
PhotoScan Python Reference

Release 1.2.4

Agisoft LLC

March 24, 2016

CONTENTS

1 Overview	3
1.1 Introduction to Python scripting in PhotoScan	3
2 Application Modules	5
3 Python API Change Log	59
3.1 PhotoScan version 1.2.4 build 2399	59
3.2 PhotoScan version 1.2.3 build 2331	59
3.3 PhotoScan version 1.2.2 build 2294	59
3.4 PhotoScan version 1.2.1 build 2278	59
3.5 PhotoScan version 1.2.0 build 2198	60
3.6 PhotoScan version 1.2.0 build 2126	60
3.7 PhotoScan version 1.1.1 build 2009	61
3.8 PhotoScan version 1.1.0 build 2004	61
3.9 PhotoScan version 1.1.0 build 1976	61
3.10 PhotoScan version 1.0.0 build 1795	62
3.11 PhotoScan version 1.0.0 build 1742	62
3.12 PhotoScan version 0.9.1 build 1703	63
3.13 PhotoScan version 0.9.0 build 1586	63
3.14 PhotoScan version 0.8.5 build 1423	64
3.15 PhotoScan version 0.8.4 build 1289	64
3.16 PhotoScan version 0.8.3 build 1212	64
3.17 PhotoScan version 0.8.3 build 1154	64
Python Module Index	65

Copyright (c) 2016 Agisoft LLC.

1.1 Introduction to Python scripting in PhotoScan

This API is in development and will be extended in the future PhotoScan releases.

Note: Python scripting is supported only in PhotoScan Professional edition.

PhotoScan uses Python 3.3 as a scripting engine.

Python commands and scripts can be executed in PhotoScan in one of the following ways:

- From PhotoScan “Console” pane using it as standard Python console
- From the “Tools” menu using “Run script...” command

The following PhotoScan functionality can be accessed from Python scripts:

- Open/save/create PhotoScan projects
- Add/remove chunks, cameras, markers
- Add/modify camera calibrations, ground control data, assign geographic projections and coordinates
- Perform processing steps (align photos, build dense cloud, build mesh, texture, decimate model, etc...)
- Export processing results (models, textures, orthophotos, DEMs)
- Access data of generated models, point clouds, images

APPLICATION MODULES

PhotoScan module provides access to the core processing functionality, including support for inspection and manipulation with project data.

The main component of the module is a Document class, which represents a PhotoScan project. Multiple Document instances can be created simultaneously if needed. Besides that a currently opened project in the application can be accessed using `PhotoScan.app.document` property.

The following example performs main processing steps on existing project and saves back the results:

```
>>> import PhotoScan
>>> doc = PhotoScan.app.document
>>> doc.open("project.psz")
>>> chunk = doc.chunk
>>> chunk.matchPhotos(accuracy=PhotoScan.HighAccuracy, preselection=PhotoScan.GenericPreselection)
>>> chunk.alignCameras()
>>> chunk.buildDenseCloud(quality=PhotoScan.MediumQuality)
>>> chunk.buildModel(surface=PhotoScan.Arbitrary, interpolation=PhotoScan.EnabledInterpolation)
>>> chunk.buildUV(mapping=PhotoScan.GenericMapping)
>>> chunk.buildTexture(blending=PhotoScan.MosaicBlending, size=4096)
>>> doc.save()
```

class `PhotoScan.Accuracy`
Alignment accuracy in [HighestAccuracy, HighAccuracy, MediumAccuracy, LowAccuracy, LowestAccuracy]

class `PhotoScan.Antenna`
GPS antenna position relative to camera.

fixed
Fix antenna flag.

Type `boolean`

location
Antenna coordinates.

Type `Vector`

location_acc
Antenna location accuracy.

Type `Vector`

location_ref
Antenna location reference.

Type `Vector`

rotation

Antenna rotation angles.

Type `Vector`

rotation_acc

Antenna rotation accuracy.

Type `Vector`

rotation_ref

Antenna rotation reference.

Type `Vector`

class `PhotoScan.Application`

Application class provides access to several global application attributes, such as document currently loaded in the user interface, software version and OpenCL device configuration. It also contains helper routines to prompt the user to input various types of parameters, like displaying a file selection dialog or coordinate system selection dialog among others.

An instance of Application object can be accessed using `PhotoScan.app` attribute, so there is usually no need to create additional instances in the user code.

The following example prompts the user to select a new coordinate system, applies it to the active chunk and saves the project under the user selected file name:

```
>>> import PhotoScan
>>> doc = PhotoScan.app.document
>>> crs = PhotoScan.app.getCoordinateSystem("Select Coordinate System", doc.chunk.crs)
>>> doc.chunk.crs = crs
>>> path = PhotoScan.app.getSaveFileName("Save Project As")
>>> if not doc.save(path):
...     PhotoScan.app.messageBox("Can't save project")
```

activated

PhotoScan activation status.

Type `boolean`

addMenuItem (*label*, *func* [, *shortcut*] [, *icon*])

Create a new menu entry.

Parameters

- **label** (*string*) – Menu item label.
- **func** (*function*) – Function to be called.
- **shortcut** (*string*) – Keyboard shortcut.
- **icon** (*string*) – Icon.

addMenuSeparator (*label*)

Add menu separator.

Parameters **label** (*string*) – Menu label.

console

Console pane.

Type `ConsolePane`

cpu_cores_inactive

Number of CPU cores to reserve for GPU tasks during processing. It is recommended to deactivate one CPU core for each GPU in use for optimal performance.

Type int

document

Main application document object.

Type `Document`

enumOpenCLDevices ()

Enumerate installed OpenCL devices.

Returns A list of devices.

Return type list

getCoordinateSystem ([label][, value])

Prompt user for coordinate system.

Parameters

- **label** (*string*) – Optional text label for the dialog.
- **value** (`CoordinateSystem`) – Default value.

Returns Selected coordinate system. If the dialog was cancelled, None is returned.

Return type `CoordinateSystem`

getExistingDirectory ([hint])

Prompt user for the existing folder.

Parameters **hint** (*string*) – Optional text label for the dialog.

Returns Path to the folder selected. If the input was cancelled, empty string is returned.

Return type string

getFloat (label=' ', value=0)

Prompt user for the floating point value.

Parameters

- **label** (*string*) – Optional text label for the dialog.
- **value** (*float*) – Default value.

Returns Floating point value entered by the user.

Return type float

getInt (label=' ', value=0)

Prompt user for the integer value.

Parameters

- **label** (*string*) – Optional text label for the dialog.
- **value** (*int*) – Default value.

Returns Integer value entered by the user.

Return type int

getOpenFileName ([hint])

Prompt user for the existing file.

Parameters **hint** (*string*) – Optional text label for the dialog.

Returns Path to the file selected. If the input was cancelled, empty string is returned.

Return type *string*

getOpenFileNames (*[hint]*)

Prompt user for one or more existing files.

Parameters **hint** (*string*) – Optional text label for the dialog.

Returns List of file paths selected by the user. If the input was cancelled, empty list is returned.

Return type *list*

getSaveFileName (*[hint]*)

Prompt user for the file. The file does not have to exist.

Parameters **hint** (*string*) – Optional text label for the dialog.

