
photoshop-connection Documentation

Release 0.2.0

Kota Yamaguchi

May 23, 2023

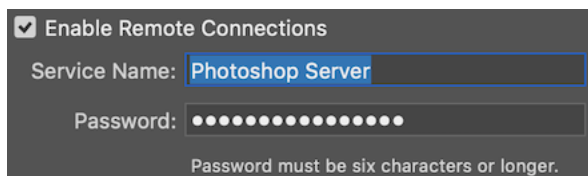
1	Overview	3
2	photoshop	5
3	photoshop.protocol	17
4	photoshop.crypto	19
5	Indices and tables	21
	Python Module Index	23
	Index	25

Python package to remotely execute [ExtendScript](#) in Adobe Photoshop.

1.1 Prerequisites

Photoshop must be configured to accept remote connection.

Open the plug-ins dialog from the *Preferences > Plug-ins...* menu in Photoshop, and check *Enable Remote Connections* option. Enter password to the given field, and click *OK* button and restart Photoshop.



Photoshop must be launched and running for the package to work.

1.2 Usage

Create a session with `photoshop.PhotoshopConnection`, and use one of the API method to work on a document.

Open a file, get the thumbnail image, then close the file:

```
from photoshop import PhotoshopConnection

with PhotoshopConnection(password='secret') as conn:
    conn.execute('open(File("/server/path/to/example.psd"))')
    jpeg_binary = conn.get_document_thumbnail()
    conn.execute('activeDocument.close()')
```

Upload a local PSD file to the server, edit, then download:

```
with PhotoshopConnection(PASSWORD) as conn:
    with open('input.psd', 'rb') as f:
        tmpfile = conn.upload(f.read(), suffix='.psd')
        conn.execute('''
            open(File("%s"));
            activeDocument.activeLayer.name = "edited";
            activeDocument.save();
            activeDocument.close();
            ''' % tmpfile)
    with open('output.psd', 'wb') as f:
        f.write(conn.download(tmpfile).get('data'))
    # Don't forget to remove the temp file.
    conn.execute('File("%s").remove()' % tmpfile)
```


2.1 PhotoshopConnection

```
class photoshop.PhotoshopConnection (password: Optional[str] = None, host: str = 'localhost',
                                       port: int = 49494, validator: Optional[Callable[[str],
                                       None]] = None)
```

Photoshop session.

Parameters

- **password** – Password for the connection, configured in Photoshop. If *None*, try to get password from *PHOTOSHOP_PASSWORD* environment variable.
- **host** – IP address of Photoshop host, default *localhost*.
- **port** – Connection port default to 49494.
- **validator** – Validate function for ECMAscript.

Example:

```
from esprima import parseScript
with PhotoshopConnection(validator=parseScript) as c:
    c.execute('bad_script +') # Raises an Error
```

Raises ConnectionRefusedError – if failed to connect to Photoshop.

Example:

```
from photoshop import PhotoshopConnection

with PhotoshopConnection(password='secret', host='192.168.0.1') as conn:
    conn.execute('alert("hi");')
```

close() → *None*
Close the session.

download (*path: str, file_type: Optional[str] = None, **kwargs*) → Dict[str, Any]

Download the specified document. The file type must be in the format supported by Photoshop.

Parameters

- **path** – file path on the server.
- **file_type** – file type, see `open_document()`.

Returns *dict*. See return type of `get_document_stream()`

execute (*script: str, receive_output: bool = False, timeout: Optional[float] = None*) → Dict[str, Any]

Execute the given ExtendScript in Photoshop.

Parameters

- **script** – ExtendScript to execute in Photoshop.
- **receive_output** – Indicates extra return value is returned from Photoshop.
- **timeout** – Timeout in seconds to wait for response.

Returns *dict*. See `receive()`.

Raises **RuntimeError** – if error happens in remote.

get_document_info (*version: Optional[str] = None, document: Optional[str] = None, placed_ids: Optional[Sequence[str]] = None, layer: Union[int, Tuple[int, int], None] = None, expand_smart_objects: bool = False, get_text_styles: bool = False, get_full_text_styles: bool = False, get_default_layer_effect: bool = False, get_comp_layer_settings: bool = False, get_path_data: bool = False, image_info: Optional[bool] = None, comp_info: Optional[bool] = None, layer_info: bool = True, include_ancestors: bool = True*) → Dict[str, Any]

Return complete document info in JSON format.

