python-syncthing Documentation

Release 2.3.0

Blake VandeMerwe

Mar 06, 2018

Contents

Python bindings to the Syncthing REST interface.

- Syncthing
- Syncthing REST Documentation
- Syncthing Forums
- Pypi (`syncthing`)

CHAPTER 1

Reference

CHAPTER 2

Module

exception syncthing.**SyncthingError**
Base Syncthing Exception class all non-assert errors will raise from.

class syncthing.**ErrorEvent**(*when*, *message*)
tuple[datetime.datetime,str]: used to process error lists more easily, instead of by two-key dictionaries.

message
Alias for field number 1

when
Alias for field number 0

class syncthing.**BaseAPI**(*api_key*, *host='localhost'*, *port=8384*, *timeout=10.0*, *is_https=False*, *ssl_cert_file=None*)
Placeholder for HTTP REST API URL prefix.

class syncthing.**System**(*api_key*, *host='localhost'*, *port=8384*, *timeout=10.0*, *is_https=False*, *ssl_cert_file=None*)
HTTP REST endpoint for System calls.

add_discovery(*device*, *address*)
Add an entry to the discovery cache.

Parameters

- **device** (*str*) – Device ID.
- **address** (*str*) – destination address, a valid hostname or IP address that's serving a Syncthing instance.

Returns None

browse(*path=None*)
Returns a list of directories matching the path given.

Parameters **path** (*str*) – glob pattern.

Returns List[str]

can_upgrade()
Returns when there's a newer version than the instance running.

Returns bool

clear()
Remove all recent errors.

Returns None

clear_errors()
Alias function for *clear()*.

config()
Returns the current configuration.

Returns dict

```
>>> s = _syncthing().system
>>> config = s.config()
>>> config
...
{...}
>>> 'version' in config and config['version'] >= 15
True
>>> 'folders' in config
True
>>> 'devices' in config
True
```

config_insync()
Returns whether the config is in sync, i.e. whether the running configuration is the same as that on disk.

Returns bool

connections()
Returns the list of configured devices and some metadata associated with them. The list also contains the local device itself as not connected.

Returns dict

```
>>> s = _syncthing().system
>>> connections = s.connections()
>>> sorted([k for k in connections.keys()])
['connections', 'total']
>>> isinstance(connections['connections'], dict)
True
>>> isinstance(connections['total'], dict)
True
```

debug()
Returns the set of debug facilities and which of them are currently enabled.

Returns dict

```
>>> s = _syncthing().system
>>> debug = s.debug()
>>> debug
...
{...}
>>> len(debug.keys())
```

```
2
>>> 'enabled' in debug and 'facilities' in debug
True
>>> isinstance(debug['enabled'], list) or debug['enabled'] is None
True
>>> isinstance(debug['facilities'], dict)
True
```

disable_debug(**on*)
Disables debugging for specified facilities.

Parameters **on** (*str*) – debugging points to apply `disable`.

Returns None

discovery()
Returns the contents of the local discovery cache.

Returns dict

do_upgrade()
Perform an upgrade to the newest released version and restart. Does nothing if there is no newer version than currently running.

Returns None

enable_debug(**on*)
Enables debugging for specified facilities.

Parameters **on** (*str*) – debugging points to apply `enable`.

Returns None

errors()
Returns the list of recent errors.

Returns of *ErrorEvent* tuples.

Return type list

log()
Returns the list of recent log entries.

Returns dict

pause(*device*)
Pause the given device.

Parameters **device** (*str*) – Device ID.

Returns with keys `success` and `error`.

Return type dict

ping(*with_method='GET'*)
Pings the Syncthing server.

Parameters **with_method** (*str*) – uses a given HTTP method, options are `GET` and `POST`.

Returns dict

reset()
Erase the current index database and restart Syncthing.

Returns None

reset_folder(*folder*)
Erase the database index from a given folder and restart Syncthing.

Parameters **folder** (*str*) – Folder ID.

Returns None

restart()
Immediately restart Syncthing.

Returns None

resume(*device*)
Resume the given device.

Parameters **device** (*str*) – Device ID.

Returns with keys `success` and `error`.