Returns Path to the file selected. If the input was cancelled, empty string is returned.

Return type *string*

getString (*label='', value=''*)

Prompt user for the string value.

Parameters

- **label** (*string*) – Optional text label for the dialog.
- **value** (*string*) – Default value.

Returns String entered by the user.

Return type *string*

gpu_mask

GPU device bit mask: 1 - use device, 0 - do not use (i.e. value 5 enables device number 0 and 2).

Type *int*

messageBox (*message*)

Display message box to the user.

Parameters **message** (*string*) – Text message to be displayed.

quit ()

Exit application.

update ()

Update user interface during long operations.

version

PhotoScan version.

Type *string*

viewpoint

Viewpoint in the model view.

Type *Viewpoint*

class `PhotoScan.BlendingMode`

Blending mode in [`AverageBlending`, `MosaicBlending`, `MinBlending`, `MaxBlending`, `DisabledBlending`]

class PhotoScan.**Calibration**

Calibration object contains camera calibration information including image size, focal length, principal point coordinates and distortion coefficients.

b1

Affinity.

Type float**b2**

Non-orthogonality.

Type float**cx**

Principal point X coordinate.

Type float**cy**

Principal point Y coordinate.

Type float**error** (*point*, *proj*)

Returns projection error.

Parameters

- **point** (*Vector*) – Coordinates of the point to be projected.
- **proj** (*Vector*) – Pixel coordinates of the point.

Returns 2D projection error.**Return type** *Vector***f**

Focal length.

Type float**height**

Image height.

Type int**k1**

Radial distortion coefficient K1.

Type float**k2**

Radial distortion coefficient K2.

Type float**k3**

Radial distortion coefficient K3.

Type float**k4**

Radial distortion coefficient K4.

Type float

load (*path*, *format*='xml')

Loads calibration from file.

Parameters

- **path** (*string*) – path to calibration file
- **format** (*string*) – Calibration format in ['xml', 'australis', 'photomodeler', 'calibcam', 'calcam', 'inpho', 'usgs'].

Returns success of operation

Return type boolean

p1

Tangential distortion coefficient P1.

Type float

p2

Tangential distortion coefficient P2.

Type float

p3

Tangential distortion coefficient P3.

Type float

p4

Tangential distortion coefficient P4.

Type float

project (*point*)

Returns projected pixel coordinates of the point.

Parameters **point** (*Vector*) – Coordinates of the point to be projected.

Returns 2D projected point coordinates.

Return type *Vector*

save (*path*, *format*='xml'[, *focal_length*][, *pixel_size*][, *label*])

Saves calibration to file.

Parameters

- **path** (*string*) – path to calibration file
- **format** (*string*) – Calibration format in ['xml', 'australis', 'photomodeler', 'calibcam', 'calcam', 'inpho', 'usgs'].
- **focal_length** (*float*) – Focal length in mm used to convert normalized calibration coefficients to PhotoModeler and CalCam coefficients.
- **pixel_size** (*Vector*) – Pixel size in mm used to convert normalized calibration coefficients to Australis and CalibCam coefficients.
- **label** (*string*) – Calibration label used in Australis, CalibCam and CalCam formats.

Returns success of operation

Return type boolean

type

Camera model.

Type `Sensor.Type`

unproject (*point*)

Returns direction corresponding to the image point.

Parameters **point** (`Vector`) – Pixel coordinates of the point.

Returns 3D vector in the camera coordinate system.

Return type `Vector`

width

Image width.

Type `int`

class `PhotoScan.Camera`

Camera instance

```
>>> import PhotoScan
>>> chunk = PhotoScan.app.document.addChunk()
>>> chunk.addPhotos(["IMG_0001.jpg", "IMG_0002.jpg"])
>>> camera = chunk.cameras[0]
>>> camera.photo.meta["Exif/FocalLength"]
'18'
```

center

Camera station coordinates for the photo in the chunk coordinate system.

Type `Vector`

enabled

Enables/disables the photo.

Type `boolean`

error (*point*, *proj*)

Returns projection error.

Parameters

- **point** (`Vector`) – Coordinates of the point to be projected.
- **proj** (`Vector`) – Pixel coordinates of the point.

Returns 2D projection error.

Return type `Vector`

frames

Camera frames.

Type list of `Camera`

group

Camera group.

Type `CameraGroup`

key

Camera identifier.

Type `int`

label

Camera label.

Type string

mask

Camera mask.

Type `Mask`

meta

Camera meta data.

Type `MetaData`

open (*path* [, *layer*])

Loads specified image file.

Parameters

- **path** (*string*) – Path to the image file to be loaded.
- **layer** (*int*) – Optional layer index in case of multipage files.

Returns Success of operation.

Return type boolean

orientation

Image orientation (1 - normal, 6 - 90 degree, 3 - 180 degree, 8 - 270 degree).

Type int

photo

Camera photo.

Type `Photo`

planes

Camera planes.

Type list of `Camera`

project (*point*)

Returns coordinates of the point projection on the photo.

Parameters **point** (`Vector`) – Coordinates of the point to be projected.

Returns 2D point coordinates.

Return type tuple of 2 floats

reference

Camera reference data.

Type `CameraReference`

selected

Selects/deselects the photo.

Type boolean

sensor

Camera sensor.

Type `Sensor`

thumbnail

Camera thumbnail.

Type `Thumbnail`

transform

4x4 matrix describing photo location in the chunk coordinate system.

Type `Matrix`

class `PhotoScan.CameraGroup`

`CameraGroup` objects define groups of multiple cameras. The grouping is established by assignment of a `CameraGroup` instance to the `Camera.group` attribute of participating cameras.

The `type` attribute of `CameraGroup` instances defines the effect of such grouping on processing results and can be set to `Folder` (no effect) or `Station` (coincident projection centers).

class `Type`

Camera group type in [`Folder`, `Station`]

`CameraGroup.label`

Camera group label.

Type `string`

`CameraGroup.selected`

Current selection state.

Type `boolean`

`CameraGroup.type`

Camera group type.

Type `CameraGroup.Type`

class `PhotoScan.CameraReference`

`CameraReference` object contains measured camera location data.

accuracy

Camera location accuracy.

Type `Vector`

accuracy_ypr

Camera rotation accuracy.

Type `Vector`

enabled

Enabled flag.

Type `boolean`

location

Camera coordinates.

Type `Vector`

rotation

Camera rotation angles.

Type `Vector`

class `PhotoScan.Chunk`

A `Chunk` object:

- provides access to all chunk components (sensors, cameras, camera groups, markers, scalebars)
- contains data inherent to individual frames (point cloud, model, etc)
- implements processing methods (`matchPhotos`, `alignCameras`, `buildDenseCloud`, `buildModel`, etc)

- provides access to other chunk attributes (transformation matrix, coordinate system, meta-data, etc..)

New components can be created using corresponding addXXX methods (addSensor, addCamera, addCameraGroup, addMarker, addScalebar, addFrame). Removal of components is supported by a single remove method, which can accept lists of various component types.

In case of multi-frame chunks the Chunk object contains an additional reference to the particular chunk frame, initialized to the current frame by default. Various methods that work on a per frame basis (matchPhotos, buildModel, etc) are applied to this particular frame. A frames attribute can be used to obtain a list of Chunk objects that reference all available frames.

