rspub-core Documentation

Release 1

h

Apr 10, 2017

Contents

genindex | modindex | search

genindex | modindex | search

CHAPTER 1

rspub.cli

Command line interface

module: rspub.cli.rscli

`Command line interface to publish resources under the ResourceSync Framework`

The module `rscli.py` offers an interface to configure, select and run the publishing of resources under the ResourceSync framework. Start *rscli* from anywhere on the system:

```
python3 rspub/cli/rscli.py
```

The internals of the command line interface resemble a three-room house. You enter the house in the `rspub` room. From there you can enter the rooms `configure` and `select`. You leave the rooms and the house by typing `exit`. In all rooms you can get help by typing `help`.

`rspub.cli.rscli.`**`str2bool`**(*v*, *none=False*)

class `rspub.cli.rscli.`**`SuperCmd`**

Bases: `cmd.Cmd`

`stop` = **False**

`__init__`()

`postcmd`(*stop*, *line*)

`do_exit`(*line*)

`help_exit`()

`do_EOF`(*line*)

EOF, Ctrl+D, Ctrl+C:

```
Exit the application.
```

static `complete_configuration`(*text*)

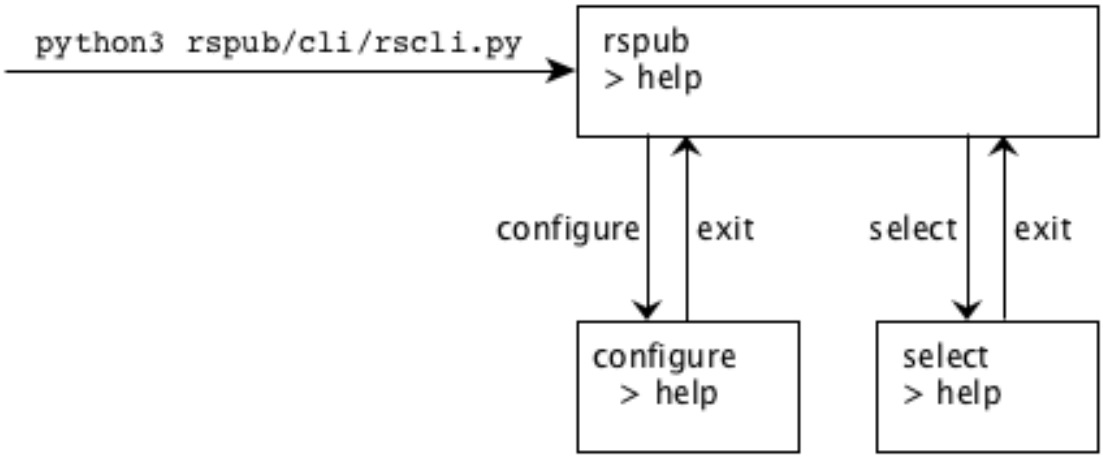

Fig. 1.1: Fig. 1. Geography of *rscli*.

do_list_configurations(*line*)
list_configurations:

```
List saved configurations
```

do_list_parameters(*line*)
list_parameters:

```
List current parameters
```

class rspub.cli.rscli.**RsPub**
Bases: *rspub.cli.rscli.SuperCmd*, *rspub.util.observe.EventObserver*

prompt = 'rspub > '

intro = '== \nCommand Line Interface for ResourceSy

__init__()

do_configure(*line*)
configure:

```
Switch to configuration mode
```

do_select(*line*)
select:

```
Switch to select mode
```

do_run(*line*)
run:

```
run rspub with the current configuration.
```

do_exit(*line*)
EOF, Ctrl+D, Ctrl+C:

```
Exit the application.
```

confirm_clear_metadata_directory(**args*, ***kwargs*)

static inform_completed_document(**args*, ***kwargs*)

static inform_execution_end(**args*, ***kwargs*)

class rspub.cli.rscli.**Configure**

Bases: *rspub.cli.rscli.SuperCmd*

prompt = 'configure > '

intro = '============================== \nConfigure Metadata Publishing \n==========================

__init__()

do_open_configuration(*name*)

open_configuration [name]:

```
Open a saved configuration
```

complete_open_configuration(*text*, *line*, *begidx*, *endidx*)

do_save_configuration(*name*)

save_configuration [name]:

```
Save the current configuration as (name)
```

do_remove_configuration(*name*)

remove_configuration [name]:

```
Remove a saved configuration
```

complete_remove_configuration(*text*, *line*, *begidx*, *endidx*)

do_reset(*line*)

reset:

```
Reset the configuration to default settings.
```

do_resource_dir(*path*)

resource_dir:

```
resource_dir          - Get the parameter
resource_dir [path]   - Set the parameter
---------------------------------------
The resource_dir acts as the root of the resources to be published.
The urls to the resources are calculated relative to the resource_dir.
```

complete_resource_dir(*text*, *line*, *begidx*, *endidx*)

do_metadata_dir(*path*)

metadata_dir:

```
metadata_dir          - Get the parameter
metadata_dir [path]   - Set the parameter
---------------------------------------
The metadata_dir is where sitemaps will be stored.
The metadata_dir is always relative to the resource_dir
```

do_description_dir(*path*)
description_dir:

```
description_dir         - Get the parameter
description_dir [path]  - Set the parameter
description_dir None    - Reset the parameter
------------------------------------------
The path to the directory of the (local copy of) the source description,
aka '.well-known/resourcesync'
```

complete_description_dir(*text*, *line*, *begidx*, *endidx*)

do_url_prefix(*url*)
url_prefix:

```
url_prefix           - Get the parameter
url_prefix [prefix]  - Set the parameter
---------------------------------------
The url_prefix is used to prefix urls to documents and resources.
```

do_has_wellknown_at_root(*value*)
has_wellknown_at_root:

```
has_wellknown_at_root             - Get the parameter
has_wellknown_at_root (yes | no)  - Set the parameter
--------------------------------------------------
The description document '.well-known/resourcesync' is at the root
of the server address.
```

do_strategy(*name*)
strategy:

```
strategy             - Get the parameter
strategy [strategy]  - Set the parameter
----------------------------------------
The strategy determines what will be done by ResourceSync upon execution.
```

complete_strategy(*text*, *line*, *begidx*, *endidx*)

do_discard_selector_file(*line*)
discard_selector_file:

```
Remove the association between this configuration and selector (if any).
An association between a configuration and a selector is set after execution
of ResourceSync with a Selector as file selector.
```

do_select_mode(*mode*)
select_mode:

```
select_mode         - Get the parameter
select_mode [mode]  - Set the parameter
---------------------------------------
Mode for selecting resources.
```

complete_select_mode(*text*, *line*, *begidx*, *endidx*)

do_plugin_dir(*path*)
plugin_dir:

```
plugin_dir          - Get the parameter
plugin_dir [path]   - Set the parameter
plugin_dir None     - Reset the parameter
----------------------------------------
The directory where plugins can be found.
```

complete_plugin_dir(*text*, *line*, *begidx*, *endidx*)

do_max_items_in_list(*value*)

max_items_in_list:

```
max_items_in_list                    - Get the parameter
max_items_in_list (int, 1 - 50000)   - Set the parameter
----------------------------------------------------
The maximum amount of records in a sitemap.
```

do_zero_fill_filename(*value*)

zero_fill_filename:

```
zero_fill_filename                - Get the parameter
zero_fill_filename (int, 1 - 10)  - Set the parameter
--------------------------------------------------
The amount of digits in a sitemap filename.
```

do_is_saving_pretty_xml(*value*)

is_saving_pretty_xml:

```
is_saving_pretty_xml              - Get the parameter
is_saving_pretty_xml (yes | no)   - Set the parameter
--------------------------------------------------
Determines appearance of sitemap xml.
```

do_is_saving_sitemaps(*value*)

is_saving_sitemaps:

```
is_saving_sitemaps              - Get the parameter
is_saving_sitemaps (yes | no)   - Set the parameter
--------------------------------------------------
Determines if sitemaps will be written to disk.
```

class rspub.cli.rscli.**Select**

Bases: *rspub.cli.rscli.SuperCmd*

prompt = 'select > '

intro = '==================================== \nSelect data for ResourceSync Publishing \n==========