Return type dict

set_config(*config*, *and_restart=False*)
Post the full contents of the configuration, in the same format as returned by `config()`. The configuration will be saved to disk and the `configInSync` flag set to `False`. Restart Syncthing to activate.

show_error(*message*)
Send an error message to the active client. The new error will be displayed on any active GUI clients.

Parameters **message** (*str*) – Plain-text message to display.

Returns None

```
>>> s = _syncthing()
>>> s.system.show_error('my error msg')
>>> s.system.errors()[0]
...
ErrorEvent(when=datetime.datetime(...), message='"my error msg"')
>>> s.system.clear_errors()
>>> s.system.errors()
[]
```

shutdown()
Causes Syncthing to exit and not restart.

Returns None

status()
Returns information about current system status and resource usage.

Returns dict

upgrade()
Checks for a possible upgrade and returns an object describing the newest version and upgrade possibility.

Returns dict

version()
Returns the current Syncthing version information.

Returns dict

class syncthing.**Database**(*api_key*, *host='localhost'*, *port=8384*, *timeout=10.0*, *is_https=False*, *ssl_cert_file=None*)
HTTP REST endpoint for Database calls.

browse (*folder*, *levels=None*, *prefix=None*)
Returns the directory tree of the global model.

Directories are always JSON objects (map/dictionary), and files are always arrays of modification time and size. The first integer is the files modification time, and the second integer is the file size.

Parameters

- **folder** (*str*) – The root folder to traverse.
- **levels** (*int*) – How deep within the tree we want to dwell down. (0 based, defaults to unlimited depth)
- **prefix** (*str*) – Defines a prefix within the tree where to start building the structure.

Returns dict

completion (*device*, *folder*)
Returns the completion percentage (0 to 100) for a given device and folder.

Parameters

- **device** (*str*) – The Syncthing device the folder is syncing to.
- **folder** (*str*) – The folder that is being synced.

Returs: int

file (*folder*, *file_*)
Returns most data available about a given file, including version and availability.

Parameters

- **folder** (*str*) –
- **file** (*str*) –

Returns dict

ignores (*folder*)
Returns the content of the `.stignore` as the ignore field. A second field, expanded, provides a list of strings which represent globbing patterns described by gobwas/glob (based on standard wildcards) that match the patterns in `.stignore` and all the includes.

If appropriate these globs are prepended by the following modifiers: `!` to negate the glob, `(?i)` to do case insensitive matching and `(?d)` to enable removing of ignored files in an otherwise empty directory.

Args: folder

Returns: dict

need (*folder*, *page=None*, *perpage=None*)
Returns lists of files which are needed by this device in order for it to become in sync.

Args: folder (str): page (int): If defined applies pagination accross the

collection of results.

perpage (int): If defined applies pagination across the collection of results.

Returns: dict

override (*folder*)
Request override of a send-only folder.

Parameters **folder** (*str*) – folder ID.

Returns dict

prio (*folder*, *file_*)
Moves the file to the top of the download queue.

Parameters

- **folder** (*str*) –
- **file** (*str*) –

Returns dict

scan (*folder*, *sub=None*, *next_=None*)
Request immediate rescan of a folder, or a specific path within a folder.

Args: folder (str): Folder ID. sub (str): Path relative to the folder root. If sub is omitted

the entire folder is scanned for changes, otherwise only the given path children are scanned.

next_ (int): Delays Syncthing's automated rescan interval for a given amount of seconds.

Returns: str

set_ignores (*folder*, **patterns*)
Applies `patterns` to `folder`'s `.stignore` file.

Parameters

- **folder** (*str*) –
- **patterns** (*str*) –

Returns dict

status (*folder*)
Returns information about the current status of a folder.

Note: This is an expensive call, increasing CPU and RAM usage on the device. Use sparingly.

Parameters **folder** (*str*) – Folder ID.

Returns dict

class syncthing.**Statistics** (*api_key*, *host='localhost'*, *port=8384*, *timeout=10.0*, *is_https=False*, *ssl_cert_file=None*)
HTTP REST endpoint for Statistic calls.

device ()
Returns general statistics about devices.

Currently, only contains the time the device was last seen.

Returns dict

folder ()
Returns general statistics about folders.