The following example performs image matching and alignment for the active chunk:

```
>>> import PhotoScan
>>> chunk = PhotoScan.app.document.chunk
>>> for frame in chunk.frames:
...     frame.matchPhotos(accuracy=PhotoScan.HighAccuracy)
>>> chunk.alignCameras()
```

accuracy_cameras

Expected accuracy of camera coordinates in meters.

Type `Vector`

accuracy_cameras_ypr

Expected accuracy of camera orientation angles in degrees.

Type `Vector`

accuracy_markers

Expected accuracy of marker coordinates in meters.

Type `Vector`

accuracy_projections

Expected accuracy of marker projections in pixels.

Type `float`

accuracy_tiepoints

Expected tie point accuracy in pixels.

Type `float`

addCamera ()

Add new camera to the chunk.

Returns Created camera.

Return type `Camera`

addCameraGroup ()

Add new camera group to the chunk.

Returns Created camera group.

Return type `CameraGroup`

addFrame ()

Add new frame to the chunk.

Returns Created frame.

Return type `Frame`

addMarker ()

Add new marker to the chunk.

Returns Created marker.

Return type `Marker`

addPhotos (*filenames* [, *progress*])

Add a list of photos to the chunk.

Parameters

- **filenames** (*list of string*) – A list of file paths.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type `boolean`

addScalebar (*point1*, *point2*)

Add new scalebar to the chunk.

Parameters

- **point1** (`Marker` or `Camera`) – First endpoint.
- **point2** – Second endpoint.

Returns Created scalebar.

Return type `Scalebar`

addSensor ()

Add new sensor to the chunk.

Returns Created sensor.

Return type `Sensor`

alignCameras ([*cameras*] [, *min_image*] [, *progress*])

Perform photo alignment for the chunk.

Parameters

- **cameras** (*list of Camera*) – A list of cameras to be aligned to the existing cameras.
- **min_image** (*int*) – Minimum number of point projections.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type `boolean`

buildContours (*source_data*=*'elevation'*, *interval*=1 [, *min_value*] [, *max_value*] [, *progress*])

Build contours for the chunk.

Parameters

- **source_data** (*string*) – Source data for contour generation in [*'elevation'*, *'orthomosaic'*].
- **interval** (*float*) – Contour interval.
- **min_value** (*float*) – Minimum value of contour range.
- **max_value** (*float*) – Maximum value of contour range.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

buildDem (*source=DenseCloudData*, *interpolation=EnabledInterpolation* [, *projection*] [, *region*] [, *classes*] [, *progress*])
Build elevation model for the chunk.

Parameters

- **source** (*PhotoScan.DataSource*) – Selects between dense point cloud and sparse point cloud. If not specified, uses dense cloud if available.
- **interpolation** (*PhotoScan.Interpolation*) – Interpolation mode.
- **projection** (*Matrix* or *CoordinateSystem*) – Sets output projection.
- **region** (*tuple of 4 floats*) – Region to be exported in the (x0, y0, x1, y1) format.
- **classes** (*list of int*) – List of dense point classes to be used for surface extraction.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

buildDenseCloud (*quality=MediumQuality*, *filter=AggressiveFiltering* [, *cameras*], *keep_depth=False*, *reuse_depth=False* [, *progress*])
Generate depth maps for the chunk.

Parameters

- **quality** (*PhotoScan.Quality*) – Depth map quality.
- **filter** (*PhotoScan.FilterMode*) – Depth map filtering level.
- **cameras** (list of *Camera*) – A list of cameras to be processed.
- **keep_depth** (*boolean*) – Enables keep depth maps option.
- **reuse_depth** (*boolean*) – Enables reuse depth maps option.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

buildModel (*surface=Arbitrary*, *interpolation=EnabledInterpolation*, *face_count=MediumFaceCount* [, *source*] [, *classes*] [, *progress*])
Generate model for the chunk frame.

Parameters

- **surface** (*PhotoScan.SurfaceType*) – Type of object to be reconstructed.
- **interpolation** (*PhotoScan.Interpolation*) – Interpolation mode.
- **face_count** (*PhotoScan.FaceCount* or *int*) – Target face count.
- **source** (*PhotoScan.DataSource*) – Selects between dense point cloud and sparse point cloud. If not specified, uses dense cloud if available.
- **classes** (*list of int*) – List of dense point classes to be used for surface extraction.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

buildOrthomosaic (*surface*=ElevationData, *blending*=MosaicBlending, *color_correction*=False[, *projection*][[, *region*][[, *dx*][[, *dy*][[, *progress*]])
Build orthomosaic for the chunk.

Parameters

- **surface** (PhotoScan.DataSource) – Orthorectification surface.
- **blending** (PhotoScan.BlendingMode) – Orthophoto blending mode.
- **color_correction** (boolean) – Enables color correction.
- **projection** (Matrix or CoordinateSystem) – Sets output projection.
- **region** (tuple of 4 floats) – Region to be exported in the (x0, y0, x1, y1) format.
- **dx** (float) – Pixel size in the X dimension in projected units.
- **dy** (float) – Pixel size in the Y dimension in projected units.
- **progress** (Callable[[float], None]) – Progress callback.

Returns Success of operation.

Return type boolean

buildPoints (*error*=10[, *min_image*][[, *progress*]])
Rebuild point cloud for the chunk.

Parameters

- **error** (float) – Reprojection error threshold.
- **min_image** (int) – Minimum number of point projections.
- **progress** (Callable[[float], None]) – Progress callback.

Returns Success of operation.

Return type boolean

buildTexture (*blending*=MosaicBlending, *color_correction*=False, *size*=2048[, *cameras*][[, *progress*]])
Generate texture for the chunk.

Parameters

- **blending** (PhotoScan.BlendingMode) – Texture blending mode.
- **color_correction** (boolean) – Enables color correction.
- **size** (int) – Texture size.
- **cameras** (list of Camera) – A list of cameras to be used for texturing.
- **progress** (Callable[[float], None]) – Progress callback.

Returns Success of operation.

Return type boolean

buildTiledModel ([*pixel_size*], *tile_size*=256[, *progress*])
Build tiled model for the chunk.

Parameters

- **pixel_size** (float) – Target model resolution in meters.

- **tile_size** (*int*) – Size of tiles in pixels.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

buildUV (*mapping=GenericMapping, count=1* [*camera*] [*progress*])
Generate uv mapping for the model.

Parameters

- **mapping** (*PhotoScan.MappingMode*) – Texture mapping mode.
- **count** (*int*) – Texture count.
- **camera** (*Camera*) – Camera to be used for texturing in MappingCamera mode.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

camera_groups
List of camera groups in the chunk.

Type list of *CameraGroup*

cameras
List of cameras in the chunk.

Type list of *Camera*

cir_transform
CIR calibration matrix.

Type *CirTransform*

copy (*frames*)
Make a copy of the chunk.

Parameters **frames** (list of *Frame*) – Optional list of frames to be copied.

Returns Copy of the chunk.

Return type *Chunk*

crs
Geographic coordinate system used as a world coordinate system.

Type *CoordinateSystem*

decimateModel (*face_count* [*progress*])
Decimate the model to the specified face count.