__init__()

do_load_selector(*path*)

load_selector:

```
load_selector [path] - Load Selector from location [path]
---------------------------------------------------------
If the current Selector has unsaved changes, you will be
prompted to save or discard.
```

complete_load_selector(*text*, *line*, *begidx*, *endidx*)

do_save_selector(*path*)
save_selector:

```
save_selector          - Save current selector
save_selector [path]   - Save current selector as [path]
```

complete_save_selector(*text*, *line*, *begidx*, *endidx*)

do_include_path(*path*)
include_path:

```
include_path [path] - Add a file or directory to the collection of includes.
--------------------------------------------------------------------------------
The [path] can be relative or absolute.
```

complete_include_path(*text*, *line*, *begidx*, *endidx*)

do_list_includes(*line*)
list_includes:

```
List absolute filenames of the included files.
```

do_exclude_path(*path*)
exclude_path:

```
exclude_path [path] - Add a file or directory to the collection of excludes.
--------------------------------------------------------------------------------
The [path] can be relative or absolute.
```

complete_exclude_path(*text*, *line*, *begidx*, *endidx*)

do_list_excludes(*line*)
list_excludes:

```
List absolute filenames of the excluded files.
```

do_list_selected(*line*)
list_selected:

```
List absolute filenames of the selected files. The selected files are
the relative complement of excludes with respect to includes.
(list_includes \ list_excludes)
```

do_read_includes(*path*)
read_includes:

```
read_includes [path]   - Read included filenames from a file at [path]
```

complete_read_includes(*text*, *line*, *begidx*, *endidx*)

do_read_excludes(*path*)
read_excludes:

```
read_excludes [path]   - Read excluded filenames from a file at [path]
```

complete_read_excludes(*text*, *line*, *begidx*, *endidx*)

do_clear_includes(*line*)
clear_includes:

```
Clear included filenames from selector.
```

do_clear_excludes(*line*)
clear_excludes:

```
Clear excluded filenames from selector.
```

do_discard_include(*path*)
discard_include:

```
discard_include [path]  - Remove [path] from included filenames.
```

complete_discard_include(*text*, *line*, *begidx*, *endidx*)

do_discard_exclude(*path*)
discard_exclude:

```
discard_exclude [path]  - Remove [path] from excluded filenames.
```

complete_discard_exclude(*text*, *line*, *begidx*, *endidx*)

do_get_included_entries(*line*)
get_included_entries:

```
List included entries.
```

do_get_excluded_entries(*line*)
get_excluded_entries:

```
List excluded entries.
```

check_exit()

do_exit(*line*)

do_EOF(*line*)
EOF, Ctrl+D, Ctrl+C:

```
Exit the application.
```

genindex | modindex | search

CHAPTER 2

rspub.core

Configuration

module: rspub.core.config

Save and load multiple configurations

The class *Configurations* (mark the *s* at the end) enables you to save, load, remove and list multiple configurations.

Class *Configuration* (mark the absence of *s* at the end) is a singleton. It should not be used directly. In stead use *rspub.core.rs_paras.RsParameters*.

The location where configurations are stored is system-dependent:

- {user-home}\AppData\Local\Programs\rspub\config\ on Windows
- {user-home}/.config/rspub/config/ on Mac and Linux
- {user-home}/rspub/config/ fallback

See also:

RsParameters

class rspub.core.config.**Configurations**

Bases: object

Enables saving, loading, listing and removing *configurations*

All methods are static:

```
Configurations.list_configurations()
Configurations.load_configuration("collection_1")
# etc.
```

static list_configurations() → list

List available configurations

Returns list of names of previously saved configurations

static load_configuration(*name: str*)
Load the configuration with the given name

Parameters **name** – name of a previously saved configuration

Returns the restored Configuration

static save_configuration_as(*name: str*)
Save the current configuration under the given name

Any previously saved configurations with the same name will be overwritten without warning.

Parameters **name** – name under which the configuration will be saved

static remove_configuration(*name: str*)
Remove the configuration with the given name

Parameters **name** – the name of the configuration to remove

Returns **True** if the configuration was successfully removed, **False** otherwise

static current_configuration_name()
Get the name of the current configuration

Returns name of the current configuration

static rspub_config_dir()

class rspub.core.config.**Configuration**
Bases: object

Singleton persisting object for storing configuration parameters

> **Warning:** Do not use class Configuration directly. Use *RsParameters* in stead.

static reset()

config_path = '/home/docs/.config/rspub/core'

config_file = '/home/docs/.config/rspub/core/DEFAULT.cfg'

name()

persist()

core_items()

core_clear()

resource_dir(*fallback='/home/docs'*)

set_resource_dir(*resource_dir*)

metadata_dir(*fallback='metadata'*)

set_metadata_dir(*metadata_dir*)

description_dir(*fallback=None*)

set_description_dir(*description_dir*)

selector_file(*fallback=None*)

set_selector_file(*selector_file*)

simple_select_file (*fallback=None*)

set_simple_select_file (*simple_file*)

select_mode (*fallback='simple'*)

set_select_mode (*mode*)

plugin_dir (*fallback=None*)

set_plugin_dir (*plugin_dir*)

history_dir (*fallback=None*)

set_history_dir (*history_dir*)

url_prefix (*fallback='http://www.example.com'*)

set_url_prefix (*urlprefix*)

strategy (*fallback='resourcelist'*)

set_strategy (*strategy*)

max_items_in_list (*fallback=50000*)

set_max_items_in_list (*max_items*)

zero_fill_filename (*fallback=4*)

set_zero_fill_filename (*zfill*)

is_saving_pretty_xml (*fallback=True*)

set_is_saving_pretty_xml (*p_xml*)

is_saving_sitemaps (*fallback=True*)

set_is_saving_sitemaps (*is_saving*)

has_wellknown_at_root (*fallback=True*)

set_has_wellknown_at_root (*at_root*)

last_excution ()

set_last_execution (*date_string*)

last_strategy ()

set_last_strategy (*strategy*)

last_sitemaps (*fallback=[]*)

set_last_sitemaps (*sitemaplist*)

exp_scp_server (*fallback='example.com'*)

set_exp_scp_server (*exp_scp_server*)

exp_scp_port (*fallback=22*)

set_exp_scp_port (*exp_scp_port*)

exp_scp_user (*fallback='username'*)

set_exp_scp_user (*exp_scp_user*)

exp_scp_document_root (*fallback='/var/www/html/'*)

set_exp_scp_document_root (*exp_scp_document_root*)

zip_filename (*fallback='/home/docs/resourcesync.zip'*)

set_zip_filename (*zip_filename*)

imp_scp_server (*fallback='example.com'*)

set_imp_scp_server (*imp_scp_server*)

imp_scp_port (*fallback=22*)

set_imp_scp_port (*imp_scp_port*)

imp_scp_user (*fallback='username'*)

set_imp_scp_user (*imp_scp_user*)

imp_scp_remote_path (*fallback='~'*)

set_imp_scp_remote_path (*imp_scp_remote_path*)

imp_scp_local_path (*fallback='/home/docs'*)

set_imp_scp_local_path (*imp_scp_local_path*)

parser = <configparser.ConfigParser object>

Parameters

module: rspub.core.rs_paras

Parameters for ResourceSync publishing

The class *RsParameters* validates parameters for ResourceSync publishing that are used throughout the application. RsParameters can be persisted as configuration.

Multiple sets of parameters can be saved and reused as named configurations. This enables configuring rspub-core to publish metadata on different sets of resources. Each configuration can have its own selection mechanism, metadata directory, strategy etc. Each set of resources can than be published in its own capability list.

The class *RsParameters* in this module and the class *rspub.core.config.Configurations* are important assets in this endeavour. RsParameters can be associated with a saved *rspub.core.selector.Selector*.

class rspub.core.rs_paras.**RsParameters** (*config_name=None*, *resource_dir=None*, *metadata_dir=None*, *description_dir=None*, *url_prefix=None*, *strategy=None*, *selector_file=None*, *simple_select_file=None*, *select_mode=None*, *plugin_dir=None*, *history_dir=None*, *max_items_in_list=None*, *zero_fill_filename=None*, *is_saving_pretty_xml=None*, *is_saving_sitemaps=None*, *has_wellknown_at_root=None*, *exp_scp_server=None*, *exp_scp_port=None*, *exp_scp_user=None*, *exp_scp_document_root=None*, *zip_filename=None*, *imp_scp_server=None*, *imp_scp_port=None*, *imp_scp_user=None*, *imp_scp_remote_path=None*, *imp_scp_local_path=None*, ***kwargs*)

Bases: object

Class capturing the core parameters for ResourceSync publishing

Parameters can be set in the *__init__()* method of this class and as properties. Each parameter gets a screening on validity and a *ValueError* will be raised if it is not valid. Parameters can be saved collectively as a configuration. Multiple named configurations can be stored by using the method *save_configuration_as()*. Named configurations can be restored by giving the *config_name* at initialisation:

```
# paras is an instance of RsParameters with configuration adequately set for␣
↪collection 1
# it is saved as 'collection_1_config':
paras.save_configuration_as("collection_1_config")

# ...
# Later on it is restored...
paras = RsParameters(config_name="collection_1_config")
```

Note that the class `rspub.core.Configurations` has a method for listing saved configurations by name.

