using serialize and unserialize for caching value of variables:

```
{def $children=false()}

{set-block variable=$children_ser}
{cache-block expiry="0" subtree_expiry="content/view/full/2"}
{set $children=fetch( 'content', 'list', hash( 'parent_node_id', 2, 'as_object',␣
↪false() ) )}
{$children|serialize}
{/cache-block}
{/set-block}

{if $children|not}
{set $children=$children_ser|trim|unserialize}
{/if}
```

1.27 server

1.27.1 Summary

Returns content of the server variable identified by the parameter. By server variables we understand information such as headers, paths, and script locations, variables defined in CGI specification etc. For more information, see http://php.net/reserved.variables.server

1.27.2 Usage

```
server( variable_name )
```

Note: Variable name is case insensitive, the operator will capitalize it before obtaining its value.

1.27.3 Parameters

Name	Description	Required	Default
variable_name	Name of the server variable to return	Yes	

1.27.4 Examples

```
{server( 'remote_addr' )}
```

Returns the client's IP address.

```
{server( 'http_referer' )}
```

Returns from which URL the client got to the current page.

```
{server( 'http_user_agent' )}
```

Returns the identification of client's browser.

1.28 set_array_element

1.28.1 Summary

Sets an array element in the input array and returns the modified array.

1.28.2 Usage

```
input|set_array_element( key, value )
```

1.28.3 Parameters

Name	Description	Required	Default
key	Key of the array element to set	Yes	
value	Value to assign	Yes	

1.28.4 Examples

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|set_array_element( 'c', 0 )}
```

Returns array(‘a’ => 10, ‘c’ => 0, ‘d’ => 15, ‘b’ => 8, ‘e’ => 6).

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|set_array_element( 'f', 12 )}
```

Returns array(‘a’ => 10, ‘c’ => 2, ‘d’ => 15, ‘b’ => 8, ‘e’ => 6, ‘f’ => 12).

1.29 shortenw

1.29.1 Summary

Returns a shortened version of the input string (without breaking words).

Note that ‘w’ at the end of the operator name stands for word.

1.29.2 Usage

```
input|shortenw( length [, ellipsis] )
```

1.29.3 Parameters

Name	Description	Required	Default
length	Maximum length of the returned string (including ellipsis)	Yes	
ellipsis	Ellipsis	No	...

1.29.4 Examples

```
{'The quick brown fox jumps over the lazy dog'|shortenw( 16 )}
```

Returns 'The quick…'.

```
{'The quick brown fox jumps over the lazy dog'|shortenw( 16, '-' )}
```

Returns 'The quick brown-'.

```
{'Textwithnospaces'|shortenw( 10 )}
```

Returns 'Textwit…'.

1.30 shuffle

1.30.1 Summary

Randomizes the order of the elements in the input array and returns is as output.

1.30.2 Usage

```
input|shuffle
```

1.30.3 Parameters

None

1.30.4 Examples

```
{range( 1, 10 )|shuffle}
```

Creates an array of numbers from 1 to 10, inclusive, and shuffles them. Note that range is also an operator in the Swark extension.

1.31 sort

1.31.1 Summary

Sorts and returns the input array.

1.31.2 Usage

```
input|sort
```

Note that this operator assigns new keys to the output array.

1.31.3 Parameters

None.

1.31.4 Examples

```
{hash( 'a', 10, 'c', 2, 'd', 15, 'b', 8, 'e', 6 )|sort}
```

Returns array(2, 6, 8, 10, 15).

1.32 split_by_length

1.32.1 Summary

Converts the input string to an array.

1.32.2 Usage

```
input|split_by_length( [ length ] )
```

1.32.3 Parameters

Name	Description	Required	Default
length	Chunk length or lengths (if array)	No	1

1.32.4 Examples

```
{'abcdef'|split_by_length}
```

Returns array(‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’).

```
{'abcdef'|split_by_length( 2 )}
```

Returns array(‘ab’, ‘cd’, ‘ef’).

```
{'abcdef'|split_by_length( array( 1, 2, 3 ) )}
```

Returns array(‘a’, ‘bc’, ‘def’).

```
{'abcdefgh'|split_by_length( array( 1, 2 ) )}
```

Returns array(‘a’, ‘bc’, ‘d’, ‘ef’, ‘g’, ‘h’).

1.33 strpos

1.33.1 Summary

Returns position of the first occurrence of the parameter in the input string.