Parameters

- **face_count** (*int*) – Target face count.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

dense_cloud
Generated dense point cloud for the current frame.

Type `DenseCloud`

depth_maps

Generated depth maps for the current frame.

Type `DepthMaps`

detectMarkers (*type=TargetCircular12bit, tolerance=50, inverted=False, noparity=False* [, *progress*])

Create markers from coded targets.

Parameters

- **type** (`PhotoScan.TargetType`) – Type of targets.
- **tolerance** (*int*) – Detector tolerance (0 - 100).
- **inverted** (*boolean*) – Detect markers on black background.
- **noparity** (*boolean*) – Disable parity checking.
- **progress** (`Callable[[float], None]`) – Progress callback.

Returns Success of operation.

Return type `boolean`

elevation

Generated elevation model for the current frame.

Type `Elevation`

enabled

Enables/disables the chunk.

Type `boolean`

estimateImageQuality ([*cameras*] [, *progress*])

Estimate image quality.

Parameters

- **cameras** (list of `Camera`) – Optional list of cameras to be processed.
- **progress** (`Callable[[float], None]`) – Progress callback.

Returns Success of operation.

Return type `boolean`

exportCameras (*path, format='xml', projection, rotation_order='xyz'*)

Export point cloud and/or camera positions.

Parameters

- **path** (*string*) – Path to output file.
- **format** (*string*) – Export format in ['xml', 'chan', 'boujou', 'bundler', 'rzml', 'opk', 'patb', 'bingo', 'aerosys', 'inpho'].
- **projection** (`CoordinateSystem`) – Output coordinate system.
- **rotation_order** (*string*) – Rotation order (CHAN format only) in ['xyz', 'xzy', 'yxz', 'yzx', 'zxy', 'zyx']

Returns Success of operation.

Return type `boolean`

exportContours (*path*, *items*='polygons')

Export contours to shape file.

Parameters

- **path** (*string*) – Path to shape file.
- **items** (*string*) – Items to export in ['polylines', 'polygons'].

Returns Success of operation.

Return type boolean

exportDem (*path*, *format*='tif'[, *projection*][, *region*][, *dx*][, *dy*][, *blockw*][, *blockh*], *nodata*=-32767, *write_kml*=False, *write_world*=False, *tiff_big*=False[, *progress*])

Export digital elevation model.

Parameters

- **path** (*string*) – Path to output DEM.
- **format** (*string*) – Export format in ['tif', 'asc', 'bil', 'xyz', 'kmz'].
- **projection** (*CoordinateSystem*) – Output coordinate system.
- **region** (*tuple of 4 floats*) – Region to be exported in the (x0, y0, x1, y1) format.
- **dx** (*float*) – Pixel size in the X dimension in projected units.
- **dy** (*float*) – Pixel size in the Y dimension in projected units.
- **blockw** (*int*) – Specifies block width of the DEM mosaic in pixels.
- **blockh** (*int*) – Specifies block height of the DEM mosaic in pixels.
- **nodata** (*float*) – No-data value.
- **write_kml** (*boolean*) – Enables/disables kml file generation.
- **write_world** (*boolean*) – Enables/disables world file generation.
- **tiff_big** (*boolean*) – Enables/disables BigTIFF compression for TIFF files.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

exportMarkers (*path*)

Export markers.

Parameters **path** (*string*) – Path to output file.

Returns Success of operation.

Return type boolean

exportMatches (*path*, *format*='bingo', *precision*=3, *export_points*=True, *export_markers*=False, *use_labels*=False[, *progress*])

Export point matches.

Parameters

- **path** (*string*) – Path to output file.
- **format** (*string*) – Export format in ['bingo', 'orima', 'patb'].
- **precision** (*int*) – Number of digits after the decimal point.

- **export_points** (*boolean*) – Enables/disables export of automatic tie points.
- **export_markers** (*boolean*) – Enables/disables export of manual matching points.
- **use_labels** (*boolean*) – Enables/disables label based item identifiers.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

exportModel (*path*, *binary=True*, *precision=6*, *texture_format='jpg'*, *texture=True*, *normals=True*, *colors=True*, *cameras=True*, *udim=False*, *strip_extensions=False* [, *comment*] [, *format*] [, *projection*] [, *shift*] [, *progress*])

Export generated model for the chunk.

Parameters

- **path** (*string*) – Path to output model.
- **binary** (*boolean*) – Enables/disables binary encoding (if supported by format).
- **precision** (*int*) – Number of digits after the decimal point (for text formats).
- **texture_format** (*string*) – Texture format in ['jpg', 'png', 'tif', 'exr', 'bmp'].
- **texture** (*boolean*) – Enables/disables texture export.
- **normals** (*boolean*) – Enables/disables export of vertex normals.
- **colors** (*boolean*) – Enables/disables export of vertex colors.
- **cameras** (*boolean*) – Enables/disables camera export.
- **udim** (*boolean*) – Enables/disables UDIM texture layout.
- **strip_extensions** (*boolean*) – Strips camera label extensions during export.
- **comment** (*string*) – Optional comment (if supported by selected format).
- **format** (*string*) – Export format in ['3ds', 'obj', 'ply', 'vrmf', 'collada', 'dxf', 'dxf-3dface', 'fbx', 'pdf', 'u3d', 'stl', 'kmz'].
- **projection** (*CoordinateSystem*) – Output coordinate system.
- **shift** (*3-element vector*) – Optional shift to be applied to vertex coordinates.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

exportOrthomosaic (*path*, *format='tif'*, *raster_transform=RasterTransformNone* [, *projection*] [, *region*] [, *dx*] [, *dy*] [, *blockw*] [, *blockh*], *write_kml=False*, *write_world=False*, *tiff_compression='lzw'*, *tiff_big=False* [, *progress*])

Export orthophoto for the chunk.

Parameters

- **path** (*string*) – Path to output orthophoto.
- **format** (*string*) – Export format in ['tif', 'jpg', 'png', 'kmz'].
- **raster_transform** (*PhotoScan.RasterTransformType*) – Raster band transformation.
- **projection** (*CoordinateSystem*) – Output coordinate system.

- **region** (*tuple of 4 floats*) – Region to be exported in the (x0, y0, x1, y1) format.
- **dx** (*float*) – Pixel size in the X dimension in projected units.
- **dy** (*float*) – Pixel size in the Y dimension in projected units.
- **blockw** (*int*) – Specifies block width of the orthophoto mosaic in pixels.
- **blockh** (*int*) – Specifies block height of the orthophoto mosaic in pixels.
- **write_kml** (*boolean*) – Enables/disables kml file generation.
- **write_world** (*boolean*) – Enables/disables world file generation.
- **tiff_compression** (*string*) – Tiff compression in ['none', 'lzw', 'jpeg', 'packbits', 'deflate'].
- **tiff_big** (*boolean*) – Enables/disables BigTIFF compression for TIFF files.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

exportOrthophotos (*path*, *raster_transform*=*RasterTransformNone*[[, *projection*][[, *region*][[, *dx*][[, *dy*], *write_kml*=*False*, *write_world*=*False*, *tiff_compression*='lzw', *tiff_big*=*False*[[, *progress*]])

Export orthophoto for the chunk.