RsParameters can be cloned:

```
# paras1 is an instance of RsParameters
paras2 = RsParameters(**paras1.__dict__)
paras1 == paras2    # False
paras1.__dict__ == paras2.__dict__  # True
```

Besides parameters the RsParameters class also has methods for derived properties.

See also:

rspub.core.config

__init__(*config_name=None*, *resource_dir=None*, *metadata_dir=None*, *description_dir=None*, *url_prefix=None*, *strategy=None*, *selector_file=None*, *simple_select_file=None*, *select_mode=None*, *plugin_dir=None*, *history_dir=None*, *max_items_in_list=None*, *zero_fill_filename=None*, *is_saving_pretty_xml=None*, *is_saving_sitemaps=None*, *has_wellknown_at_root=None*, *exp_scp_server=None*, *exp_scp_port=None*, *exp_scp_user=None*, *exp_scp_document_root=None*, *zip_filename=None*, *imp_scp_server=None*, *imp_scp_port=None*, *imp_scp_user=None*, *imp_scp_remote_path=None*, *imp_scp_local_path=None*, ***kwargs*)

Construct an instance of *RsParameters*

All `parameters` will get their value from

1. the _named argument in ***kwargs*. (this is for cloning instances of RsParameters). If not available:
2. the named argument. If not available:
3. the parameter as saved in the current configuration. If not available:
4. the default configuration value.

Parameters

- **config_name** (*str*) – the name of the configuration to read. If given, sets the current configuration.
- **resource_dir** (*str*) – parameter *resource_dir()*
- **metadata_dir** (*str*) – parameter *metadata_dir()*
- **description_dir** (*str*) – parameter *description_dir()*
- **url_prefix** (*str*) – parameter *url_prefix()*
- **int, str] strategy** (*Union[Strategy,)* – parameter *strategy()*

- **selector_file** (*str*) – parameter *selector_file()*
- **simple_select_file** (*str*) – parameter *simple_select_file()*
- **select_mode** (SelectMode) – parameter *select_mode()*
- **plugin_dir** (*str*) – parameter *plugin_dir()*
- **history_dir** (*str*) – parameter *history_dir()*
- **max_items_in_list** (*int*) – parameter *max_items_in_list()*
- **zero_fill_filename** (*int*) – parameter *zero_fill_filename()*
- **is_saving_pretty_xml** (*bool*) – parameter *is_saving_pretty_xml()*
- **is_saving_sitemaps** (*bool*) – parameter *is_saving_sitemaps()*
- **has_wellknown_at_root** (*bool*) – parameter *has_wellknown_at_root()*
- **exp_scp_server** (*str*) – parameter *exp_scp_server()*
- **exp_scp_port** (*int*) – parameter *exp_scp_port()*
- **exp_scp_user** (*str*) – parameter *exp_scp_user()*
- **exp_scp_document_root** (*str*) – parameter *exp_scp_document_root()*
- **zip_filename** (*str*) – parameter *zip_filename()*
- **imp_scp_server** (*str*) – parameter *imp_scp_server()*
- **imp_scp_port** (*int*) – parameter *imp_scp_port()*
- **imp_scp_user** (*str*) – parameter *imp_scp_user()*
- **imp_scp_remote_path** (*str*) – parameter *imp_scp_remote_path()*
- **imp_scp_local_path** (*str*) – parameter *imp_scp_local_path()*
- **kwargs** – named arguments, same as parameters, but preceded by _

Raises ValueError if a parameter is not valid or if the configuration with the given *config_name* is not found

resource_dir

parameter The local root directory for ResourceSync publishing (str)

The given value should point to an existing directory. A relative path will be made absolute, calculated from the current working directory (*os.getcwd()*).

The resource_dir acts as the root of the resources to be published. The urls to the resources are calculated relative to the resource_dir. Example:

```
resourece_dir:  /abs/path/to/resource_dir
resource:       /abs/path/to/resource_dir/sub/path/to/resource
url:                        url_prefix + /sub/path/to/resource
```

default: user home directory

See also: *url_prefix()*

metadata_dir

parameter The directory for ResourceSync documents (str)

The metadata_dir is the directory where sitemap documents will be saved. Names and relative path names are allowed. An absolute path will raise a `ValueError`.

The metadata directory will be calculated relative to the *resource_dir()*.

If the metadata directory does not exist it will be created during execution of a synchronization.

`default:` 'metadata'

See also: *abs_metadata_dir()*

description_dir
`parameter` `Directory where a version of the description document is kept` (str)

The description document, also known as *.well-known/resourcesync*, is keeping links to the capability list(s) at the site. A local copy of the description document (or the real description document if synchronization takes place at the server) will be updated with newly created capability lists. The *description_dir* should point to a directory where the `.well-known/resourcesync` document can be found.

If *description_dir* is **None** the *abs_metadata_dir()* will be taken as *description_dir*.

If the document *{description_dir}*`/.well-known/resourcesync` does not exist it will be created.

`default:` **None**

See also: *abs_description_path()*

url_prefix
`parameter` `The URL-prefix for ResourceSync publishing` (str)

The url_prefix substitutes *resource_dir()* when calculating urls to resources. The *url_prefix* should be the host name of the server or host name + path that points to the root directory of the resources. *url_prefix* + *relative/path/to/resource* should yield a valid url.

Example. Paths to resources are relative to the server host:

```
path to resource:           {resource_dir}/path/to/resource
url_prefix:         http://www.example.com
url to resource:    http://www.example.com/path/to/resource
```

Example. Paths to resources are relative to some directory on the server:

```
path to resource:                       {resource_dir}/path/to/resource
url_prefix:         http://www.example.com/my/resources
url to resource:    http://www.example.com/my/resources/path/to/resource
```

`default:` 'http://www.example.com'

See also: *resource_dir()*

strategy
`parameter` `Strategy for ResourceSync publishing` (str | int | *Strategy*)

The *strategy* determines what will be done by `ResourceSync` upon execution. At the moment valid values for *strategy* are:

- `0` *resourcelist* - new resourcelist: create new resourcelist(s)
- `1` *new_changelist* - new changelist: create a new changelist on every execution
- `2` *inc_changelist* - incremental changelist: add changes to an existing changelist

If strategies new resourcelist or incremental changelist are chosen and there is no previous resourcelist found in the metadata directory the strategy *resourcelist* will be executed.

default: *rspub.core.rs_enum.Strategy.resourcelist*

selector_file
parameter Location of file to construct a *Selector* (str)

A *rspub.core.selector.Selector* can be used as input for the execute methods. The *selector_file* specifies the location of the selector file.

default: **None**

simple_select_file

select_mode

history_dir
parameter Directory for storing reports on executed synchronisations (str)

Currently not in use.

plugin_dir
parameter Directory where plugins can be found (str)

The given value should point to an existing directory. A relative path will be made absolute, calculated from the current working directory (*os.getcwd()*).

At the moment plugins for *ResourceGateBuilder* can be provided.

default: **None**

See also: *rspub.util.gates*

max_items_in_list
parameter The maximum amount of records in a sitemap (int, 1 - 50000)

The 'community defined' maximum amount of records in a sitemap document is 50000. If on execution the maximum amount is reached, new sitemaps of the same category will be created with the remaining records.

default: 50000

zero_fill_filename
parameter The amount of digits in a sitemap filename (int, 1 - 10)

Filenames of resourcelist, changelist etc. are numbered and are post-fixed with this number filled with zero's up to *zero_fill_filename*. Examples of filenames with *zero_fill_filename* set at 4:

```
changelist_0002.xml
changelist_0003.xml
```

default: 4

is_saving_pretty_xml
parameter Determines appearance of sitemap xml (bool)

If no humans need to read or inspect sitemaps there is no need for linebreaks etc.

default: **True**, with linebreaks

is_saving_sitemaps
parameter Determines if sitemaps will be written to disk (bool)

An execution can be a dry-run. With this parameter set to **False** sitemaps will be generated, but not written to disk.

default: **True**, write sitemaps to disk

has_wellknown_at_root
parameter Where is the description document *.well-known/resourcesync* on the server (bool)

The description document is the main entry point for third parties trying to discover resources at a source. Capability lists point toward this document in their *rel:up* attribute. If for some reason the *.well-known/resourcesync* cannot be at the root of the server the *rel:up* link in capability lists will be made to be pointing at *.well-known/resourcesync* relative to *abs_metadata_dir()*.

default: **True**, the *.well-known/resourcesync* is at the root of the server

exp_scp_server

exp_scp_port

exp_scp_user

exp_scp_document_root
parameter The directory from which the web server will serve files (str)

Example. Paths to resources are relative to the server host:

```
url_prefix:         http://www.example.com
url to resource:    http://www.example.com/path/to/resource
scp_document_root:          /var/www/html/
scp_document_path:
path on server:             /var/www/html/path/to/resource
```

Example. Paths to resources are relative to some directory on the server:

```
url_prefix:         http://www.example.com/my/resources
url to resource:    http://www.example.com/my/resources/path/to/resource
scp_document_root:          /var/www/html/
scp_document_path:                        my/resources
path on server:             /var/www/html/my/resources/path/to/resource
```

default: '/var/www/html/'

zip_filename

imp_scp_server

imp_scp_port

imp_scp_user

imp_scp_remote_path
parameter The directory at the remote server from which to import files (str)

default: '~'

imp_scp_local_path

save_configuration(*on_disk=True*)
function Save current configuration

Save the current values of parameters to configuration. If *on_disk* is **True** (the default) persist the configuration to disk under the current configuration name.

