pgimp Documentation

Release 1.0.0-alpha-22

Mathias Burger

Oct 03, 2020

Contents

Pgimp let's you interact with gimp in python3. Gimp itself only allows you to use python2 and the documentation of its procedural database (pdb) is C-style and not integrated into your IDE.

Therefore pgimp generates python skeletons for full autocompletion and documentation when writing scripts that will be executed within gimp's python interpreter.

Furthermore pgimp provides python3 classes and methods to interact with gimp files or whole collections of files conveniently.

CHAPTER 1

Use cases

- Autocompletion for writing gimp scripts.
- Batch creation or update of gimp files or data extraction from gimp files.
- Workflows where machine learning data has to be annotated. Raw data can be converted to gimp files where the annotation process can happen (gimp's thresholding tools etc. make it easy to do annotation for pixelwise segmentation). After the masks are created, they can be converted back to e.g. numpy files.

CHAPTER 2

Table of content

2.1 Tutorial

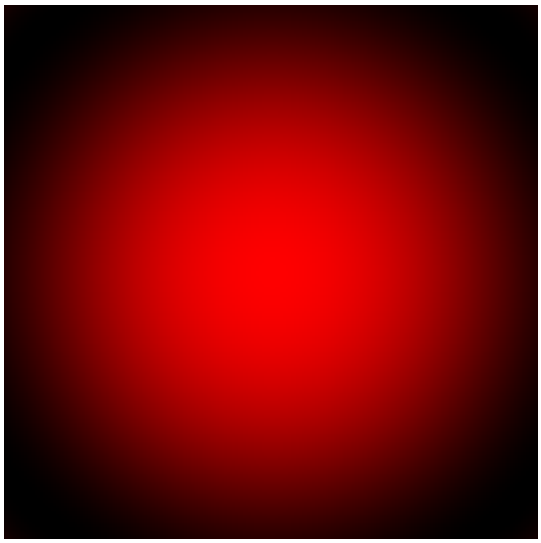
2.1.1 Create an image and export it

Create a temporary xcf file containing a sphere and export it to png.

Source code:

```
import os

import numpy as np

from pgimp.GimpFile import GimpFile
from pgimp.util import file
from pgimp.util.TempFile import TempFile

if __name__ == '__main__':
    img_path = file.relative_to(__file__, '../../../doc/source/_static/img')
    png_file = os.path.join(img_path, 'sphere.png')

    # generate sphere data
    x = np.arange(-1, 1, 0.01)
    y = np.arange(-1, 1, 0.01)
    xx, yy = np.meshgrid(x, y, sparse=True)
    z = np.sin(xx**2 + yy**2)

    # generate rgb image data
    img = np.zeros(shape=(200, 200, 3), dtype=np.uint8)
    img[:, :, 0] = (1-z)*255

    # create temporary gimp file an export to png
    with TempFile('.xcf') as tmp:
        GimpFile(tmp).create('Background', img).export(png_file)
```

Result:



2.1.2 Create a multi layer image and export to npz

Create a multi layer image with a mask and export the layers to a numpy npz archive. Read the npz file, apply the mask, create a new gimp file and export it to png.

Source code:

```
import os

import numpy as np

from pgimp.GimpFile import GimpFile
from pgimp.util import file
from pgimp.util.TempFile import TempFile

if __name__ == '__main__':
    img_path = file.relative_to(__file__, '../../../doc/source/_static/img')
    png_file = os.path.join(img_path, 'mask_applied.png')

    height = 100
    width = 200

    # layer content
    bg = np.zeros(shape=(height, width), dtype=np.uint8)
    fg = np.ones(shape=(height, width), dtype=np.uint8) * 255
    mask = np.zeros(shape=(height, width), dtype=np.uint8)
    mask[:, width//4:3*width//4+1] = 255

    with TempFile('.xcf') as xcf, TempFile('.npz') as npz:
        # create gimp file
        gimp_file = GimpFile(xcf) \
            .create('Background', bg) \
            .add_layer_from_numpy('Foreground', fg) \
            .add_layer_from_numpy('Mask', mask)

        # save layer data to numpy arrays
```

(continues on next page)

(continued from previous page)

```
        arr_bg = gimp_file.layer_to_numpy('Background')
        arr_fg = gimp_file.layer_to_numpy('Foreground')
        arr_mask = gimp_file.layer_to_numpy('Mask')

        # save data as npz
        np.savez_compressed(npz, bg=arr_bg, fg=arr_fg, mask=arr_mask)

        # load data from npz
        loaded = np.load(npz)
        loaded_bg = loaded['bg']
        loaded_fg = loaded['fg']
        loaded_mask = loaded['mask']

    # merge background and foreground using mask
    mask_idxs = loaded_mask == 255
    img = loaded_bg.copy()
    img[mask_idxs] = loaded_fg[mask_idxs]

    with TempFile('.xcf') as xcf:
        # create a temporary gimp file and export to png
        gimp_file = GimpFile(xcf) \
            .create('Background', img) \
            .export(png_file)
```

Result:

2.2 API Documentation

2.2.1 Exception Handling

class pgimp.GimpException.**GimpException**

Bases: `Exception`

When pgimp encounters a gimp related exception, it will automatically map it onto `GimpException` or a subtype so that you can easily handle gimp related errors.