Parameters

- **path** (*string*) – Path to output orthophoto.
- **raster_transform** (*PhotoScan.RasterTransformType*) – Raster band transformation.
- **projection** (*CoordinateSystem*) – Output coordinate system.
- **region** (*tuple of 4 floats*) – Region to be exported in the (x0, y0, x1, y1) format.
- **dx** (*float*) – Pixel size in the X dimension in projected units.
- **dy** (*float*) – Pixel size in the Y dimension in projected units.
- **write_kml** (*boolean*) – Enables/disables kml file generation.
- **write_world** (*boolean*) – Enables/disables world file generation.
- **tiff_compression** (*string*) – Tiff compression in ['none', 'lzw', 'jpeg', 'packbits', 'deflate'].
- **tiff_big** (*boolean*) – Enables/disables BigTIFF compression for TIFF files.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

exportPoints (*path*[[, *source*], *binary*=*True*, *precision*=*6*, *normals*=*True*, *colors*=*True*[[, *comment*][[, *format*][[, *projection*][[, *shift*][[, *blockw*][[, *blockh*][[, *classes*][[, *progress*]])

Export point cloud.

Parameters

- **path** (*string*) – Path to output file.

- **source** (`PhotoScan.DataSource`) – Selects between dense point cloud and sparse point cloud. If not specified, uses dense cloud if available.
- **binary** (*boolean*) – Enables/disables binary encoding for selected format (if applicable).
- **precision** (*int*) – Number of digits after the decimal point (for text formats).
- **normals** (*boolean*) – Enables/disables export of point normals.
- **colors** (*boolean*) – Enables/disables export of point colors.
- **comment** (*string*) – Optional comment (if supported by selected format).
- **format** (*string*) – Export format in ['obj', 'ply', 'xyz', 'las', 'laz', 'e57', 'cl3', 'pts', 'u3d', 'pdf', 'potree', 'oc3'].
- **projection** (`CoordinateSystem`) – Output coordinate system.
- **shift** (*3-element vector*) – Optional shift to be applied to vertex coordinates.
- **blockw** (*float*) – Tile width in meters.
- **blockh** (*float*) – Tile height in meters.
- **classes** (*list of int*) – List of dense point classes to be exported.
- **progress** (`Callable[[float], None]`) – Progress callback.

Returns Success of operation.

Return type boolean

exportReport (*path* [, *title*] [, *description*] [, *progress*])

Export processing report in PDF format.

Parameters

- **path** (*string*) – Path to output report.
- **title** (*string*) – Report title.
- **description** (*string*) – Report description.
- **progress** (`Callable[[float], None]`) – Progress callback.

Returns Success of operation.

Return type boolean

exportShapes (*path*, *items*=*'polygons'*)

Export shapes layer to file.

Parameters

- **path** (*string*) – Path to shape file.
- **items** (*string*) – Items to export in ['points', 'polylines', 'polygons'].

Returns Success of operation.

Return type boolean

exportTiledModel (*path*, *format*=*'tls'*, *mesh_format*=*'collada'* [, *progress*])

Export generated tiled model for the chunk.

Parameters

- **path** (*string*) – Path to output model.
- **format** (*string*) – Export format in ['tls', 'lod', 'zip'].

- **mesh_format** (*string*) – Mesh format for zip export in ['3ds', 'obj', 'ply', 'vrm1', 'colada', 'dxf', 'fbx', 'pdf', 'u3d', 'stl'].
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

frame

Current frame index.

Type int

frames

List of frames in the chunk.

Type list of `Frame`

image_brightness

Image brightness as percentage.

Type float

importCameras (*path, format='xml'*)

Import camera positions.

Parameters

- **path** (*string*) – Path to the file.
- **format** (*string*) – File format in ['xml', 'bingo', 'bundler', 'rzml', 'visionmap'].

Returns Success of operation.

Return type boolean

importDem (*path* [, *projection*] [, *progress*])

Import elevation model from file.

Parameters

- **path** (*string*) – Path to elevation model in GeoTIFF format.
- **projection** (`CoordinateSystem`) – Default coordinate system if not specified in GeoTIFF file.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

importMarkers (*path*)

Import markers.

Parameters **path** (*string*) – Path to the file.

Returns Success of operation.

Return type boolean

importMasks (*path=''*, *method='alpha'*, *operation='replacement'*, *tolerance=10* [, *cameras*] [, *progress*])

Import masks for multiple cameras.

Parameters

- **path** (*string*) – Mask file name template.

- **method** (*string*) – Method in ['alpha', 'file', 'background', 'model'].
- **operation** (*string*) – Operation in ['replacement', 'union', 'intersection', 'difference'].
- **tolerance** (*int*) – Background masking tolerance.
- **cameras** (list of `Camera`) – Optional list of cameras to be processed.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

importModel (*path* [, *format*] [, *projection*] [, *shift*] [, *progress*])

Import model from file.

Parameters

- **path** (*string*) – Path to model.
- **format** (*string*) – Model format in ['obj', 'ply', '3ds', 'dae', 'dxf', 'fbx', 'u3d', 'stl'].
- **projection** (`CoordinateSystem`) – Model coordinate system.
- **shift** (*3-element vector*) – Optional shift to be applied to vertex coordinates.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

importShapes (*path*='', *replace*=False, *boundary*=`Shape.NoBoundary`)

Import shapes layer from file.

Parameters

- **path** (*string*) – Path to shape file.
- **replace** (*boolean*) – Replace current shapes with new data.
- **boundary** (`Shape.BoundaryType`) – Boundary type to be applied to imported shapes.

Returns Success of operation.

Return type boolean

key

Chunk identifier.

Type int

label

Chunk label.

Type string

loadReference (*path*, *format*, *columns*='nxyzabc', *delimiter*=' ', *group_delimiters*=False, *skip_rows*=0)

Import reference data from the specified file.

Parameters

- **path** (*string*) – Path to the file with reference data.
- **format** (*string*) – Format of the file in ['xml', 'tel', 'csv', 'mavinci', 'bramor']
- **columns** (*string*) – column order in csv format (n - label, x/y/z - coordinates, s - accuracy, a/b/c - yaw, pitch, roll)

- **delimiter** (*string*) – column delimiter in csv format
- **group_delimiters** (*boolean*) – combine consecutive delimiters in csv format
- **skip_rows** (*int*) – number of rows to skip in csv format

Returns Success of operation.

Return type boolean

loadReferenceExif ()

Import camera locations from EXIF meta data.

Returns Success of operation.

Return type boolean

markers

List of markers in the chunk.

Type list of `Marker`

master_channel

Master channel index (-1 for default).

Type int

matchPhotos (*accuracy=HighAccuracy, preselection=NoPreselection, filter_mask=False, keypoint_limit=40000, tiepoint_limit=4000[, progress]*)

Perform image matching for the chunk frame.

Parameters

- **accuracy** (`PhotoScan.Accuracy`) – Alignment accuracy.
- **preselection** (`PhotoScan.Preselection`) – Image pair preselection method.
- **filter_mask** (*boolean*) – Filter points by mask.
- **keypoint_limit** (*int*) – Maximum number of key points to look for in each photo.
- **tiepoint_limit** (*int*) – Maximum number of tie points to generate for each photo.
- **progress** (`Callable[[float], None]`) – Progress callback.