Currently contains the last scan time and the last synced file.

Returns dict

class syncthing.**Syncthing**(*api_key*, *host='localhost'*, *port=8384*, *timeout=10.0*, *is_https=False*, *ssl_cert_file=None*)

Default interface for interacting with Syncthing server instance.

Parameters

- **api_key** (*str*) –
- **host** (*str*) –
- **port** (*int*) –
- **timeout** (*float*) –
- **is_https** (*bool*) –
- **ssl_cert_file** (*str*) –

system
instance of *System*.

database
instance of *Database*.

stats
instance of *Statistics*.

misc
instance of *Misc*.

Note:

- attribute db is an alias of *database*
- attribute sys is an alias of *system*

---

syncthing.**keys_to_datetime**(*obj*, **keys*)

Converts all the keys in an object to DateTime instances.

Parameters

- **obj** (*dict*) – the JSON-like dict object to modify inplace.
- **keys** (*str*) – keys of the object being converted into DateTime instances.

Returns obj inplace.

Return type dict

```
>>> keys_to_datetime(None) is None
True
>>> keys_to_datetime({})
{}
>>> a = {}
>>> id(keys_to_datetime(a)) == id(a)
True
>>> a = {'one': '2016-06-06T19:41:43.039284',
         'two': '2016-06-06T19:41:43.039284'}
>>> keys_to_datetime(a) == a
True
>>> keys_to_datetime(a, 'one')['one']
datetime.datetime(2016, 6, 6, 19, 41, 43, 39284)
```

```
>>> keys_to_datetime(a, 'one')['two']
'2016-06-06T19:41:43.039284'
```

syncthing.**parse_datetime**(*s*, ***kwargs*)

Converts a time-string into a valid `DateTime` object.

Args: s (str): string to be formatted.

`**kwargs` is passed directly to `dateutil_parser()`.

Returns: `DateTime`

CHAPTER 3

System Endpoints

class syncthing.**System**

HTTP REST endpoint for System calls.

add_discovery(*device*, *address*)

Add an entry to the discovery cache.

Parameters

- **device** (*str*) – Device ID.
- **address** (*str*) – destination address, a valid hostname or IP address that's serving a Syncthing instance.

Returns None

browse(*path=None*)

Returns a list of directories matching the path given.

Parameters **path** (*str*) – glob pattern.

Returns List[str]

can_upgrade()

Returns when there's a newer version than the instance running.

Returns bool

clear()

Remove all recent errors.

Returns None

clear_errors()

Alias function for *clear()*.

config()

Returns the current configuration.

Returns dict

```
>>> s = _syncthing().system
>>> config = s.config()
>>> config
...
{...}
>>> 'version' in config and config['version'] >= 15
True
>>> 'folders' in config
True
>>> 'devices' in config
True
```

config_insync()
Returns whether the config is in sync, i.e. whether the running configuration is the same as that on disk.

Returns bool

connections()
Returns the list of configured devices and some metadata associated with them. The list also contains the local device itself as not connected.

Returns dict

```
>>> s = _syncthing().system
>>> connections = s.connections()
>>> sorted([k for k in connections.keys()])
['connections', 'total']
>>> isinstance(connections['connections'], dict)
True
>>> isinstance(connections['total'], dict)
True
```

debug()
Returns the set of debug facilities and which of them are currently enabled.

Returns dict

```
>>> s = _syncthing().system
>>> debug = s.debug()
>>> debug
...
{...}
>>> len(debug.keys())
2
>>> 'enabled' in debug and 'facilities' in debug
True
>>> isinstance(debug['enabled'], list) or debug['enabled'] is None
True
>>> isinstance(debug['facilities'], dict)
True
```

disable_debug(**on*)
Disables debugging for specified facilities.

Parameters **on** (*str*) – debugging points to apply `disable`.

Returns None

discovery()
Returns the contents of the local discovery cache.

Returns dict

do_upgrade()
Perform an upgrade to the newest released version and restart. Does nothing if there is no newer version than currently running.

Returns None

enable_debug(**on*)
Enables debugging for specified facilities.

Parameters **on** (*str*) – debugging points to apply `enable`.

Returns None

errors()
Returns the list of recent errors.