Parameters

- **version** – optional requested version (you always get the current version back, but this does a sanity check, and errors on an incompatible version). Example: '1.4.0'.
- **document** – optional document id, uses active doc if not specified.
- **placed_ids** – Photoshop 16.1 and later, optional. reference smart object(s) within the document series of "ID" from layer:smartObject:{} or "placedID" from "image:placed:{}".
- **layer** – *None* for all layers in photoshop, or specify one of the following: - integer ID of a single layer, e.g. 0. - (first, last) tuple of layer IDs, e.g., (1, 6). - 'selected' for currently selected layers.
- **expand_smart_objects** – default is false, recursively get doc info for any smart objects. can be slow.
- **get_text_styles** – default is false, return more detailed text info. can be slow.
- **get_full_text_styles** – default is false, return all text information (getTextStyles must also be true).
- **get_default_layer_effect** – default is false, return all layer fx even if they are disabled.
- **get_comp_layer_settings** – default is false, enumerate layer settings in layer comps.
- **get_path_data** – default is false, return path control points for shapes.

- **image_info** – return image-wide info (size, resolution etc.), default is *layer* != ‘selected’.
- **comp_info** – return comp info in “comps” array, default is true, default is *layer* != ‘selected’.
- **layer_info** – return layer info in “layers” array, default is true.
- **include_ancestors** – 16.1 and later, include surrounding layer groups if doing selected layers/range/single layer id. default is true. should only be used with single layers (otherwise grouping may not be accurate).

Returns *dict*.

Raises RuntimeError – if error happens in remote.

```
get_document_stream(document: Optional[str] = None, placed_ids: Optional[Sequence[str]] =
                    None, placed_id: Optional[str] = None, layer: Union[int, Tuple[int, int],
                    None] = None, position: Optional[int] = None, size: Optional[int] = None,
                    path_only: Optional[bool] = None) → Dict[str, Any]
```

Get the file info and file stream for a smart object.

Parameters

- **document** – optional document id, uses active doc if not specified.
- **placed_ids** – Photoshop 16.1 and later, optional. reference smart object(s) within the document series of “ID” from layer:smartObject:{} or “placedID” from “image:placed:[{}]”.
- **placed_id** – return file for smart object with this placed id (“ID” from layer:smartObject:{} or “placedID” from “image:placed:[{}]”).
- **layer** – when integer ID of a single layer is specified, e.g. 0, return file for smart object with this layer id. When *placed_id* is *None* and *layer* is also *None*, return placed smart object stream the selected layers
- **position** – offset into file (defaults to 0).
- **size** – number of bytes to return (defaults to all bytes).
- **path_only** – instead of returning the file stream back over the wire, write it to a file local to the server, and return the path as a string argument in the JSON part of the FileStream Reply.

Returns

dict with the following fields:

- *mimeFormat*: mime string.
- *position*: position of file data returned.
- *size*: number of file bytes returned.
- *fullSize*: total number of bytes in file.
- *path*: string, server-local path to file if path was set to true in the request).
- *data*: actual data in bytes. if *path* is True, this is empty.

Raises RuntimeError – if error happens in remote.

Note: The maximum size returned by PS is 2 GB, if you have a smart object bigger than 2 GB, you need to use the position/size format. To return chunks, or the path format to write it to a temp file. Document stream/attributes are returned as a FileStream Reply.

get_document_thumbnail (*document: Optional[str] = None, max_width: int = 2048, max_height: int = 2048, format: int = 1, placed_ids: Optional[Sequence[str]] = None*) → Union[bytes, photoshop.protocol.Pixmap]

Send a thumbnail of a document's composite.

Parameters

- **document** – optional document id, uses active doc if not specified.
- **max_width** – maximum width of thumbnail.
- **max_height** – maximum height of thumbnail.
- **format** – 1 is JPEG, 2 is pixmap (uncompressed w/ transparency).
- **placed_ids** – Photoshop 16.1 and later, optional. reference smart object(s) within the document series of “ID” from layer:smartObject:{} or “placedID” from “image:placed:[]”.