Returns Success of operation.

Return type boolean

meta

Chunk meta data.

Type `MetaData`

model

Generated model for the current frame.

Type `Model`

optimizeCameras (*fit_f=True, fit_cxcy=True, fit_b1=True, fit_b2=True, fit_k1k2k3=True, fit_p1p2=True, fit_k4=False, fit_p3=False, fit_p4=False[, progress]*)

Perform optimization of point cloud / camera parameters.

Parameters

- **fit_f** (*boolean*) – Enables optimization of focal length coefficient.
- **fit_cxcy** (*boolean*) – Enables optimization of principal point coordinates.

- **fit_b1** (*boolean*) – Enabled optimization of aspect ratio.
- **fit_b2** (*boolean*) – Enables optimization of skew coefficient.
- **fit_k1k2k3** (*boolean*) – Enables optimization of k1, k2 and k3 radial distortion coefficients.
- **fit_p1p2** (*boolean*) – Enables optimization of p1 and p2 tangential distortion coefficients.
- **fit_k4** (*boolean*) – Enables optimization of k4 radial distortion coefficient.
- **fit_p3** (*boolean*) – Enables optimization of p3 tangential distortion coefficient.
- **fit_p4** (*boolean*) – Enables optimization of p4 tangential distortion coefficient.
- **progress** (*Callable[[float], None]*) – Progress callback.

Returns Success of operation.

Return type boolean

orthomosaic

Generated orthomosaic for the current frame.

Type `Orthomosaic`

point_cloud

Generated sparse point cloud.

Type `PointCloud`

raster_transform

Raster transform.

Type `RasterTransform`

region

Reconstruction volume selection.

Type `Region`

remove (*items*)

Remove items from the chunk.

Parameters *items* (list of `Frame`, `Sensor`, `CameraGroup`, `Camera`, `Marker` or `Scalebar`) – A list of items to be removed.

Returns Success of operation.

Return type boolean

resetRegion ()

Reset reconstruction volume selector to default position.

saveReference (*path*, *format*, *items*=*'cameras'*)

Export reference data to the specified file.

Parameters

- **path** (*string*) – Path to the output file.
- **format** (*string*) – Export format in [`'xml'`, `'tel'`, `'csv'`].
- **items** (*string*) – Items to export in CSV format in [`'cameras'`, `'markers'`, `'scalebars'`].

Returns Success of operation.

Return type boolean

scalebars

List of scale bars in the chunk.

Type list of `Scalebar`

selected

Selects/deselects the chunk.

Type boolean

sensors

List of sensors in the chunk.

Type list of `Sensor`

shapes

Shapes for the current frame.

Type `Shapes`

smoothModel (*passes* = 3[, *progress*])

Smooth mesh using Laplacian smoothing algorithm.

Parameters

- **passes** (*int*) – Number of smoothing passes to perform.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

Returns Success of operation.

Return type boolean

thinPointCloud (*point_limit*=1000)

Remove excessive tracks from the point cloud.

Parameters **point_limit** (*int*) – Maximum number of points for each photo.

Returns Success of operation.

Return type boolean

trackMarkers ([*start*][, *end*][, *progress*])

Track marker projections through the frame sequence.

Parameters

- **start** (*int*) – Starting frame index.
- **end** (*int*) – Ending frame index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

Returns Success of operation.

Return type boolean

ttransform

4x4 matrix specifying chunk location in the world coordinate system.

Type `ChunkTransform`

updateTransform ()

Update chunk transformation based on reference data.

class `PhotoScan.ChunkTransform`

Transformation between chunk and world coordinates systems.

matrix
Transformation matrix.

Type `Matrix`

rotation
Rotation component.

Type `Matrix`

scale
Scale component.

Type `float`

translation
Translation component.

Type `Vector`

class `PhotoScan.CirTransform`
CIR calibration matrix.

calibrate ()
Calibrate CIR matrix based on orthomosaic histogram. :return: Success of operation. :rtype: boolean

coeffs
Color matrix.

Type `Matrix`

reset ()
Reset CIR calibration matrix.

class `PhotoScan.ConsolePane`
ConsolePane class provides access to the console pane

clear ()
Clear console pane.

contents
Console pane contents.

Type `string`

class `PhotoScan.CoordinateSystem`
Coordinate reference system (local, geographic or projected).

The following example changes chunk coordinate system to WGS 84 / UTM zone 41N and loads reference data from file:

```
>>> import PhotoScan
>>> chunk = PhotoScan.app.document.chunk
>>> chunk.crs = PhotoScan.CoordinateSystem("EPSG::32641")
>>> chunk.loadReference("gcp.txt", "csv")
>>> chunk.updateTransform()
```

authority
Authority identifier of the coordinate system.

Type `string`

init (*crs*)
Initialize projection based on specified WKT definition or authority identifier.

Parameters `crs` (*string*) – WKT definition of coordinate system or authority identifier.

Returns Success of operation.

Return type `boolean`

localframe (*point*)

Returns 4x4 transformation matrix to LSE coordinates at the given point.

Parameters `point` (`Vector`) – Coordinates of the origin in the geocentric coordinates.

Returns Transformation from geocentric coordinates to local coordinates.

Return type `Matrix`

project (*point*)

Projects point from geocentric coordinates to projected geographic coordinate system.

Parameters `point` (`Vector`) – 3D point in geocentric coordinates.

Returns 3D point in projected coordinates.

Return type `Vector`

unproject (*point*)

Unprojects point from projected coordinates to geocentric coordinates.

Parameters `point` (`Vector`) – 3D point in projected coordinate system.

Returns 3D point in geocentric coordinates.

Return type `Vector`

wkt

WKT string identifier of the coordinate system.

Type `string`

class `PhotoScan.DataSource`

Data source in [`PointCloudData`, `DenseCloudData`, `ModelData`, `ElevationData`]

class `PhotoScan.DenseCloud`

Dense point cloud data.

assignClass (*to=0, from=-1*)

Assign class to points with specified original class.

Parameters

- **to** (*int*) – Target class.
- **from** (*int*) – Initial class (-1 for any class).

assignClassToSelection (*to=0, from=-1*)

Assign class to selected points with specified original class.

Parameters

- **to** (*int*) – Target class.
- **from** (*int*) – Initial class (-1 for any class).

classifyGroundPoints (*max_angle=15.0, max_distance=1.0, cell_size=50.0*)

Classify points into ground and non ground classes.

Parameters

- **max_angle** (*float*) – Maximum angle (degrees).

- **max_distance** (*float*) – Maximum distance (meters).
- **cell_size** (*float*) – Cell size (meters).

Returns Success of operation.

Return type boolean

compactPoints ()

Permanently removes deleted points from dense cloud.

Returns Success of operation.

Return type boolean

copy ()

Returns a copy of the dense cloud.

Returns Copy of the dense cloud.

Return type `DenseCloud`

cropSelectedPoints ([*point_class*])

Crop selected points.

Parameters **point_class** (*int*) – Class of points to be removed.

meta

Dense cloud meta data.

Type `MetaData`

point_count

Number of points in dense cloud.

Type `int`

removePoints (*point_class*)

Remove selected points.