Parameters **on_disk** – **True** if configuration should be saved to disk, **False** otherwise

See also: *current_configuration_name()*

save_configuration_as(*name: str*)
function Save current configuration under *name*

Save the current configuration under the given *name*. If a configuration under the given *name* already exists it will be overwritten without warning.

Parameters **name** (*str*) – the name under which the configuration will be saved

See also: *load_configuration()*

reset()

abs_metadata_dir() → str
derived The absolute path to metadata directory

Returns absolute path to metadata directory

abs_metadata_path(*filename*)
derived The absolute path to file in the metadata directory

Parameters **filename** (*str*) – the filename to position relative to the *abs_metadata_dir()*

Returns absolute path to file in the metadata directory

abs_description_path()
derived The absolute path to (the local copy of) the file *.well-known/resourcesync*

Returns absolute path to (the local copy of) the file .well-known/resourcesync

server_root()
derived The server root (of the web server) as derived from *url_prefix*

Returns server root

server_path()
derived The server path as derived from *url_prefix*

Returns server path

description_url()
derived The current description url

The current description url either points to {server root}/.well-known/resourcesync or to a file in the metadata directory.

Returns current description url

See also: *has_wellknown_at_root()*

capabilitylist_url() → str
derived The current capabilitylist url

The current capabilitylist url points to 'capabilitylist.xml' in the metadata directory.

Returns current capabilitylist url

uri_from_path(*path*)
derived Calculate the url of a path relative to *resource_dir*

Parameters **path** (*str*) – the path to calculate the url from

Returns the url of the path relative to resource_dir

abs_history_dir()
derived The absolute path to directory for reports on synchronizations

Currently not in use.

Returns absolute path to directory for reports

static configuration_name()
function Current configuration name

Returns current configuration name

example_filename(*ordinal*)

describe(*as_string=False*, *fill=23*)
function List parameters and derived values

List parameters, values and derived values as a list of tuples. Each tuple contains:

n	field	contents
0	bool	**True** for parameter, **False** for derived value
1	name	The name of the parameter or derived value
2	value	The value of the parameter or derived value
3..	...	Anything else

Parameters

- **as_string** – return contents as a printable string
- **fill** – if as_string: fill column 'name' with *fill* spaces

Returns list[list] or str

Selector

module: rspub.core.selector

class rspub.core.selector.**SelectorEvent**
Bases: enum.Enum

An enumeration.

file_does_not_exist = 0

not_a_regular_file = 1

file_excluded = 2

next_file = 10

class rspub.core.selector.**Selector**(*location=None*)
Bases: *rspub.util.observe.Observable*

__init__(*location=None*)

static filter_base_paths(*abs_paths*)

static is_base_path(*x*, *other_paths*)

include(**filenames*)

exclude(**filenames*)

discard_include(**filenames*)

discard_exclude(**filenames*)

clear_includes()

clear_excludes()

list_includes()

list_excludes()

relativize_includes(*root_path*)

relativize_excludes(*root_path*)

get_included_entries()

get_excluded_entries()

is_empty()

read_includes(*filename*)

read_excludes(*filename*)

write_includes(*filename*)

write_excludes(*filename*)

write(*filename=None*)

read(*filename*)

abs_location()

ResourceSync

module: rspub.core.rs

```
Publish resources under the ResourceSync Framework
```

The class *ResourceSync* is the main entrance to the rspub-core library. It is in essence a one-method class, its main method: *execute()*. This method takes as argument `filenames`: an iterable of files and/or directories to process. (List and i.e. *Selector* are iterables.) Upon execution *ResourceSync* will call the correct *Executor* that will walk all the files and directories named in `filenames` and that takes care of creating the right type of sitemap: resourcelist, changelist etc. and complete the corresponding sitemaps as capabilitylist and description.

Before you call *execute()* on *ResourceSync* it may be advisable to set the proper parameters for your synchronization. *ResourceSync* is a subclass of *RsParameters* and the description of `parameters` in that class is a good starting point to learn about the type, meaning and function of these parameters. Here we will highlight some and discuss aspects of these parameters.

Selecting resources

The algorithm for selecting resources can be shaped by you, the user of this library. If the default algorithm suites you - so much for the better - then you don't have to do anything and you can safely skip this paragraph.

The default algorithm is implemented by the *GateBuilder* class *ResourceGateBuilder*. This default class builds a *gate()* that allows any file that is encountered in the list of files and directories of the `filenames` argument. It will exclude however any file that is not in *resource_dir()* or any of its subdirectories, hidden files and

files from the directories *metadata_dir()*, *description_dir()* and *plugin_dir()* in case any of these directories are situated on the search-paths described in `filenames`.

You can implement your own resource *gate()* by supplying a class named *ResourceGateBuilder* in a directory you specify under the *plugin_dir()* `parameter`. Your *ResourceGateBuilder* should subclass *ResourceGateBuilder* or at least implement the methods *build_includes()* and *build_excludes()*. A detailed description of how to create your own *ResourceGateBuilder* can be found in *rspub.pluggable.gate*.

By shaping your own selection algorithm you could for instance say "include all the files from directory *x* but exclude the subdirectory *y* and from directory *z* choose only those files whose filenames start with 'abc' and from directory *z/b* choose only xml-files where the x-path expression *//such/and/so* yields 'foo' or 'bar'." Anything goes, as long as you can express it as a predicate, that is, say 'yes' or 'no' to a resource, given the filename of the resource.

See also:

rspub.util.gates, *rspub.pluggable.gate*

Strategies and executors

The *Strategy* tells *ResourceSync* in what way you want your resources processed. Or better: *ResourceSync* will choose the *Executor* that fits your chosen strategy. Do you want new resourcelists every time you call *ResourceSync.execute()*, do you want new changelists or perhaps an incremental changelist. There are slots for other strategies in rspub-core, such as resourcedump and changedump, but these strategies are not yet implemented.

If new changelist or incremental changelist is your strategy and there is no resourcelist.xml yet in your *metadata_dir()* then *ResourceSync* will create a resourcelist.xml the first time you call *execute()*.

The *Strategy* `resourcelist` does not require much system resources. Resources will be processed one after the other and sitemap documents are written to disk once they are processed and these sitemaps will at most take 50000 records. The strategies `new_changelist` and `inc_changelist` will compare previous and present state of all your selected resources. In order to do so they collect metadata from all the present resources in your selection and compare it to the previous state as recorded in resourcelists and subsequent changelists. This will be perfectly OK in most situations, however if the number of resources is very large this comparison might be undoable. Anyway, large amounts of resources will probably be managed by some kind of repository system that enables to query for the requested data. It is perfectly alright to write your own *Executor* that handles the synchronisation of resources in your repository system and you are invited to share these executors. A suitable plugin mechanism to accommodate such extraterrestrial executors could be accomplished in a next version of rspub-core.

See also:

rspub.core.rs_paras.RsParameters.strategy(), *rspub.core.rs_enum.Strategy*, *rspub.core.executors*

Multiple collections

ResourceSync is a subclass of *RsParameters* and so the parameters set on *ResourceSync* can be saved and reinstituted later on. *Configurations* has methods for listing and removing previously saved configurations. Multiple collections of resources could be synchronized, each collection with its own configuration. Synchronizing the collection 'spam' could go along these lines:

```
# get a list of previously saved configurations
[print(x) for x in Configurations.list_configurations()]
# rspub_core
# spam_config
# eggs_config

# prepare for synchronization of collection 'all about spam'
```

```
resourcesync = ResourceSync(config_name="spam_config")
# spam resources are in two directories
filenames = ["resources/green_spam", "resources/blue_spam"]
# do the synchronization
resourcesync.execute(filenames)
```

See also:

rspub.core.rs_paras.RsParameters, *rspub.core.config.Configurations*, *save_configuration_as()*

Observe execution

ResourceSync is a subclass of *Observable*. The executor to which the execution is delegated inherits all observers registered with *ResourceSync*. *ResourceSync* it self does not fire events.