Returns JPEG bytes if *format* is 1, or *Pixmap* if *format* is 2.

Raises RuntimeError – if error happens in remote.

get_layer_shape (*document: Optional[str] = None, layer: Union[int, Tuple[int, int], None] = None, version: str = '1.0.0', placed_ids: Optional[Sequence[str]] = None*) → Optional[Dict[str, Any]]

Return path/fill/strokeStyle for a shape layer(s).

Parameters

- **document** – optional document id, uses active doc if not specified.
- **placed_ids** – Photoshop 16.1 and later, optional. reference smart object(s) within the document series of “ID” from layer:smartObject:{} or “placedID” from “image:placed:[]”.
- **layer** – *None* for currently selected layers in photoshop, or specify one of the following:
 - integer ID of a single layer, e.g. 0. - (*first, last*) tuple of layer IDs, e.g., (1, 6).
- **version** – format version. Valid versions are 1.0.0 in 14.1, and 1.0, 1.0.0, 1.1, or 1.1.0 in Photoshop 14.2

Returns *dict* of the following schema, or *None* if no valid layer is specified.

Schema:

```
{ "path":
  { "pathComponents": // arrays of paths to be filled and boolean operators
    [ { "shapeOperation": ("intersect"/"add"/"subtract"/"xor")
      "subpathListKey": [ //list of subpath objects that make up the_
↪ component
        { "closedSubpath": true, // (if subpath is closed)
          "points": [ { // array of knot objects (anchor and control_
↪ points)
            anchor: [x, y] //point on path
            forward: [x1, y1] //forward bezier control
            backward: [x2, y2] //backward bezier control
```

(continues on next page)

(continued from previous page)

```

        }, //next knot...
        ...]
        "origin":{"origin": ("ellipse"/"rect"/"roundedrect"/"line"/"unknown")
        "radii": [r1,r2,r3,r4], //radii for rounded rect if any
        "bounds":["top":top,"left":left,"right":right,"bottom":bottom], //bounds_
↪of entire path
        "defaultFill":true/false}, //whether path starts out filled or not
"fill":
    {"color":{"red":red,"green":green,"blue":blue},"class":"solidColorLayer"}
    //or
    {"gradient":{"gradient object"},"class":"gradientLayer"}
    //or
    {"pattern":{"pattern object"},"class":"patternLayer"}
"strokeStyle":
    {(strokeStyle object)}
}

```

Example:

```

{"path":{"pathComponents":
    [{"shapeOperation":"add",
      "subpathListKey":[
        {"closedSubpath":true,
          "points": [{"anchor": [234.5,36], "forward": [307.125,36], "backward
↪": [161.875,36]},
          {"anchor": [366,167], "forward": [366,239.349], "backward": [366,
↪94.651]},
          {"anchor": [234.5,298], "forward": [161.875,298], "backward": [307.
↪125,298]},
          {"anchor": [103,167], "forward": [103,94.651], "backward": [103,
↪239.349]}]}],
      "origin":{"origin":"ellipse","bounds": [35,102,299,367]}
    ]},
    "bounds": [35,102,299,367],
    "defaultFill": false,
    "fill": {"color": {"red": 0, "green": 0, "blue": 0}, "class": "solidColorLayer"}
}

```

Raises RuntimeError – if error happens in remote.

get_layer_thumbnail (*document: Optional[str] = None, max_width: int = 2048, max_height: int = 2048, convert_rgb_profile: bool = True, icc_profile: Optional[str] = None, interpolation: Optional[str] = None, transform: Optional[Dict[str, Any]] = None, layer: Union[int, Tuple[int, int], None] = None, layer_settings: Optional[Sequence[Dict[str, Any]]] = None, image_settings: Optional[Sequence[Dict[str, Any]]] = None, include_layers: Optional[Dict[str, Any]] = None, clip_bounds: Union[bool, Tuple[int, ...], None] = None, bounds: Optional[bool] = False, bounds_only: Optional[bool] = False, thread: Optional[bool] = None, layer_comp_id: Optional[str] = None, layer_comp_index: Optional[int] = None, dither: bool = True, color_dither: bool = True*) → Optional[photoshop.protocol.Pixmap]

Send a thumbnail of layer composite, or a range of layers, with optional settings/transform applied.