Parameters **point_class** (*int*) – Class of points to be removed.

removeSelectedPoints ([*point_class*])

Remove selected points.

Parameters **point_class** (*int*) – Class of points to be removed.

selectMaskedPoints (*cameras*, *softness=4*)

Select dense points based on image masks.

Parameters

- **cameras** (list of `Camera`) – A list of cameras to use for selection.
- **softness** (*float*) – Mask edge softness.

Returns Success of operation.

Return type boolean

class `PhotoScan.DepthMap`

Depth map data.

calibration

Depth map calibration.

Type `Calibration`

copy ()

Returns a copy of the depth map.

Returns Copy of the depth map.

Return type `DepthMap`

image ()

Returns image data.

Returns Image data.

Return type `Image`

setImage (image)

Parameters **image** (`Image`) – Image object with depth map data.

class `PhotoScan.DepthMaps`

A set of depth maps generated for a chunk frame.

items ()

List of items.

keys ()

List of item keys.

meta

Depth maps meta data.

Type `MetaData`

values ()

List of item values.

class `PhotoScan.Document`

PhotoScan project.

Contains list of chunks available in the project. Implements processing operations that work with multiple chunks. Supports saving/loading project files.

The project currently opened in PhotoScan window can be accessed using `PhotoScan.app.document` attribute. Additional Document objects can be created as needed.

The following example saves active chunk from the opened project in a separate project:

```
>>> import PhotoScan
>>> doc = PhotoScan.app.document
>>> doc.save(path = "project.psz", chunks = [doc.chunk])
```

addChunk ()

Add new chunk to the document.

Returns Created chunk.

Return type `Chunk`

alignChunks (chunks, reference, method='points', fix_scale=False, accuracy=HighAccuracy, preselection=False, filter_mask=False, point_limit=40000)

Align specified set of chunks.

Parameters

- **chunks** (*list*) – List of chunks to be aligned.
- **reference** (`Chunk`) – Chunk to be used as a reference.

- **method** (*string*) – Alignment method in ['points', 'markers', 'cameras'].
- **fix_scale** (*boolean*) – Fixes chunk scale during alignment.
- **accuracy** (*PhotoScan.Accuracy*) – Alignment accuracy.
- **preselection** (*boolean*) – Enables image pair preselection.
- **filter_mask** (*boolean*) – Filter points by mask.
- **point_limit** (*int*) – Maximum number of points for each photo.

Returns Success of operation.

Return type boolean

append (*document*)

Append the specified Document object to the current document.

Parameters **document** (*Document*) – Document object to be appended.

Returns Success of operation.

Return type boolean

chunk

Active Chunk.

Type *Chunk*

chunks

List of chunks in the document.

Type *Chunks*

clear ()

Clear the contents of the Document object.

Returns Success of operation.

Return type boolean

mergeChunks (*chunks, merge_dense_clouds=False, merge_models=False, merge_markers=False*)

Merge specified set of chunks.

Parameters

- **chunks** (*list*) – List of chunks to be merged.
- **merge_dense_clouds** (*boolean*) – Enables/disables merging of dense clouds.
- **merge_models** (*boolean*) – Enables/disables merging of polygonal models.
- **merge_markers** (*boolean*) – Enables/disables merging of corresponding marker across the chunks.

Returns Success of operation.

Return type boolean

meta

Document meta data.

Type *MetaData*

open (*path*)

Load document from the specified file.

Parameters **path** (*string*) – Path to the file.

Returns Success of operation.

Return type boolean

path

Path to the document file.

Type string

remove (*items*)

Remove a set of items from the document.

Parameters **items** (list of [Chunk](#)) – A list of items to be removed.

Returns Success of operation.

Return type boolean

save ([*path*] [, *chunks*], *compression* = 6, *absolute_paths* = *False*)

Save document to the specified file.

Parameters

- **path** (*string*) – Optional path to the file.
- **chunks** (list of [Chunk](#)) – List of chunks to be saved.
- **compression** (*int*) – Project compression level.
- **absolute_paths** (*boolean*) – Store absolute image paths.

Returns Success of operation.

Return type boolean

class `PhotoScan.Elevation`

Digital elevation model.

bottom

Y coordinate of the bottom side.

Type float

crs

Coordinate system of elevation model.

Type `CoordinateSystem`

height

Elevation model height.

Type int

left

X coordinate of the left side.

Type float

max

Maximum elevation value.

Type float

meta

Elevation model meta data.

Type `MetaData`

min
Minimum elevation value.

Type float

resolution
DEM resolution in meters.

Type float

right
X coordinate of the right side.

Type float

top
Y coordinate of the top side.

Type float

width
Elevation model width.

Type int

class PhotoScan.**FaceCount**
Face count in [HighFaceCount, MediumFaceCount, LowFaceCount]

class PhotoScan.**FilterMode**
Depth filtering mode in [AggressiveFiltering, ModerateFiltering, MildFiltering, NoFiltering]

class PhotoScan.**Image**
Image(width, height, channels, datatype='U8')

1 or 3-channel image

Parameters

- **width** (*int*) – image width
- **height** (*int*) – image height
- **channels** (*string*) – color channel layout, e.g. 'RGB', 'RGBA', etc.

channels
Channel mapping for the image.

Type string

cn
Number of color channels.

Type int

convert (*channels*[, *datatype*])
Convert image to specified data type and channel layout.

Parameters

- **channels** (*string*) – color channels to be loaded, e.g. 'RGB', 'RGBA', etc.
- **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F32', 'F64']

Returns Converted image.

Return type [Image](#)

copy()

Return a copy of the image.

Returns copy of the image

Return type `Image`

data_type

Data type used to store pixel values.

Type string

fromstring(data, width, height, channels, datatype='U8')

Create image from byte array.

Parameters

- **data** (*string*) – raw image data
- **width** (*int*) – image width
- **height** (*int*) – image height
- **channels** (*string*) – color channel layout, e.g. 'RGB', 'RGBA', etc.
- **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F32', 'F64']

Returns Created image.

Return type `Image`

height

Image height.

Type int

open(path, layer=0, datatype='U8', channels)

Load image from file.

Parameters

- **path** (*string*) – path to the image file
- **layer** (*int*) – image layer in case of multipage file
- **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F32', 'F64']
- **channels** (*string*) – color channels to be loaded, e.g. 'RGB', 'RGBA', etc.

Returns Loaded image.

Return type `Image`

resize(width, height)

Resize image to specified dimensions.

Parameters

- **width** (*int*) – new image width
- **height** (*int*) – new image height

Returns resized image

Return type `Image`

save(path)

Save image to the file.

Parameters `path` (*string*) – path to the image file

Returns success of operation

Return type boolean

tostring ()

Convert image to byte array.

Returns Raw image data.

Return type string

undistort (*calib, center_principal_point = True, square_pixels = True*)

Undistort image using provided calibration.

Parameters

- **calib** (*Calibration*) – lens calibration
- **center_principal_point** (*boolean*) – moves principal point to the image center
- **square_pixels** (*boolean*) – create image with square pixels

Returns undistorted image

Return type *Image*

warp (*calib0, trans0, calib1, trans1*)

Warp image by rotating virtual viewpoint.

