# Synchronizer

*Release 2.0.0*

**Aug 03, 2020**

# API Reference

A collection of utilities for CGI-VFX to copy files from one place to another, find out basic stat differences between them and handle file sequences and textures (tx files).

CHAPTER 1

# Installation

```
pip install synchronizer
```

# CHAPTER 2

# Getting Started

1. **Process paths** Copies `src_path` to `trg_path`. Takes both files and directories as source. If given source is a file and it's part of a sequence it'll find and copy the entire sequence of files.

```
from synchronizer import copier
copier.process_paths(src_path, trg_path, force_overwrite=True, **kwargs)
'''
kwargs:
include_tx = True
only_tx = True
find_sequence = True
'''
```

2. **Sync status** Compares two files or directory paths and return sync status. Sync status refers to `name` and `os.stat()` comparisons.

```
from synchronizer import syncstatus
syncstatus.get_sync_status(
    src_path, trg_path,
    ignore_name=False,
    ignore_stats=['st_uid', 'st_gid', 'st_atime',
            'st_ctime', 'st_ino', 'st_dev'])
```

3. **Get most recent** Compares two paths and returns whichever has the most recent stat time. Default stat used for comparison is `st_mtime` which is: Time of most recent content modification.

```
from synchronizer import syncstatus
syncstatus.get_most_recent(src_path, trg_path, use_stat='st_mtime')
```

4. **Get sequence files** Find and return all files that are part of a sequence matching `file_path`. If no sequence found, returns `None`. Two files are enough to make a sequence, even if they're not sequential. This assumes the sequence digits are right beside the file extension.

   **e.g.:**

   - C_myfile_v568.jpg

- MJ_thisisafileseq_455868.dpx
- MB_udimsforthewin.1008.tx

```
from synchronizer import utils
utils.get_sequence_files(file_path)
```

## 2.1 Copier module

`synchronizer.copier.`**`process_paths`**(*src_path*, *trg_path*, *force_overwrite=True*, ***kwargs*)

Copies `src_path` to `trg_path`. Takes both files and directories as source. If given source is a file and it's part of a sequence it'll find and copy the entire sequence of files.

**Arguments:** `src_path` {str} – Path to a file or directory

`trg_path` {str} – Path to a directory

**Keyword Arguments:** `force_overwrite` {bool} – Empties trg_path before copying src_path contents. If src_path it's a file it'll only remove that file. (default: {True})

**Optional Keyword Arguments:** `include_tx` {bool} – If tx files are found that match given src_path, they're also copied.

`only_tx` {bool} – Finds tx files that match given src_path, but copies tx only, not src_path. For this flag to work, include_tx must be passed and set to True.

`find_sequence` {bool} – If set to False, it'll skip trying to find sequence files for given src_path (default: {True})

**Returns:** [bool] – If files were processed correctly, True is returned. False otherwise.

## 2.2 Sync Status module

`synchronizer.syncstatus.`**`compare_stats`**(*src_path, trg_path, ignore_name=False, ignore_stats=['st_uid', 'st_gid', 'st_atime', 'st_ctime', 'st_ino', 'st_dev']*)

Compares stats and file names for two given paths. Returns a dict with all comparison results.

**Arguments:** `src_path` {str} – Source path, file or directory

`trg_path` {str} – Target path, file or directory

**Keyword Arguments:** `ignore_name` {bool} – Ignores name comparison (default: {False})

`ignore_stats` {list} – Ignores this list of stats. Names correspond to what os.stat() returns. (default: ['st_uid', 'st_gid', 'st_atime', 'st_ctime', 'st_ino', 'st_dev'])

-'st_mode': 'File type and file mode bits'

-'st_ino': 'inode or file index'

-'st_dev': 'Device'

-'st_nlink': 'Number of hard links'

-'st_uid': 'User id of owner'

-'st_gid': 'Group id of owner'

-'st_size': 'File size'

-'st_atime': 'Most recent access'

-'st_mtime': 'Last modification'

-'st_ctime': 'Most recent metadata change'

**Returns:** [dict] – {Stat description: Comparison result bool}

`synchronizer.syncstatus.`**`get_dir_size`**(*dir_path*)

Walks thru given directory to calculate total size.

**Arguments:** `dir_path` {str} – Directory to measure size.

**Returns:** [int] – Size of directory in bytes, as reported by the sum of all its files os.stat()

[None] – If dir_path is not a directory, returns None

`synchronizer.syncstatus.`**`get_most_recent`**(*src_path*, *trg_path*, *use_stat='st_mtime'*)

Compares two paths and returns whichever has the most recent stat time. Default stat used for comparison is st_mtime which is: Time of most recent content modification.