Parameters

- **document** – optional document id, uses active doc if not specified.
- **max_width** – maximum width of thumbnail.
- **max_height** – maximum height of thumbnail.
- **placed_ids** – Photoshop 16.1 and later, optional. reference smart object(s) within the document series of “ID” from layer:smartObject:{} or “placedID” from “image:placed:{{}}”.
- **convert_rgb_profile** – if True, the thumbnail is converted to the working RGB space in “Color Settings...”.
- **icc_profile** – optional, Photoshop 16.1, and later. convert to profile with this name, e.g. srgb is “sRGB IEC61966-2.1”
- **interpolation** – interpolation method to use for any downscaling necessary to fit into requested “width”/“height”. supported interpolation types (from image size dialog/action):
 - “nearestNeighbor”
 - “bilinear”
 - “bicubic”
 - “bicubicSmoother”
 - “bicubicSharper”
 - “bicubicAutomatic”
 - “preserveDetailsUpscale”
 - “automaticInterpolation”default is “bicubicSharper”.
- **transform** – scale/transform layers by this before building thumbnails (scales original source data, such as smart obj/vectors). if this is specified, the thumbnail is built on a worker thread in Photoshop.

Example:

```
transform = {  
    'scale_x': 100.0,  
    'scale_y': 100.0,  
    'interpolation': 'bicubicSharper',  
    'dumb_scaling': True  
}
```

- *scale_x*: percent, 100.0 == 1x
 - *scale_y*: percent, 100.0 == 1x
 - *interpolation*: Optional, similar to interpolation above, but this is just used for the transform step (not the thumbnail), it defaults to Photoshop’s “Image Interpolation” preference.
 - *dumb_scaling*: For PS >= 14.2. Make smart shapes scale like non-smart shapes (round rect corners will scale), default is False.
- **layer** – *None* for currently selected layers in photoshop, or specify one of the following:
 - integer ID of a single layer, e.g. 0. - (*first*, *last*) tuple of layer IDs, e.g., (1, 6).
 - **document** – optional document id, uses active doc if not specified

- **layer_settings** – Action list to modify the layer before the thumbnail is retrieved. This option is available when *layer* param is specified by tuple range. The argument should be list of dict with the following keys:
 - *enabled*: make the layer visible/invisible.
 - *blendOptions*: blending settings to use.
 - *layerEffects*: fx settings to use.
 - *offset*: integer offset of layer in dict.
 - *vectorMask*: vector mask to apply in dict.
 - *FXRefPoint*: effect reference point.

Example:

```
[
  {
    'enabled': True,
    'blendOptions': [],
    'layerEffects': [],
    'offset': {
      'horizontal': 0,
      'vertical': 0
    },
    'vectorMask': {
      'enabled': False,
      'offset': {
      }
      'invert': False,
    },
    'FXRefPoint': {
      'horizontal': 0,
      'vertical': 0
    }
  }
]
```

- **image_settings** –
- **include_layers** – include additional layers to the requested layer. dict with one or more of the following keys.
 - *adjustors*: adjustors above the layer, default is *visible*.
 - *ancestors*: enclosing groups (includes group blending, fx, masks), default is *all*. *visible* and *all* incorporate any blending parameters/masks of the ancestor groups. *visible* returns an empty thumbnail for any layer inside an invisible group. *none* substitutes default groups for any groups around the layer.
 - *children*: if layer is a group (includes group blending, fx, masks), default is *visible*.
 - *clipbase*: clip base if layer is clipped. The clip base is a layer that a clipped layer is clipped to, default is *all*.
 - *clipped*: clipped layers if layer is clip base, default is *visible*.

Values are one of ‘*all*’, ‘*none*’, or ‘*visible*’.

 - *all*: include all layers of this type (force them visible).
 - *none*: include no layers of this type.

- *visible*: include visible layers of this type.