1.33.2 Usage

```
input|strpos( needle [, offset] )
```

1.33.3 Parameters

Name	Description	Required	Default
needle	Searched string	Yes	
offset	Allows to specify at which character in the input to start the search	No	0

1.33.4 Examples

```
{'The quick brown fox jumps over the lazy dog'|strpos( 'brown' )}
```

Returns 10.

```
{'Does the quick brown fox jumps over the lazy dog?'|strpos( 'the', 6 )}
```

Returns 36.

1.34 str_replace

1.34.1 Summary

Replaces all occurrences of the search string with the replacement string.

1.34.2 Usage

```
input|str_replace( search, replace )
```

1.34.3 Parameters

Name	Description	Required
search	The string to search for (or array of such strings)	Yes
replace	Replacement string (or array of such strings)	Yes

1.34.4 Examples

```
{'Hello, World!'|str_replace( 'World', 'Earth' )}
```

returns 'Hello, Earth!'.

1.35 strrpos

1.35.1 Summary

Returns position of the last occurrence of the parameter in the input string.

1.35.2 Usage

```
input|strrpos( needle [, offset] )
```

1.35.3 Parameters

Name	Description	Required	Default
needle	Searched string	Yes	

1.35.4 Examples

```
{'Does the quick brown fox jumps over the lazy dog?'|strrpos( 'the' )}
```

Returns 36.

1.36 substr

1.36.1 Summary

Returns part of the input string.

1.36.2 Usage

```
input|substr( start [, length] )
```

1.36.3 Parameters

Name	Description	Required	Default
start	Position in the input where to start the extraction. If negative, the position will be counted from the end of the input string.	Yes	
length	Number of characters to extract, if 0 or not specified, the rest of the string is returned. If negative, it is the number of characters that will be omitted from the end of the input string.	No	0

1.36.4 Examples

```
{'The quick brown fox jumps over the lazy dog'|substr( 4 )}
```

Returns 'quick brown fox jumps over the lazy dog'.

```
{'The quick brown fox jumps over the lazy dog'|substr( 4, 11 )}
```

Returns 'quick brown'.

```
{'The quick brown fox jumps over the lazy dog'|substr( 4, -3 )}
```

Returns 'quick brown fox jumps over the lazy'.

```
{'The quick brown fox jumps over the lazy dog'|substr( -8 )}
```

Returns 'lazy dog'.

```
{'The quick brown fox jumps over the lazy dog'|substr( -8, 4 )}
```

Returns 'lazy'.

1.37 unserialize

1.37.1 Summary

Returns the value previously stored by the serialize operator.

1.37.2 Usage

```
input|unserialize
```

1.37.3 Parameters

None.

1.37.4 Examples

See examples of the serialize operator.

1.38 uri_path_segment

1.38.1 Summary

Returns a segment of the input URL's path.

1.38.2 Usage

```
input|uri_path_segment( [ index ] )
```

1.38.3 Parameters

Name	Description	Required	Default
index	Which path segement to return, if negative, counted from the end	No	-1

1.38.4 Examples

```
{'http://www.seeds.no/nor/content/search?SearchText=publish#results'|uri_path_segment}
```

Returns 'search'.

```
{$child.url|uri_path_segment}
```

Returns last path segment of $child's url which can be used as an anchor.

1.39 user_id_by_login

1.39.1 Summary

Returns the user ID of a user identified by the login name.

1.39.2 Usage

```
user_id_by_login( login )
```

1.39.3 Parameters

Name	Description	Required	Default
login	Login name of a user	Yes	

1.39.4 Examples

```
{user_id_by_login( 'admin' )}
```

Returns 14.

1.40 variable_names

1.40.1 Summary

Shows or returns comma separated names of all available template variables. If the parameter is true or 1 or not given, the string is not returned but shown in the debug output.

1.40.2 Usage

```
variable_names( [in_debug] )
```

1.40.3 Parameters

Name	Description	Required	Default
in_debug	If true, uses debug output instead of returning the names	No	1

1.40.4 Examples

```
{variable_names()}
```

Placed in the pagelayout.tpl, it adds a debug notice '$module_result, $site, $ezinfo, $current_user, $anonymous_user_id, $access_type, $warning_list, $navigation_part, $uri_string, $requested_uri_string, $ui_context, $ui_component, $DesignKeys:used, $DesignKeys:matched'.

1.41 embed_design_file

1.41.1 Summary

Embeds a file from the design folders. Embeds a file located in any design folders on the site. This is similar to the *ezdesign* operator but will return the contents of the file instead of the path.