Returns of *ErrorEvent* tuples.

Return type list

log()
Returns the list of recent log entries.

Returns dict

pause(*device*)
Pause the given device.

Parameters **device** (*str*) – Device ID.

Returns with keys `success` and `error`.

Return type dict

ping(*with_method='GET'*)
Pings the Syncthing server.

Parameters **with_method** (*str*) – uses a given HTTP method, options are `GET` and `POST`.

Returns dict

prefix = '/rest/system/'

reset()
Erase the current index database and restart Syncthing.

Returns None

reset_folder(*folder*)
Erase the database index from a given folder and restart Syncthing.

Parameters **folder** (*str*) – Folder ID.

Returns None

restart()
Immediately restart Syncthing.

Returns None

resume(*device*)
Resume the given device.

Parameters **device** (*str*) – Device ID.

Returns with keys `success` and `error`.

Return type dict

set_config(*config*, *and_restart=False*)
Post the full contents of the configuration, in the same format as returned by `config()`. The configuration will be saved to disk and the `configInSync` flag set to `False`. Restart Syncthing to activate.

show_error(*message*)
Send an error message to the active client. The new error will be displayed on any active GUI clients.

Parameters **message** (*str*) – Plain-text message to display.

Returns None

```
>>> s = _syncthing()
>>> s.system.show_error('my error msg')
>>> s.system.errors()[0]
...
ErrorEvent(when=datetime.datetime(...), message='"my error msg"')
>>> s.system.clear_errors()
>>> s.system.errors()
[]
```

shutdown()
Causes Syncthing to exit and not restart.

Returns None

status()
Returns information about current system status and resource usage.

Returns dict

upgrade()
Checks for a possible upgrade and returns an object describing the newest version and upgrade possibility.

Returns dict

version()
Returns the current Syncthing version information.

Returns dict

CHAPTER 4

Database Endpoints

class syncthing.**Database**

HTTP REST endpoint for Database calls.

browse(*folder*, *levels=None*, *prefix=None*)

Returns the directory tree of the global model.

Directories are always JSON objects (map/dictionary), and files are always arrays of modification time and size. The first integer is the files modification time, and the second integer is the file size.

Parameters

- **folder** (*str*) – The root folder to traverse.
- **levels** (*int*) – How deep within the tree we want to dwell down. (0 based, defaults to unlimited depth)
- **prefix** (*str*) – Defines a prefix within the tree where to start building the structure.

Returns dict

completion(*device*, *folder*)

Returns the completion percentage (0 to 100) for a given device and folder.

Parameters

- **device** (*str*) – The Syncthing device the folder is syncing to.
- **folder** (*str*) – The folder that is being synced.

Returs: int

file(*folder*, *file_*)

Returns most data available about a given file, including version and availability.

Parameters

- **folder** (*str*) –
- **file** (*str*) –

Returns dict

ignores (*folder*)
Returns the content of the `.stignore` as the ignore field. A second field, expanded, provides a list of strings which represent globbing patterns described by gobwas/glob (based on standard wildcards) that match the patterns in `.stignore` and all the includes.

If appropriate these globs are prepended by the following modifiers: `!` to negate the glob, `(?i)` to do case insensitive matching and `(?d)` to enable removing of ignored files in an otherwise empty directory.

Args: folder

Returns: dict

need (*folder*, *page=None*, *perpage=None*)
Returns lists of files which are needed by this device in order for it to become in sync.

Args: folder (str): page (int): If defined applies pagination accross the

collection of results.

perpage (int): If defined applies pagination across the collection of results.

Returns: dict

override (*folder*)
Request override of a send-only folder.

Parameters **folder** (*str*) – folder ID.

Returns dict

prefix = '/rest/db/'

prio (*folder*, *file_*)
Moves the file to the top of the download queue.

Parameters

- **folder** (*str*) –
- **file** (*str*) –

Returns dict

scan (*folder*, *sub=None*, *next_=None*)
Request immediate rescan of a folder, or a specific path within a folder.

Args: folder (str): Folder ID. sub (str): Path relative to the folder root. If sub is omitted

the entire folder is scanned for changes, otherwise only the given path children are scanned.

next_ (int): Delays Syncthing's automated rescan interval for a given amount of seconds.