Example:

```
{
    'adjustors': 'none',
    'children': 'all',
}
```

- **clip_bounds** – clip the layer thumbnail to the document canvas bounds if specified. Can specify *True* to bound to document size, or specify tuple of (*top*, *left*, *right*, *bottom*).
- **bounds** – return the thumbnail bounds as JSON on same transaction. (default is *False*).
- **bounds_only** – Just return the thumbnail bounds as JSON on same transaction. (no thumbnail data) (default is *false*).
- **thread** – build the thumbnail on a thread. By default, the thumbnail is threaded if there is a “transform”, otherwise it is done on the main thread unless a user event occurs, then it is cancelled, and restarted on a thread *thread* can be used to override the default (either force the thumb to be started on the main thread or a background thread) it may help performance if you know that the thumbnail is either quick (best done on main thread) or slow (best done on background) there is a slight memory/performance penalty for threading in that the layer data must be copied before it is threaded.
- **layer_comp_id** – layer comp id to use (this comp is temporarily applied before getting thumbnail).
- **layer_comp_index** – layer comp index to use (this comp is temporarily applied before getting thumbnail).
- **dither** – 15.0 and later. If 1) *dither* is true 2) and either *color_dither* is false, or *dither* is checked in the global color settings (Color Settings... in Photoshop) 3) and any color/depth conversion would be “lossy” (16 to 8 bit, CMYK to RGB, etc), then dithering will occur, otherwise there will be no dithering.
- **color_dither** – see above.

Returns *Pixmap* or *None*.

Raises **RuntimeError** – if error happens in remote.

Note: “interpolation”, “transform”, “bounds”, “boundsOnly”, and “thread” are supported in background-only (layer-less) documents but only in version 15.0 and later. “layerID” should be 0 in that case. The other layer-related settings are ignored as there are no layers.

Warning: if *layer* tuple range includes a group layer, it must include the corresponding hidden “divider” layer at the bottom of the group (and vice-versa). The range can also just include layers inside a group with no group layers at all.

open_document (*path*: str, *file_type*: Optional[str] = None, *smart_object*: bool = False) → Dict[str, Any]
Open the specified document.

Parameters

- **path** – file path on the server.

- **file_type** – file type. default is *None*. This must be one of the following:

- 'ALIASPIX'
- 'BMP'
- 'CAMERARAW'
- 'COMPUSERVEGIF'
- 'DICOM'
- 'ELECTRICIMAGE'
- 'EPS'
- 'EPSPICTPREVIEW'
- 'EPSTIFFPREVIEW'
- 'FILMSTRIP'
- 'JPEG'
- 'PCX'
- 'PDF'
- 'PHOTOCDF'
- 'PHOTOSHOP'
- 'PHOTOSHOPDCS_1'
- 'PHOTOSHOPDCS_2'
- 'PHOTOSHOPEPS'
- 'PHOTOSHOPPDF'
- 'PICTFILEFORMAT'
- 'PICTRESOURCEFORMAT'
- 'PIXAR'
- 'PNG'
- 'PORTABLEBITMAP'
- 'RAW'
- 'SCITEXCT'
- 'SGIRGB'
- 'SOFTIMAGE'
- 'TARGA'
- 'TIFF'
- 'WAVEFRONTSLA'
- 'WIRELESSBITMAP'

- **smart_object** – open as a smart object.

Returns *dict* of response.

ping (*timeout: float = 10.0*) → None
Send keep alive signal to Photoshop.

Parameters **timeout** – Timeout in seconds to wait for response.

Raises **RuntimeError** – if error happens in remote.

subscribe (*event: str, callback: Callable[[photoshop.photoshop_connection.PhotoshopConnection, Optional[bytes]], bool], block: bool = False, **kwargs*) → None
Subscribe to changes, sends any relevant change info back on subscribing socket.

Parameters

- **event** – Event name, one of *Event*.
- **callback** – Callable that takes two arguments:
 - *conn*: *PhotoshopConnection* instance.
 - *data*: *bytes* data returned from Photoshop on this event. The actual data format varies by event type.Return value of *callback* signals termination of the current subscription. If *callback* returns True, subscription stops.
- **block** – Block until subscription finishes. default *False*.