See also:

rspub.util.observe, *rspub.core.executors.ExecutorEvent*

class rspub.core.rs.**ResourceSync**(***kwargs*)

Bases: *rspub.util.observe.Observable*, *rspub.core.rs_paras.RsParameters*

Main class for ResourceSync publishing

__init__(***kwargs*)

Initialization

Parameters

- **config_name** (*str*) – the name of the configuration to read. If given, sets the current configuration.
- **kwargs** – see *rspub.core.rs_paras.RsParameters.__init__()*

See also:

rspub.core.rs_paras

execute(*filenames: <built-in function iter> = None*, *start_new=False*)

Publish ResourceSync documents under conditions of current *parameters*

Call appropriate executor and publish sitemap documents on the resources found in *filenames*.

If no file/files 'resourcelist_*.xml' are found in metadata directory will always dispatch to strategy (new) resourcelist.

If parameter *is_saving_sitemaps()* is False will do a dry run: no existing sitemaps will be changed and no new sitemaps will be written to disk.

Parameters

- **filenames** – filenames and/or directories to scan
- **start_new** – erase metadata directory and create new resourcelists

class rspub.core.rs.**ExecutionHistory**(*history_dir*)

Bases: *rspub.util.observe.EventObserver*

Execution report creator

Currently not in use.

__init__(*history_dir*)

pass_inform(**args*, ***kwargs*)

inform_execution_start(**args*, ***kwargs*)

Executors

module: rspub.core.executors

Events and base classes for execution

class rspub.core.executors.**ExecutorEvent**

Bases: enum.Enum

Events fired by *Executors*

There are information events (inform) and confirmation events (confirm). If an *Observer* overrides the method *confirm()* and returns False on a confirm event, an *ObserverInterruptException* is raised.

All events are broadcast in the format:

```
[inform][confirm](source, event, **kwargs)
```

where source is the calling instance, event is the relevant event and **kwargs hold relevant information about the event.

rejected_file = 1
1 inform File rejected by resource gate

start_file_search = 2
2 inform File search was started

created_resource = 3
3 inform The metadata for a resource was created

completed_document = 10
10 inform A sitemap document was completed

found_changes = 20
20 inform Resources that changed were found

execution_start = 30
30 inform Execution of resource synchronization started

execution_end = 31
31 inform Execution of resource synchronization did end

clear_metadata_directory = 100
100 confirm Files in metadata directory will be erased

class rspub.core.executors.**SitemapData**(*resource_count=0*, *ordinal=0*, *uri=None*, *path=None*, *capability_name=None*, *document_saved=False*)

Bases: object

Holds metadata about sitemaps

__init__(*resource_count=0*, *ordinal=0*, *uri=None*, *path=None*, *capability_name=None*, *document_saved=False*)
Initialization

Parameters

- **resource_count** (*int*) – the amount of records in the sitemap
- **ordinal** (*int*) – the ordinal number as reflected in the sitemap filename and url
- **uri** (*str*) – the url of the sitemap
- **path** (*str*) – the local path of the sitemap
- **capability_name** (*str*) – the capability of the sitemap
- **document_saved** (*bool*) – True if the sitemap was saved to disk, False otherwise

class rspub.core.executors.**Executor**(*rs_parameters: rspub.core.rs_paras.RsParameters = None*)
Bases: *rspub.util.observe.Observable*

Abstract base class for ResourceSync execution

There are 6 build steps that concrete subclasses may override (or 7 if they want to completely take over the execution). Two steps are mandatory for subclasses to implement: *generate_rs_documents()* and *create_index()*. Steps *create_capabilitylist()* and *update_resource_sync()* are not abstract - they can safely be done by this *Executor*.

__init__(*rs_parameters: rspub.core.rs_paras.RsParameters = None*)
Initialization

If no *RsParameters* were given will construct new *RsParameters* from configuration found under *current_configuration_name()*.

Parameters **rs_parameters** – *RsParameters* for execution

resource_gate()
Construct or return the resource gate

Returns resource gate

execute(*filenames: <built-in function iter>*)
build step 0 Publish ResourceSync documents

Publish ResourceSync documents under conditions of current *RsParameters*.

Parameters **filenames** – iter of filenames and/or directories to scan

Returns list of *SitemapData* of generated sitemaps

prepare_metadata_dir()
build step 1 Does nothing

Subclasses that want to prepare metadata directory before generating new documents may override.

generate_rs_documents(*filenames: <built-in function iter>*) → [<class 'rspub.core.executors.SitemapData'>]
build step 2 Raises *NotImplementedError*

Subclasses must walk resources found in filenames and, if appropriate, generate sitemaps and produce sitemap data.

Parameters **filenames** – list of filenames and/or directories to scan

Returns list of *SitemapData* of generated sitemaps

post_process_documents(*sitemap_data_iter: <built-in function iter>*)
build step 3 Does nothing

Subclasses that want to post proces the documents in metadata directory may override.

Parameters **sitemap_data_iter** – iter over *SitemapData* of sitemaps generated in build step 2

create_index(*sitemap_data_iter: <built-in function iter>*)
build step 4 Raises *NotImplementedError*

Subclasses must create sitemap indexes if appropriate.

Parameters **sitemap_data_iter** – iter over *SitemapData* of sitemaps generated in build step 2

create_capabilitylist() → rspub.core.executors.SitemapData
build step 5 Create a new capabilitylist over sitemaps found in metadata directory

Returns *SitemapData* over the newly created capabilitylist

update_resource_sync(*capabilitylist_data*)
build step 6 Update description with newly created capabilitylist

Parameters **capabilitylist_data** – *SitemapData* over the newly created capabilitylist

Returns *SitemapData* over updated description

clear_metadata_dir()

resource_generator() → <built-in function iter>

walk_directories(**directories*) → [<class 'str'>]

find_ordinal(*capability*)

format_ordinal(*ordinal*)

finish_sitemap(*ordinal*, *sitemap*, *doc_start=None*, *doc_end=None*) → rspub.core.executors.SitemapData

current_rel_up_for(*sitemap*)

update_rel_index(*index_url*, *path*, *sitemap_instance*)

save_sitemap(*sitemap*, *path*)

read_sitemap(*path*, *sitemap_instance*)

Create resourcelists

module: rspub.core.exe_resourcelist

Executor creating resourcelists

class rspub.core.exe_resourcelist.**ResourceListExecutor**(*rs_parameters: rspub.core.rs_paras.RsParameters = None*)

Bases: *rspub.core.executors.Executor*

Executes the new resourcelist strategy

A ResourceListExecutor clears the metadata directory and creates new resourcelist(s) every time the executor runs (and is_saving_sitemaps).

prepare_metadata_dir()

generate_rs_documents(*filenames: <built-in function iter>*) → [<class 'rspub.core.executors.SitemapData'>]

create_index(*sitemap_data_iter: <built-in function iter>*)

resourcelist_generator(*filenames: <built-in function iter>*) → <built-in function iter>

Create changelists

module: rspub.core.exe_changelist

Executors creating changelists

Concrete classes:

- *NewChangeListExecutor*
- *IncrementalChangeListExecutor*

class rspub.core.exe_changelist.**ChangeListExecutor**(*rs_parameters: rspub.core.rs_paras.RsParameters = None*)

Bases: *rspub.core.executors.Executor*

Abstract class for creating changelists

generate_rs_documents(*filenames: <built-in function iter>*) → [<class 'rspub.core.executors.SitemapData'>]

__init__(*rs_parameters: rspub.core.rs_paras.RsParameters = None*)

create_index(*sitemap_data_iter: <built-in function iter>*) → rspub.core.executors.SitemapData

update_previous_state()

changelist_generator(*filenames: <built-in function iter>*) → <built-in function iter>

class rspub.core.exe_changelist.**NewChangeListExecutor**(*rs_parameters: rspub.core.rs_paras.RsParameters = None*)

Bases: *rspub.core.exe_changelist.ChangeListExecutor*

Implements the new changelist strategy

A *NewChangeListExecutor* creates new changelists every time the executor runs (and is_saving_sitemaps). If there are previous changelists that are not closed (md:until is not set) this executor will close those previous changelists by setting their md:until value to now (start_of_processing)

generate_rs_documents(*filenames: <built-in function iter>*)

post_process_documents(*sitemap_data_iter: <built-in function iter>*)

class rspub.core.exe_changelist.**IncrementalChangeListExecutor**(*rs_parameters: rspub.core.rs_paras.RsParameters = None*)

Bases: *rspub.core.exe_changelist.ChangeListExecutor*

Implements the incremental changelist strategy

An *IncrementalChangeListExecutor* adds changes to an already existing changelist every time the executor runs (and is_saving_sitemaps).

generate_rs_documents(*filenames: <built-in function iter>*)