Example:

```
>>> from pgimp.GimpScriptRunner import GimpScriptRunner
>>> try:
...     GimpScriptRunner().execute('1/0')
... except Exception as e:
...     str(e).split('\n')[-2]
'ZeroDivisionError: integer division or modulo by zero'
```

class pgimp.GimpScriptRunner.**GimpNotInstalledException**
Bases: *pgimp.GimpException.GimpException*

Indicates that gimp needs to be installed on the system in order for the software to work.

class pgimp.GimpScriptRunner.**GimpScriptException**
Bases: *pgimp.GimpException.GimpException*

Indicates a general error that occurred while trying to execute the script.

class pgimp.GimpScriptRunner.**GimpScriptExecutionTimeoutException**
Bases: *pgimp.GimpException.GimpException*

Thrown when the script execution time exceeds the specified timeout.

class pgimp.GimpScriptRunner.**GimpUnsupportedOSException**
Bases: *pgimp.GimpException.GimpException*

Indicates that your operating system is not supported.

2.2.2 Interacting with Gimp

Running Scripts

class pgimp.GimpScriptRunner.**GimpScriptRunner**(*environment: Dict[str, str] = None*, *working_directory='/home/docs/checkouts/readthedocs.org/user_builds*

Bases: object

Executes python2 scripts within gimp's python interpreter and is used to create higher-level functionality and abstractions that can be used in python3.

When the virtual framebuffer xvfb is installed, it will be automatically used and no other windowing system is required. This is important for batch jobs on machines that do not provide a graphical user interface.

Example:

```
>>> from pgimp.GimpScriptRunner import GimpScriptRunner
>>> GimpScriptRunner().execute('print("Hello from within gimp")')
'Hello from within gimp\n'
```

execute(*string: str*, *parameters: Dict[str, Union[int, float, str, bytes, list, tuple, dict]] = None*, *timeout_in_seconds: float = None*) → Optional[str]
Execute a given piece of code within gimp's python interpreter.

Example:

```
>>> from pgimp.GimpScriptRunner import GimpScriptRunner
>>> GimpScriptRunner().execute('print("Hello from within gimp")')
'Hello from within gimp\n'
```

Parameters

- **string** – The code to be executed as string.
- **parameters** – Parameter names and values. Supported types will be encoded as string, be passed to the script and be decoded there.
- **timeout_in_seconds** – How long to wait for completion in seconds until a *GimpScriptExecutionTimeoutException* is thrown.

Returns The output produced by the script if no output stream is defined.

execute_and_parse_bool(*string: str*, *parameters: dict = None*, *timeout_in_seconds: float = None*) → bool

Execute a given piece of code within gimp's python interpreter and decode the result to bool.

Example:

```
>>> from pgimp.GimpScriptRunner import GimpScriptRunner
>>> GimpScriptRunner().execute_and_parse_bool(
...     'from pgimp.gimp.parameter import return_bool; return_bool("truthy")'
... )
True
```

See also *execute()*.

execute_and_parse_json(*string: str*, *parameters: dict = None*, *timeout_in_seconds: float = None*) → Union[None, int, float, str, list, dict]

Execute a given piece of code within gimp's python interpreter and decode the result to json.

Example:

```
>>> from pgimp.GimpScriptRunner import GimpScriptRunner
>>> GimpScriptRunner().execute_and_parse_json(
...     'from pgimp.gimp.parameter import return_json; return_json({"a": "b",
↪"c": [1, 2]})'
... )['c']
[1, 2]
```

See also *execute()*.

execute_binary(*string: str*, *parameters: dict = None*, *timeout_in_seconds: float = None*) → bytes

Execute a given piece of code within gimp's python interpreter and decode the result to bytes.

Example:

```
>>> import numpy as np
>>> from pgimp.GimpScriptRunner import GimpScriptRunner
>>> print(
...     np.frombuffer(
...         GimpScriptRunner().execute_binary(
...             "from pgimp.gimp.parameter import *; import sys; sys.stdout.
↪write(get_bytes('arr'))",
...             parameters={"arr": np.array([i for i in range(0, 3)],␣
↪dtype=np.uint8).tobytes()}),
...             dtype=np.uint8
...     )
... )
[0 1 2]
```

See also *execute()*.

Returns Raw bytes to be decoded to your target type.

execute_file(*file: str*, ***, *parameters: dict = None*, *timeout_in_seconds: float = None*) → Optional[str]

Execute a script from a file within gimp's python interpreter.

Example:

```
>>> from pgimp.GimpScriptRunner import GimpScriptRunner
>>> from pgimp.util.file import relative_to
```

(continues on next page)

(continued from previous page)

```
>>> GimpScriptRunner().execute_file(relative_to(__file__, 'test-resources/
↪hello.py'))
'Hello from within gimp\n'
```

See also *execute()*.

Gimp Files

Use `GimpFile` to interact with a single file and `GimpFileCollection` to interact with a collection of files.

CHAPTER 3

Indices and tables

- genindex

Index

E

G