Example:

```
import json
import time

def handler(conn, data):
    print(json.loads(data.decode('utf-8')))
    return True # This terminates subscription

with PhotoshopConnection() as conn:
    conn.subscribe('imageChanged', handler)
    conn.execute('documents.add()')
    time.sleep(5)
```

upload (*data: bytes, suffix: Optional[str] = None*) → str
Upload arbitrary data to Photoshop, and returns the file path where the data is saved.

Parameters

- **data** – *bytes* to send.
- **suffix** – suffix to append to the temporary file name.

Returns Temporary server-side file path in *str*.

Raises **RuntimeError** – if error happens in remote.

Example:

```
with open('/path/to/example.psd', 'rb') as f:
    filepath = conn.upload(f.read(), suffix='.psd')
conn.open_document(filepath)
```

2.2 Event

class photoshop.Event

List of events in *subscribe()*.

See [Kevlar API](#).

Asrt = 'Asrt'

activeViewChanged = 'activeViewChanged'

backgroundColorChanged = 'backgroundColorChanged'

closedDocument = 'closedDocument'

colorSettingsChanged = 'colorSettingsChanged'

currentDocumentChanged = 'currentDocumentChanged'

documentChanged = 'documentChanged'

foregroundColorChanged = 'foregroundColorChanged'

generatorDocActivated = 'generatorDocActivated'

generatorMenuChanged = 'generatorMenuChanged'

idle = 'idle'

imageChanged = 'imageChanged'

keyboardShortcutsChanged = 'keyboardShortcutsChanged'

newDocumentViewCreated = 'newDocumentViewCreated'

quickMaskStateChanged = 'quickMaskStateChanged'

toolChanged = 'toolChanged'

workspaceChanged = 'workspaceChanged'

CHAPTER 3

photoshop.protocol

```
class photoshop.protocol.ContentType
```

Message content type.

```
CANCEL_COMMAND = 8
```

```
DATA = 5
```

```
ERROR_STRING = 1
```

```
EVENT_STATUS = 9
```

```
FILE_STREAM = 7
```

```
ILLEGAL = 0
```

```
IMAGE = 3
```

```
KEEP_ALIVE = 6
```

```
PROFILE = 4
```

```
SCRIPT = 2
```

```
SCRIPT_SHARED = 10
```

```
class photoshop.protocol.Pixmap(width: int, height: int, row_bytes: int, color_mode: int, channels: int, bits: int, data: bytes)
```

Pixmap representing an uncompressed pixels, ARGB, row-major order.

Variables

- **width** – width of the image.
- **height** – height of the image.
- **row_bytes** – bytes per row.
- **color_mode** – color mode of the image.
- **channels** – number of channels.
- **bits** – bits per pixel.

- **data** – raw data bytes.

dump () → bytes

Dump Pixmap to bytes.

classmethod parse (*data: bytes*) → photoshop.protocol.Pixmap

Parse Pixmap from data.

topil () → PIL.Image.Image

Convert to PIL Image.

class photoshop.protocol.**Protocol** (*password: str*)

Photoshop protocol.

VERSION = 1

receive (*socket: socket.socket*) → Dict[str, Any]

Receives data from Photoshop.

Parameters **socket** – socket to receive data.

Returns

dict of the following fields.

- **status**: execution status, 0 when success, otherwise error.
- **protocol**: protocol version, equal to 1.
- **transaction**: transaction id.
- **content_type**: data type. See [ContentType](#).
- **body**: body of the response data, *dict* for IMAGE type, otherwise bytes.

Example:

```
{
  'status': 0,
  'protocol': 1,
  'transaction': 0,
  'content_type': ContentType.SCRIPT,
  'body': b'[ActionDescriptor]'
}
```

Raises **AssertionError** – if response format is invalid.

send (*socket: socket.socket, content_type: photoshop.protocol.ContentType, data: bytes, transaction: int = 0, status: int = 0*) → None

Sends data to Photoshop.

Parameters

- **content_type** – See [ContentType](#).
- **data** – bytes to send.
- **transaction** – transaction id.
- **status** – execution status, should be 0.