Transport

module: rspub.core.transport

Transport resources and sitemaps to the web server

class rspub.core.transport.**TransportEvent**
Bases: enum.Enum

Events fired by *Transport*

All events are broadcast in the format:

```
[inform][confirm](source, event, **kwargs)
```

where source is the calling instance, event is the relevant event and **kwargs hold relevant information about the event.

copy_resource = 1
1 inform A resource was copied to a temporary location

copy_sitemap = 2
2 inform A sitemap was copied to a temporary location

copy_file = 3
3 confirm Copy file confirm message with interrupt

transfer_file = 4
4 confirm Transfer file confirm message with interrupt

resource_not_found = 10
10 inform A resource was not found

start_copy_to_temp = 15
15 inform Start copy resources and sitemaps to temporary directory

zip_resources = 20
20 inform Start packaging resources and sitemaps

scp_resources = 21
21 inform Start transfer of files with scp

ssh_client_creation = 22
22 inform Trying to create ssh client

scp_exception = 23
23 inform Encountered exception while transferring files with scp

scp_progress = 24
24 inform Progress as defined by SCPClient

scp_transfer_complete = 25
25 inform Transfer of one file complete

transport_start = 30
30 inform Transport started

transport_end = 31
31 inform Transport ended

class rspub.core.transport.**ResourceAuditorEvent**
Bases: enum.Enum

Events fired by *Transport*

All events are broadcast in the format:

```
[inform](source, event, **kwargs)
```

where `source` is the calling instance, `event` is the relevant event and `**kwargs` hold relevant information about the event.

site_map_not_found = 11
11 inform‘‘ A sitemap was not found

class rspub.core.transport.**ResourceAuditor**(*paras*)
Bases: *rspub.util.observe.Observable*

__init__(*paras*)

all_resources()

all_resources_generator()

last_resources_generator()

extract_paths(*uri*)

get_generator(*all_resources*)

class rspub.core.transport.**Transport**(*paras*)
Bases: *rspub.core.transport.ResourceAuditor*

__init__(*paras*)

handle_resources(*function*, *all_resources=False*, *include_description=True*)

zip_resources(*all_resources=False*)

scp_resources(*all_resources=False*, *password='secret'*)

create_ssh_client(*password*)

scp_put(*files*, *remote_path*)

progress(*filename*, *size*, *sent*)

Enumerations

module: rspub.core.rs_enum

class rspub.core.rs_enum.**Strategy**
Bases: enum.Enum

Strategy for ResourceSync Publishing

resourcelist = 0
0 New resourcelist *strategy*

Create new resourcelist(s) every run.

new_changelist = 1
1 New changelist *strategy*

Create a new changelist every run. If no resourcelist was found in the metadata directory switch to new resourcelist strategy.

inc_changelist = 2
2 Incremental changelist *strategy*

Add changes to an existing changelist. If no changelist exists, create a new one. If no resourcelist was found in the metadata directory switch to new resourcelist strategy.

static names()
Get Strategy names

Returns List<str> of names

static sanitize(*name*)
Verify a *Strategy* name

Parameters **name** (*str*) – string to test

Returns name if it is the name of a strategy

Raises ValueError if the given name is not the name of a strategy

static strategy_for(*value*)
Get a Strategy for the given value

Parameters **value** – may be *Strategy*, str or int

Returns *Strategy*

Raises ValueError if the given value could not be converted to a *Strategy*

describe()

class rspub.core.rs_enum.**Capability**
Bases: enum.Enum

Capabilities as defined in the ResourceSync Framework

resourcelist = 0
0 resourcelist

changelist = 1
1 changelist

resourcedump = 2
2 resourcedump

changedump = 3
3 changedump

resourcedump_manifest = 4
4 resourcedump_manifest

changedump_manifest = 5
5 changedump_manifest

capabilitylist = 6
6 capabilitylist

description = 7
7 description

class rspub.core.rs_enum.**SelectMode**
Bases: enum.Enum

Mode of selection

simple = 0

selector = 1

static names()

Get SelectMode names

Returns List<str> of names

static select_mode_for(*mode*)

genindex | modindex | search

CHAPTER 3

rspub.pluggable

Resource gate builder

module: rspub.pluggable.gate

```
Pluggable resource gate and builder
```

Build your own

The selection mechanism for resources is implemented as a *gate()* that uses predicates for including and excluding resources based on their filename. The *ResourceGateBuilder* hook allows you to shape this resource gate and adapt it completely to your needs. You can build your own ResourceGateBuilder by creating a class that subclasses *rspub.pluggable.gate.ResourceGateBuilder* or - to avoid dependencies in your code - that implements the two methods *build_includes()* and *build_excludes()*. In any case your gate builder should be named ResourceGateBuilder, because by this name your plugin will be recognized by rspub-core.

Register a ResourceGateBuilder

Your ResourceGateBuilder should be placed in a directory that is registered as *plugin_dir()* at *ResourceSync*. (There may be multiple ResourceGateBuilders in your plugin directory but this could unnecessarily complicate the building process.)

Build predicates

Predicates you supply in the lists of including and excluding predicates should be one-argument predicates that take the filename of a resource as input. The logic in your predicates could take advantage of the logical functions offered by *rspub.util.gates* and file selection filters offered in *rspub.util.resourcefilter*.

Example: Construct a predicate for directory names that end with 'abc':

```
import rspub.util.resourcefilter as rf
dir_ends_with_abc = rf.directory_pattern_predicate("abc$")

assert dir_ends_with_abc("/foo/bar/folder_abc/my_resource.txt")
assert not dir_ends_with_abc("/foo/bar/folder_def/my_resource.txt")
```

Example: Construct a predicate for xml files:

```
xml_file = rf.filename_pattern_predicate(".xml$")

assert xml_file("my_resource.xml")
assert not xml_file("my_resource.txt")
```

Example: Construct a predicate for xml files in folders that end with 'abc':

```
import rspub.util.gates as lf
xml_file_in_abc = lf.and_(dir_ends_with_abc, xml_file)

assert xml_file_in_abc("/foo/bar/folder_abc/my_resource.xml")
assert not xml_file_in_abc("/foo/bar/folder_abc/my_resource.txt")
assert not xml_file_in_abc("/foo/bar/folder_def/my_resource.xml")
```

Example: Construct a predicate for files modified after 31 July 2016:

```
recent = rf.last_modified_after_predicate("2016-08-01")
```

Example: Test a gate that will allow xml files from folders that end with 'abc', but that excludes files modified after 31 July 2016:

```
includes = [xml_files_in_abc]
excludes = [recent]
resource_gate = lf.gate(includes, excludes)
```

If you are satisfied with your gate the *includes* and *excludes* can be contributed by your ResourceGateBuilder.

Implement the build methods

When implementing the build methods *build_includes()* and *build_excludes()* it is good to know that the first builder in the chain is the default *ResourceGateBuilder* as implemented below. It defines the includes very wide: allow anything found in the *resource_dir()*. In order to effectively contribute your including predicates, you should not append them to the given list but replace the list with your own list of predicates. The excluding list as defined by the default class:*ResourceGateBuilder* contains niceties as filter out hidden files, exclude files in your *metadata_dir()* etc. If these default excluding predicates are not in your way you can append your excludes to this default list in the method *build_excludes()*.

class rspub.pluggable.gate.**ResourceGateBuilder**(*resource_dir=None*, *metadata_dir=None*, *plugin_dir=None*)

Bases: *rspub.util.gates.GateBuilder*

Default *ResourceGateBuilder*

This default class builds a *gate()* that allows any file that is encountered. It will exclude however any file that is not in *resource_dir()* or any of its subdirectories, hidden files and files from the directories *metadata_dir()*, *plugin_dir()* and *.well-known/resourcesync*.