**Arguments:** `src_path` {str} – Source path, file or directory

`trg_path` {str} – Target path, file or directory

**Keyword Arguments:** `use_stat` {str} – Stat used for comparison (default: {'st_mtime'})

**Valid options:** -'st_mtime': Time of most recent content modification

-'st_atime': Time of most recent access

-'st_ctime': Time of creation on Windows, time of most recent metadata change on Unix

**Returns:** [str] – Path of whichever has the most recent stat time.

[None] – If both path stats are equal or an invalid stat options is passed.

`synchronizer.syncstatus.`**`get_sync_status`**(*src_path, trg_path, ignore_name=False, ignore_stats=['st_uid', 'st_gid', 'st_atime', 'st_ctime', 'st_ino', 'st_dev']*)

Compare two files or directory paths and return sync status. Sync status refers to name and `os.stat()` comparisons.

**Arguments:** `src_path` {str} – Source path, file or directory

`trg_path` {str} – Target path, file or directory

**Keyword Arguments:** `ignore_name` {bool} – Ignores name comparison (default: {False})

`ignore_stats` {list} – Ignores this list of stats. Names correspond to what os.stat() returns. (default: ['st_uid', 'st_gid', 'st_atime', 'st_ctime', 'st_ino', 'st_dev'])

-'st_mode': 'File type and file mode bits'

-'st_ino': 'inode or file index'

-'st_dev': 'Device'

-'st_nlink': 'Number of hard links'

-'st_uid': 'User id of owner'

-'st_gid': 'Group id of owner'

-'st_size': 'File size'

-'st_atime': 'Most recent access'

-'st_mtime': 'Last modification'

-'st_ctime': 'Most recent metadata change'

**Returns:**

**[tuple] – (Status code, Status description)** 1 = "In sync"

2 = "Out of sync"

3 = "Both paths do not exist"

4 = "Source path does not exist"

5 = "Target path does not exist"

6 = "Different kind of paths (file-dir, dir-file)"

7 = "Source and Target are the same"

[None] – Not implemented status comparison

## 2.3 Utilities module

`synchronizer.utils.`**`create_dir`**(*dirpath*)

Creates given directory.

**Arguments:** `dirpath` {str} – Full path to a directory that needs to be created.

**Returns:** [bool] – True if directory creation was successful, False otherwise.

`synchronizer.utils.`**`get_sequence_files`**(*file_path*)

Find and return all files that are part of a sequence matching `file_path`. If no sequence found, returns None. Two files are enough to make a sequence, even if they're not sequential. This assumes the sequence digits are right beside the file extension.

**e.g.:**

- C_myfile_v568.jpg
- MJ_thisisafileseq_4568.dpx
- MB_udimsforthewin.1008.tx

**Arguments:** `file_path` {string} – Path to a file

**Returns:** [list] – List of sequence files including given file_path. None if sequence is not found.

`synchronizer.utils.`**`get_sequence_name_pattern`**(*file_path*)

Finds the name pattern and number of digits that make the name of the file. Both elements are used by other functions to identify file sequences. This assumes the sequence digits are right beside the file extension.

**e.g.:**

- C_myfile_v568.jpg
- MJ_thisisafileseq_4568.dpx
- MB_udimsforthewin.1008.tx

**Arguments:** `file_path` {string} – Full path to a file

**Returns:** [str] – A string consisting of the base name for the file without trailing digits.

**e.g.:**

- File: 'C_cresta_02__MSH-BUMP.1001.png'
- Name Pattern: 'C_cresta_02__MSH-BUMP.'

[None] – If no digits can be found in the name, returns None

`synchronizer.utils.`**`is_sequence`**(*file_path*)

Looks for sibling files in the same directory. Since two sibling files is enough to make a sequence, even if they are not sequential, if it finds one, it'll stop looking and return True. This assumes the sequence digits are right beside the file extension.

**e.g.:**

- C_myfile_v568.jpg
- MJ_thisisafileseq_4568.dpx
- MB_udimsforthewin.1008.tx

If you want to get a complete list of files, use get_sequence_files()

**Arguments:** `file_path` {str} – Full path to a file

**Returns:** [bool] – If another a file is found with the same name pattern, True is returned. Missing files are taken into account.

`synchronizer.utils.`**`is_sequence_complete`**(*files*, *name_pattern*)

Evaluates a list of sequence files, if the sequence is missing one or more files, returns False. If sequence is complete, returns True. This assumes the sequence digits are right beside the file extension.

**e.g.:**

- C_myfile_v568.jpg
- MJ_thisisafileseq_4568.dpx
- MB_udimsforthewin.1008.tx

**Arguments:** `files` {list} – List of complete file paths to a file sequence. You could use get_sequence_files() to get a list.

`name_pattern` {str} – As returned by get_sequence_name_pattern(), It's a string consisting of the base name for the file without trailing digits.

**e.g.:**

- File: 'C_cresta_02__MSH-BUMP.1001.png'
- Name Pattern: 'C_cresta_02__MSH-BUMP.'

**Returns:** [bool] – True if sequence is complete. False otherwise.

# Python Module Index

## S

# Index