This can for instance be used to embed javascript code from a file. The javascript file can then be separate from the template code and can be opened like a normal javascript file in an editor. Another typical use case is to embed handlebar templates or inline CSS.

1.41.2 Usage

```
embed_design_file( <file path> )
```

1.41.3 Parameters

Name	Description	Required	Default
file path	Path to design file relative from design folder	Yes	

1.41.4 Examples

Pass the relative file path to the operator, for instance *javascript/code.js* could then be resolved to *extension/site/design/site/javscript.code.js* if the folder *extension/site/design/site* contains this file:

```
{embed_design_file('javascript/code.js')}
```

If the second parameter is used and set to true then the returned value will contain an HTML element around the file contents if the file type is known, currently only Javascript and CSS files are supported:

```
{embed_design_file('javascript/code.js', true())}
```

1.42 ezselection_content

1.42.1 Summary

Alternative for getting object attribute content from ezselection attribute. The default content always returns an array of values, which only contains id's, whereas this operator returns an array of *[<id> => <name>]*.

In the case of single select, returns the name directly. Bypass this to return id-name-array with the named parameter *with_id_as_key* to true. This parameter is irrelevant for multiselects.

1.42.2 Usage

```
field|ezselection_content([true])
```

1.42.3 Parameters

Name	Description	Required	Default
field	eZSelection field from data_map. Do note that we do not pass in the content, and instead the entire field.	Yes	
with_id_as_key	Whether to include the id of the selection as key. This is implicit for multiselect.	No	

1.42.4 Examples

```
{$data_map.my_ezselection_single_field|ezselection_content}
```

Returns name string for selected value: *'<name>'*.

```
{$data_map.my_ezselection_single_field|ezselection_content(true)}
```

Returns array of the one selected, with identifier as key: *array(<id> => <name>)*.

```
{$data_map.my_ezselection_multiple_field|ezselection_content}
```

Returns array of selected values, with identifier as key: *array(<id> => <name>, [...])*.

CHAPTER 2

Workflow event types

2.1 autopriority

Automatically sets the priority of new nodes to the maximum of siblings prirorities incremented by value of [AutoPriority]/PriorityIncrement set in swark.ini (default 10).

2.2 defertocron

Defers the workflow to the cron (background processing).

CHAPTER 3

Custom operators

Swark also makes it easier to create custom operators.

Operators are detected from the INI file *swark.ini*. Adding a new operator only requires defining a new entry in the INI file under *[Operators]*, this maps the template operator name to a PHP class that implements the operator.

For instance to expose the *phpinfo()* function we could do.

```
[Operators]
OperatorMap[phpinfo]=MyProject\PhpInfoOperator
```

Then create the PHP file and extend *SwarkOperator*, the base class will take care of all the cruft needed to define a template operator. The class must be accessible from the autoload system in PHP.

```
<?php
namespace MyProject;

use SwarkOperator;

class PhpInfoOperator extends SwarkOperator
{
    // ...
}
```

The operator class then needs a **constructor** to initialize its operator name and its parameters (*namedParameters*), and a function to **execute**.

3.1 Constructor

The constructore defines the name of the template operator, this must match the name as specified in *swark.ini*. It also defines any parameters that it supports. Each parameter is a name with an optional default value.

For instance for our *phpinfo* operator we have one parameter which is empty by default, this matches the *$what* parameter for the *phpinfo()* function.

```
<?php
class PhpInfoOperator extends SwarkOperator
{
    function __construct()
    {
        parent::__construct('phpinfo', 'what=');
    }
}
```

3.2 Execute

The execute function takes in two parameters *$operatorValue* and *$namedParameters*. *$operatorvalue* corresponds to the value that is piped to the operator, and *$namedParameters* is the value(s) supplied as parameters using the names defined in the constructor. Any values returned from *execute* will be the return value from the template operator.

The *phpinfo* implementation is then as follows.

```
<?php
class PhpInfoOperator extends SwarkOperator
{
    static function execute($operatorValue, $namedParameters)
    {
        if ($namedParameters['what']) {
            $constants = array('INFO_GENERAL' => 1, 'INFO_ALL' => -1);
            $what = $namedParameters['what'];
            if (in_array($what, $constants)) {
                phpinfo($constants[$what]);
                return;
            }
        }

        phpinfo();
    }
}
```

An using it in an eZ template:

```
{phpinfo('INFO_GENERAL')}
```