__init__(*resource_dir=None*, *metadata_dir=None*, *plugin_dir=None*)

build_includes(*includes: list*)

build_excludes(*excludes: list*)

genindex | modindex | search

CHAPTER 4

rspub.util

Observable and observers

module: rspub.util.observe

exception rspub.util.observe.**ObserverInterruptException**
Bases: RuntimeError

class rspub.util.observe.**Observable**
Bases: object

__init__()

register(**observers*)

unregister(*observer*)

unregister_all()

observers_inform(**args*, ***kwargs*)

observers_confirm(**args*, ***kwargs*)

class rspub.util.observe.**Observer**
Bases: object

inform(**args*, ***kwargs*)

confirm(**args*, ***kwargs*)

class rspub.util.observe.**EventObserver**
Bases: *rspub.util.observe.Observer*

inform(**args*, ***kwargs*)

pass_inform(**args*, ***kwargs*)

confirm(**args*, ***kwargs*)

pass_confirm(**args*, ***kwargs*)

class rspub.util.observe.**EventPrinter**(*event_level=0*, *print_kwargs=True*)
Bases: *rspub.util.observe.Observer*

__init__(*event_level=0*, *print_kwargs=True*)

inform(**args*, ***kwargs*)

confirm(**args*, ***kwargs*)

class rspub.util.observe.**EventLogger**(*logging_level=10*, *event_level=0*)
Bases: *rspub.util.observe.Observer*

__init__(*logging_level=10*, *event_level=0*)

inform(**args*, ***kwargs*)

confirm(**args*, ***kwargs*)

class rspub.util.observe.**SelectiveEventPrinter**(**events*)
Bases: *rspub.util.observe.Observer*

__init__(**events*)

inform(**args*, ***kwargs*)

class rspub.util.observe.**SelectiveEventLogger**(**events*, *level=10*)
Bases: *rspub.util.observe.Observer*

__init__(**events*, *level=10*)

inform(**args*, ***kwargs*)

Logical functions and gate builders

module: rspub.util.gates

```
Logical functions, gate and gate builders
```

Logical functions

Each logical function takes a one-argument predicate or a list of one-argument predicates. In turn each logical function returns a one-argument predicate that is the chain of, or the negation of its arguments. There are functions to chain predicates along *not_()*, *and_()*, *or_()*, *nand_()*, *nor_()*, *xor_()* and *xnor_()*.

Each logical function, before returning the chained predicate, will check if the predicates in the argument list are truly one-argument predicates. The behavior after detection of a wrong argument can be set by the module-method *set_stop_on_creation_error()*. The default behavior after detection of a wrong argument is to throw a *GateCreationException*.

Example usage

Given closures or lambda's:

```
>>> spam = lambda word : word.startswith("spam")
>>> eggs = lambda word: word.endswith("eggs")
>>> ampersand = lambda word: len(word.split("&")) > 1
```

Now you can create a test for spam & eggs:

```
>>> from rspub.util.gates import and_
>>> spam_and_eggs = and_(spam, eggs, ampersand)
```

and reuse *spam* and *eggs* to create spam nor eggs:

```
>>> from rspub.util.gates import nor_
>>> spam_nor_eggs = nor_(spam, eggs)
```

and use the assembled predicates:

```
>>> spam_and_eggs("spam & eggs")
True
>>> spam_and_eggs("spamming leggs")
False
>>> spam_nor_eggs("bacon")
True
```

Of course your closures and lambda's all need to be able to handle the type of argument given.

Gate

The function *gate()* takes two lists of predicates, *includes* and *excludes*. Includes is the list of predicates that can permit *x* through the gate; excludes is the list of predicates that can prevent *x* from passing the gate.

Building gates

The abstract class *GateBuilder* defines the methods to construct a GateBuilder. The concrete class *PluggedInGateBuilder* walks zero or more plugin directories looking for specifically named builders in order to build a customized *gate()*.

If *GateBuilder* s are chained, a builder can overrule *includes* and *excludes* from previous builders.

Classes and functions

rspub.util.gates.**not_**(*predicate*)
Creates the negation of the given *predicate*

The outcome of a *not_f* for any *x* is:

```
f(x) = not p(x)
```

where *p* is the given predicate.

Parameters **predicate** – the predicate to negate

Returns a new predicate implementing the negation of the given predicate

rspub.util.gates.**and_**(**predicates*)
Creates the logical conjunction of the given *predicates*

Chains *predicates* in *and*. The outcome of an *and_f* for any *x* is:

```
f(x) = p_1(x) and p_2(x) and ... and p_n(x)
```

where *p_1 ... p_n* are the given predicates.

The chain of predicates is **True** if all predicates are **True**, otherwise **False**. Outcome **True** in effect says that all of the predicates evaluated as **True**.

Logical performance has been optimized. i.e. *A and B and C* is **False** if *A* evaluates as **False**; do not test *B* and *C* in this case.

Parameters **predicates** – predicates to chain in and.

Returns a new predicate implementing the combined *and* of the given predicates

rspub.util.gates.**nor_**(**predicates*)

Creates the joint denial of the given *predicates*

Chains *predicates* in *nor*. The outcome of a *nor_f* for any *x* is:

```
f(x) = not(p_1(x) or p_2(x) or ... or p_n(x))
```

where *p_1 ... p_n* are the given predicates.

The chain of predicates is **False** if at least one predicate is **True**, otherwise **True**. Outcome **True** in effect says that neither one of the predicates evaluated as **True**.

Logical performance has been optimized. i.e. *A nor B nor C* is **False** if *A* evaluates as **True**; do not test *B* and *C* in this case.

Parameters **predicates** – predicates to chain in nor.

Returns a new predicate implementing the combined *nor* of the given predicates

rspub.util.gates.**or_**(**predicates*)

Creates the logical inclusive disjunction of the given *predicates*

Chains *predicates* in *or*. The outcome of an *or_f* for any *x* is:

```
f(x) = p_1(x) or p_2(x) or ... or p_n(x)
```

where *p_1 ... p_n* are the given predicates.

The chain of predicates is **True** if at least one predicate is **True**, otherwise **False**. Outcome **True** in effect says that at least one of the predicates evaluated as **True**.

Logical performance has been optimized. i.e. *A or B or C* is **True** if *A* evaluates as **True**; do not test *B* and *C* in this case.

Parameters **predicates** – predicates to chain in or.

Returns a new predicate implementing the combined *or* of the given predicates

rspub.util.gates.**nand_**(**predicates*)

Creates the alternative denial of the given *predicates*

Chains *predicates* in *nand*. The outcome of a *nand_f* for any *x* is:

```
f(x) = not(p_1(x) and p_2(x) and ... and p_n(x))
```

where *p_1 ... p_n* are the given predicates.

The chain of predicates is **False** if all predicates are **True**, otherwise **True**. Outcome **True** in effect says that at least one of the predicates evaluated as **False**.

Logical performance has been optimized. i.e. *A nand B nand C* is **True** if *A* evaluates as **False**; do not test *B* and *C* in this case.

Parameters **predicates** – predicates to chain in nand.

Returns a new predicate implementing the combined *nand* of the given predicates

rspub.util.gates.**xor_**(**predicates*)

Creates the exclusive disjunction of the given *predicates*

Chains *predicates* in *xor*. The outcome of an *xor_f* for any *x* is:

```
f(x) = p_1(x) xor p_2(x) xor ... xor p_n(x)
```

where *p_1* ... *p_n* are the given predicates.

One definition of xor says: "A chain of XORs—a XOR b XOR c XOR d (and so on)—is true whenever an odd number of the inputs are true and is false whenever an even number of inputs are true. https://en.wikipedia.org/wiki/Exclusive_or

Some definitions even deny that there can be more than two inputs: "a Boolean operator working on two variables that has the value one if one but not both of the variables is one". https://www.google.nl/search?q=define+exclusive+OR

However, this implementation adheres to:

The chain of predicates is **True** if one and only one predicate is **True**, otherwise **False**.

Parameters **predicates** – predicates to chain with xor.

Returns a new predicate implementing the combined *xor* of the given predicates

rspub.util.gates.**xnor_**(**predicates*)

Creates the logical equality of the given *predicates*

Chains *predicates* in *xnor*. The outcome of an *xnor_f* for any *x* is:

```
f(x) = (p_1(x) and p_2(x) and ... and p_n(x)) or not(p_1(x) or p_2(x) or ... or p_
↪n(x))
```

where *p_1* ... *p_n* are the given predicates.

The chain of predicates is **True** if *all* predicates evaluate as **True** or *all* predicates evaluate as **False**. (So this is *not* the negation of xor as implemented above.)

Parameters **predicates** – predicates to chain with xnor.

Returns a new predicate implementing the combined *xnor* of the given predicates

rspub.util.gates.**gate**(*includes=[]*, *excludes=[]*)

Creates the logical conjunction of or_(*includes*), nor_(*excludes*)

Chains *including* predicates and *excluding* predicates. The outcome of a gate *g* for any *x* is:

```
g(x) = (i_1(x) or i_2(x) or ... or i_n(x)) and not(e_1(x) or e_2(x) or ... or e_
↪n(x))
```

where *i_1* ... *i_n* are given including predicates and *e_1* ... *e_n* are given excluding predicates.

The gate evaluates as **True** if at least one of *includes* is **True** and none of *excludes* are **True**.

Parameters

- **includes** (*list*) – predicates that permit *x* through gate

- **excludes** (*list*) – predicates that restrict *x* from gate

Returns a new predicate implementing the combined functions given in *includes* and *excludes*

class rspub.util.gates.**GateBuilder**
Bases: object

Abstract builder class for gates

GateBuilders should extend this abstract class or implement the next two methods. In these methods GateBuilders are free to extend on previously defined lists of permitting and restricting predicates, remove elements from them or overrule previous steps and return complete new lists.