CHAPTER 4

photoshop.crypto

```
class photoshop.crypto.EncryptDecrypt (password: bytes, salt: bytes = b'Adobe Photoshop',
                                         iterations: int = 1000, length: int = 24)

    decrypt (token: bytes) → bytes
    encrypt (message: bytes) → bytes
```


CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`

p

photoshop, [5](#)

photoshop.crypto, [19](#)

photoshop.protocol, [17](#)

A

activeViewChanged (*photoshop.Event attribute*), 15
Asrt (*photoshop.Event attribute*), 15

B

backgroundColorChanged (*photoshop.Event attribute*), 15

C

CANCEL_COMMAND (*photoshop.protocol.ContentType attribute*), 17
close() (*photoshop.PhotoshopConnection method*), 5
closedDocument (*photoshop.Event attribute*), 15
colorSettingsChanged (*photoshop.Event attribute*), 15
ContentType (*class in photoshop.protocol*), 17
currentDocumentChanged (*photoshop.Event attribute*), 15

D

DATA (*photoshop.protocol.ContentType attribute*), 17
decrypt() (*photoshop.crypto.EncryptDecrypt method*), 19
documentChanged (*photoshop.Event attribute*), 15
download() (*photoshop.PhotoshopConnection method*), 5
dump() (*photoshop.protocol.Pixmap method*), 18

E

encrypt() (*photoshop.crypto.EncryptDecrypt method*), 19
EncryptDecrypt (*class in photoshop.crypto*), 19
ERROR_STRING (*photoshop.protocol.ContentType attribute*), 17
Event (*class in photoshop*), 15
EVENT_STATUS (*photoshop.protocol.ContentType attribute*), 17
execute() (*photoshop.PhotoshopConnection method*), 6

F

FILE_STREAM (*photoshop.protocol.ContentType attribute*), 17
foregroundColorChanged (*photoshop.Event attribute*), 15

G

generatorDocActivated (*photoshop.Event attribute*), 15
generatorMenuChanged (*photoshop.Event attribute*), 15
get_document_info() (*photoshop.PhotoshopConnection method*), 6
get_document_stream() (*photoshop.PhotoshopConnection method*), 7
get_document_thumbnail() (*photoshop.PhotoshopConnection method*), 8
get_layer_shape() (*photoshop.PhotoshopConnection method*), 8
get_layer_thumbnail() (*photoshop.PhotoshopConnection method*), 9

I

idle (*photoshop.Event attribute*), 15
ILLEGAL (*photoshop.protocol.ContentType attribute*), 17
IMAGE (*photoshop.protocol.ContentType attribute*), 17
imageChanged (*photoshop.Event attribute*), 15

K

KEEP_ALIVE (*photoshop.protocol.ContentType attribute*), 17
keyboardShortcutsChanged (*photoshop.Event attribute*), 15

N

newDocumentViewCreated (*photoshop.Event attribute*), 15

O

`open_document()` (*photoshop.PhotoshopConnection method*), 12

P

`parse()` (*photoshop.protocol.Pixmap class method*), 18

`photoshop` (*module*), 5

`photoshop.crypto` (*module*), 19

`photoshop.protocol` (*module*), 17

`PhotoshopConnection` (*class in photoshop*), 5

`ping()` (*photoshop.PhotoshopConnection method*), 13

`Pixmap` (*class in photoshop.protocol*), 17

`PROFILE` (*photoshop.protocol.ContentType attribute*), 17

`Protocol` (*class in photoshop.protocol*), 18

Q

`quickMaskStateChanged` (*photoshop.Event attribute*), 15

R

`receive()` (*photoshop.protocol.Protocol method*), 18

S

`SCRIPT` (*photoshop.protocol.ContentType attribute*), 17

`SCRIPT_SHARED` (*photoshop.protocol.ContentType attribute*), 17

`send()` (*photoshop.protocol.Protocol method*), 18

`subscribe()` (*photoshop.PhotoshopConnection method*), 14

T

`toolChanged` (*photoshop.Event attribute*), 15

`topil()` (*photoshop.protocol.Pixmap method*), 18

U

`upload()` (*photoshop.PhotoshopConnection method*), 14

V

`VERSION` (*photoshop.protocol.Protocol attribute*), 18

W

`workspaceChanged` (*photoshop.Event attribute*), 15