Returns: str

set_ignores (*folder*, **patterns*)
Applies `patterns` to `folder`'s `.stignore` file.

Parameters

- **folder** (*str*) –

- **patterns** (*str*) –

Returns dict

status (*folder*)
Returns information about the current status of a folder.

Note: This is an expensive call, increasing CPU and RAM usage on the device. Use sparingly.

Parameters **folder** (*str*) – Folder ID.

Returns dict

CHAPTER 5

Events Endpoints

class syncthing.**Events**

HTTP REST endpoints for Event based calls.

Syncthing provides a simple long polling interface for exposing events from the core utility towards a GUI.

```
syncthing = Syncthing()
event_stream = syncthing.events(limit=5)

for event in event_stream:
    print(event)
    if event_stream.count > 10:
        event_stream.stop()
```

count

The number of events that have been processed by this event stream.

Returns int

disk_events()

Blocking generator of disk related events. Each event is represented as a `dict` with metadata.

Returns: generator[dict]

prefix = '/rest/'

stop()

Breaks the while-loop while the generator is polling for event changes.

Returns None

CHAPTER 6

Statistic Endpoints

class syncthing.**Statistics**

HTTP REST endpoint for Statistic calls.

device()

Returns general statistics about devices.

Currently, only contains the time the device was last seen.

Returns dict

folder()

Returns general statistics about folders.

Currently contains the last scan time and the last synced file.

Returns dict

prefix = '/rest/stats/'

CHAPTER 7

Misc. Endpoints

class syncthing.**Misc**

HTTP REST endpoint for Miscelaneous calls.

device_id(*id_*)

Verifies and formats a device ID. Accepts all currently valid formats (52 or 56 characters with or without separators, upper or lower case, with trivial substitutions). Takes one parameter, id, and returns either a valid device ID in modern format, or an error.

Parameters **id** (*str*) –

Raises *SyncthingError* – when id_ is an invalid length.

Returns str

language()

Returns a list of canonicalized localization codes, as picked up from the Accept-Language header sent by the browser. By default, this API will return a single element that's empty; however calling `Misc.get()` directly with *lang* you can set specific headers to get values back as intended.

Returns

List[str]

```
>>> s = _syncthing()
>>> len(s.misc.language())
1
>>> s.misc.language()[0]
''
>>> s.misc.get('lang', headers={'Accept-Language': 'en-us'})
['en-us']
```

prefix = '/rest/svc/'

random_string(*length=32*)

Returns a strong random generated string (alphanumeric) of the specified length.

Parameters **length** (*int*) – default 32.

Returns str

```
>>> s = _syncthing()
>>> len(s.misc.random_string())
32
>>> len(s.misc.random_string(32))
32
>>> len(s.misc.random_string(1))
1
>>> len(s.misc.random_string(0))
32
>>> len(s.misc.random_string(None))
32
>>> import string
>>> all_letters = string.ascii_letters + string.digits
>>> all([c in all_letters for c in s.misc.random_string(128)])
True
>>> all([c in all_letters for c in s.misc.random_string(1024)])
True
```

report()

Returns the data sent in the anonymous usage report.

Returns dict

```
>>> s = _syncthing()
>>> report = s.misc.report()
>>> 'version' in report
True
>>> 'longVersion' in report
True
>>> 'syncthing v' in report['longVersion']
True
```

CHAPTER 8

Running Tests

The API doctests rely on the following function to run against your instance. None of the "breaking" calls will be tested. You must set the following environment variables otherwise all tests will fail.

```
def _syncthing():
    KEY = os.getenv('SYNCTHING_API_KEY')
    HOST = os.getenv('SYNCTHING_HOST', '127.0.0.1')
    PORT = os.getenv('SYNCTHING_PORT', 8384)
    IS_HTTPS = bool(int(os.getenv('SYNCTHING_HTTPS', '0')))
    SSL_CERT_FILE = os.getenv('SYNCTHING_CERT_FILE')
    return Syncthing(KEY, HOST, PORT, 10.0, IS_HTTPS, SSL_CERT_FILE)
```

CHAPTER 9

License

Python Module Index

S

Index

S

U

V

W