See also:

gate()

build_includes(*includes: list*) → list
Define the list of permitting predicates

Either rework the given list (append, extend, remove, replace), return the given list or return a complete new list. The returned list should consist of one-argument predicates.

Parameters **includes** (*list*) – the list of permitting predicates (from previous builders)

Returns the list of permitting predicates as defined by this GateBuilder

build_excludes(*excludes: list*) → list
Define the list of restricting predicates

Either rework the given list (append, extend, remove, replace), return the given list or return a complete new list. The returned list should consist of one-argument predicates.

Parameters **excludes** (*list*) – the list of restricting predicates (from previous builders)

Returns the list of restricting predicates as defined by this GateBuilder

class rspub.util.gates.**PluggedInGateBuilder**(*builder_name: str*, *first_builder: rspub.util.gates.GateBuilder = None*, **plugin_directories: str*)
Bases: *rspub.util.gates.GateBuilder*

Builds pluggable *gates*

The PluggedInGateBuilder can be given zero or more directories where it will recursively look for GateBuilders of the given *builder_name*. It will then instantiate the builder and give it the opportunity to determine the list of including predicates and the list of excluding predicates as this builder calls *build_includes()* and *build_excludes()* on the plugged-in builder.

A class in the given *plugin_directories* will qualify as builder if at least

•it has a name equal to the given *builder_name* and

•it is a subclass of *GateBuilder* or it implements both methods of this class.

The final *gate()* can be obtained by calling *build_gate()*.

__init__(*builder_name: str*, *first_builder: rspub.util.gates.GateBuilder = None*, **plugin_directories: str*)
Initialize a *PluggedInGateBuilder*

Parameters

- **builder_name** (*str*) – the class name (either simple or qualified) of the class implementing the GateBuilder methods.

- **first_builder** (GateBuilder) – builder of default or initial predicates, may be **None**
- **plugin_directories** (*str*) – the directories where to search for GateBuilders with the given builder_name

build_includes(*includes=[]*) → list
Set initial permitting predicates

Parameters **includes** (*list*) – the list of initial permitting predicates

Returns the list of initial permitting predicates

Raises *GateCreationException* if a predicate was not a one-argument predicate

build_excludes(*excludes=[]*) → list
Set initial restricting predicates

Parameters **excludes** (*list*) – the list of initial restricting predicates

Returns the list of initial restricting predicates

Raises *GateCreationException* if a predicate was not a one-argument predicate

build_gate() → <function gate at 0x7f1329e94d08>
Build a gate as defined by found GateBuilders in *plugin_directories*

Found GateBuilders are given the chance to modify the lists *includes* and *excludes*. The initial lists *includes* and *excludes* are populated by predicates as defined by *first_builder*. If no *first_builder* was given, the initial lists will be empty lists.

Returns *gate()* as defined by found GateBuilders.

Raises *GateCreationException* if a gate could not be created because a given value is not a one-argument predicate.

Raises *GateBuilderException* if a gate could not be built because of inappropriate behavior of a GateBuilder.

See also:

gate(), *GateBuilder*, *GateBuilder.build_includes()*, *GateBuilder.build_excludes()*

exception rspub.util.gates.**GateCreationException**
Bases: ValueError

Indicates a gate could not be created because a given value is not a one-argument predicate

exception rspub.util.gates.**GateBuilderException**
Bases: *rspub.util.gates.GateCreationException*

Indicates a gate could not be built because of inappropriate behavior of a GateBuilder

rspub.util.gates.**set_stop_on_creation_error**(*stop*)
Determine module-wide behavior on gate creation errors

The function *is_one_arg_predicate()* will be called throughout this module by logical functions and gate builder classes in order to detect if a given value is a one-argument predicate. What the behavior of the detecting function will be after detecting a wrong input value can be determined by this method. Either an error message will be logged (stop = **False**) or a *GateCreationException* will be raised (stop = **True**).

Parameters **stop** (*boolean*) – **True** for stop on creation error, **False** otherwise

Returns the previous state

rspub.util.gates.**stop_on_creation_error**()
Module-wide behavior on gate creation errors

Returns **True** if stops on creation error, **False** otherwise

rspub.util.gates.**is_one_arg_predicate**(*p*)
Determines if the given *p* is a one-argument predicate

Parameters **p** – value to be inspected

Returns **True** if *p* is a one-argument predicate, **False** otherwise

Raises *GateCreationException* if *p* is not a one-argument predicate and *stop_on_creation_error()* is **True**

See also:

set_stop_on_creation_error()

Resource filters

module: rspub.util.resourcefilter

rspub.util.resourcefilter.**hidden_file_predicate**()

rspub.util.resourcefilter.**directory_pattern_predicate**(*name_pattern=''*)

rspub.util.resourcefilter.**windows_to_unix**(*path*)

rspub.util.resourcefilter.**filename_pattern_predicate**(*name_pattern=''*)

rspub.util.resourcefilter.**last_modified_after_predicate**(*t=0*)

Light plugin framework

module: rspub.util.plugg

Py-module and -class inspector

rspub.util.plugg.**APPLICATION_HOME = '/home/docs/checkouts/readthedocs.org/user_builds/rspub-core/checkouts/lates**
The absolute path to the directory that is the application home or root directory.

During run time. So the value shown in documentation is not a constant!

class rspub.util.plugg.**Inspector**(*stop_on_error=False*)
Bases: object

Find py-modules and -classes in directories.

This class loads modules during its inspection. What the behavior will be upon encountering an ImportError can be set by the constructor parameter *stop_on_error* (boolean). It will then either log the exception (default) or raise the exception.

__init__(*stop_on_error=False*)
Initialize an *Inspector*.

Parameters **stop_on_error** – **True** for stop on error, **False** otherwise

static list_py_files(**directories*) → str

Generator of py filenames.

Walks the given directories one-by-one recursively and yields each py-file it encounters. A file is considered py-file when its filename ends with *.py*.

Files *__init__.py* and *setup.py* are neglected.

Parameters **directories** (*str*) – directories to search

Returns yields absolute filenames of py-files

load_modules(**directories*)

Generator of modules.

Walks the given directories one-by-one recursively and yields each module it encounters. The encountered modules will be imported. What the behavior will be upon encountering an ImportError can be set by the constructor parameter *stop_on_error* (boolean).

Parameters **directories** (*str*) – directories to search

Returns yields imported modules

list_classes(**directories*)

Generator of classes.

Walks the given directories one-by-one recursively and yields each class it encounters.

Parameters **directories** (*str*) – directories to search

Returns yields encountered classes

list_classes_filtered(*predicates=[]*, **directories*)

Generator of filtered classes.

Walks the given directories one-by-one recursively and yields encountered classes *if* they pass all the predicates given in *predicates*.

Parameters

- **predicates** (*list*) – a list of one-argument predicates that filter classes
- **directories** (*str*) – directories to search

Returns yields encountered classes that pass the predicates

rspub.util.plugg.**is_subclass_of**(*super*)

Predicate for subclass detection

```
f(cls) = issubclass(cls, super)
```

Parameters **super** – the superclass in the detection

Returns lambda for class subclass detection

rspub.util.plugg.**is_qnamed**(*qname*)

Predicate for qualified class-name detection.

```
f(cls) = cls.qualified_name == qname
```

Parameters **qname** – the qualified name in the detection

Returns lambda for qualified class-name detection

rspub.util.plugg.**is_named**(*name*)
Predicate for loose class-name detection.

```
f(cls) = cls.name == name or cls.qualified_name == name
```

Parameters **name** – the class-name or qualified class-name in the detection

Returns lambda for loose class-name detection

rspub.util.plugg.**from_module**(*module_name*)
Predicate for module-name detection.

```
f(cls) = cls.module_name == module_name
```

Parameters **module_name** – the module-name in the detection

Returns lambda for module-name detection

rspub.util.plugg.**has_function**(*function_name*)
Predicate for class function detection.

```
f(cls) = cls.has_function_name(function_name)
```

Parameters **function_name** – the function name in the detection

Returns closure for function name detection

Defaults

module: rspub.util.defaults

Various utility functions

rspub.util.defaults.**sanitize_url_path**(*value*)

rspub.util.defaults.**sanitize_string**(*value*)

rspub.util.defaults.**w3c_datetime**(*i*)
given seconds since the epoch, return a dateTime string. from: https://gist.github.com/mnot/246088

rspub.util.defaults.**w3c_now**()

rspub.util.defaults.**md5_for_file**(*filename*, *block_size=16384*)
Compute MD5 digest for a file

Optional block_size parameter controls memory used to do MD5 calculation. This should be a multiple of 128 bytes.

rspub.util.defaults.**mime_type**(*filename*)
Not too reliable mime type analyzer.

Python Module Index

r

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

X

Z