Parameters

- **calib0** (*Calibration*) – initial calibration
- **trans0** (*Matrix*) – initial camera orientation as 4x4 matrix
- **calib1** (*Calibration*) – final calibration
- **trans1** (*Matrix*) – final camera orientation as 4x4 matrix

Returns warped image

Return type *Image*

width

Image width.

Type int

class `PhotoScan.Interpolation`

Interpolation mode in [EnabledInterpolation, DisabledInterpolation, Extrapolated]

class `PhotoScan.MappingMode`

UV mapping mode in [GenericMapping, OrthophotoMapping, AdaptiveOrthophotoMapping, SphericalMapping, CameraMapping]

class `PhotoScan.Marker`

Marker instance

frames

Marker frames.

Type list of *Marker*

key

Marker identifier.

Type int

label

Marker label.

Type string

meta

Marker meta data.

Type `MetaData`

position

Marker position in the current frame.

Type `Vector`

projections

List of marker projections.

Type `MarkerProjections`

reference

Marker reference data.

Type `MarkerReference`

selected

Selects/deselects the marker.

Type boolean

class `PhotoScan.MarkerProjection`

Marker projection.

coord

Point coordinates in pixels.

Type `Vector`

pinned

Pinned flag.

Type boolean

class `PhotoScan.MarkerProjections`

Collection of projections specified for the marker

items ()

List of items.

keys ()

List of item keys.

values ()

List of item values.

class `PhotoScan.MarkerReference`

Marker reference data.

accuracy

Marker location accuracy.

Type `Vector`

enabled

Enabled flag.

Type boolean

location

Marker coordinates.

Type `Vector`

class `PhotoScan.Mask`

Mask instance

copy ()

Returns a copy of the mask.

Returns Copy of the mask.

Return type `Mask`

image ()

Returns image data.

Returns Image data.

Return type `Image`

load (*path* [, *layer*])

Loads mask from file.

Parameters

- **path** (*string*) – Path to the image file to be loaded.
- **layer** (*int*) – Optional layer index in case of multipage files.

Returns Success of operation.

Return type boolean

setImage (*image*)

Parameters **image** (`Image`) – Image object with mask data.

class `PhotoScan.Matrix`

m-by-n matrix

```
>>> import PhotoScan
>>> m1 = PhotoScan.Matrix.diag( (1,2,3,4) )
>>> m3 = PhotoScan.Matrix( [[1,2,3,4], [1,2,3,4], [1,2,3,4], [1,2,3,4]] )
>>> m2 = m1.inv()
>>> m3 = m1 * m2
>>> x = m3.det()
>>> if x == 1:
...     PhotoScan.app.messageBox("Diagonal matrix dimensions: " + str(m3.size))
```

col (*index*)

Returns column of the matrix.

Returns matrix column.

Return type `Vector`

copy ()

Returns a copy of this matrix.

Returns an instance of itself

Return type `Matrix`

det ()

Return the determinant of a matrix.

Returns Return a the determinant of a matrix.

Return type `float`

diag (*vector*)

Create a diagonal matrix.

Parameters **vector** (`Vector` or list of floats) – The vector of diagonal entries.

Returns A diagonal matrix.

Return type `Matrix`

inv ()

Returns an inverted copy of the matrix.

Returns inverted matrix.

Return type `Matrix`

mulp (*point*)

Transforms a point in homogeneous coordinates.

Parameters **point** (`Vector`) – The point to be transformed.

Returns transformed point.

Return type `Vector`

mulv (*vector*)

Transforms vector in homogeneous coordinates.

Parameters **vector** (`Vector`) – The vector to be transformed.

Returns transformed vector.

Return type `Vector`

row (*index*)

Returns row of the matrix.

Returns matrix row.

Return type `Vector`

size

Matrix dimensions.

Type `tuple`

t ()

Return a new, transposed matrix.

Returns a transposed matrix

Return type `Matrix`

translation (*vector*)

Create a translation matrix.

Parameters **vector** (`Vector`) – The translation vector.

Returns A matrix representing translation.

Return type `Matrix`

zero ()

Set all matrix elements to zero.

class `PhotoScan.MeshFace`
Triangular face of the model

hidden

Face visibility flag.

Type boolean

selected

Face selection flag.

Type boolean

tex_vertices

Texture vertex indices.

Type tuple of 3 int

vertices

Vertex indices.

Type tuple of 3 int

class `PhotoScan.MeshFaces`
Collection of model faces

class `PhotoScan.MeshTexVertex`
Texture vertex of the model

coord

Vertex coordinates.

Type tuple of 2 float

class `PhotoScan.MeshTexVertices`
Collection of model texture vertices

class `PhotoScan.MeshVertex`
Vertex of the model

color

Vertex color.

Type tuple of 3 int

coord

Vertex coordinates.

Type `Vector`

class `PhotoScan.MeshVertices`
Collection of model vertices

class `PhotoScan.MetaData`
`MetaData(object)`
Collection of object properties

items ()

List of items.

keys ()
List of item keys.

values ()
List of item values.

class PhotoScan.**Model**
Triangular mesh model instance

area ()
Return area of the model surface.

Returns Model area.

Return type float

closeHoles (*level = 30*)
Fill holes in the model surface.

Parameters **level** (*int*) – Hole size threshold in percents.

Returns Success of operation.

Return type boolean

copy ()
Create a copy of the model.

Returns Copy of the model.

Return type Model

cropSelection ()
Crop selected faces and free vertices from the mesh.

faces
Collection of mesh faces.

Type MeshFaces

fixTopology ()
Remove polygons causing topological problems.

Returns Success of operation.

Return type boolean

loadTexture (*path*)
Load texture from the specified file.

Parameters **path** (*string*) – Path to the image file.

Returns Success of operation.

Return type boolean

meta
Model meta data.

Type MetaData

removeComponents (*size*)
Remove small connected components.

Parameters **size** (*int*) – Threshold on the polygon count of the components to be removed.

Returns Success of operation.

Return type boolean

removeSelection ()

Remove selected faces and free vertices from the mesh.

renderDepth (*transform*, *calibration*)

Render model depth image for specified viewpoint.

Parameters

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.

Returns Rendered image.

Return type *Image*

renderImage (*transform*, *calibration*)

Render model image for specified viewpoint.

Parameters

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.

Returns Rendered image.

Return type *Image*

renderMask (*transform*, *calibration*)

Render model mask image for specified viewpoint.

Parameters

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.

Returns Rendered image.

Return type *Image*

renderNormalMap (*transform*, *calibration*)

Render image with model normals for specified viewpoint.

Parameters

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.

Returns Rendered image.

Return type *Image*

saveTexture (*path*)

Save texture to the specified file.

Parameters **path** (*string*) – Path to the image file.

Returns Success of operation.

Return type boolean

setTexture (*image*, *page=0*)

Initialize texture from image data.

Parameters

- **image** (*Image*) – Texture image.
- **page** (*int*) – Texture index for multitextured models.

Returns Success of operation.

Return type boolean

tex_vertices

Collection of mesh texture vertices.

Type *MeshTexVertices*

texture (*page=0*)

Return texture image.

Parameters **page** (*int*) – Texture index for multitextured models.

Returns Texture image.

Return type *Image*

vertices

Collection of mesh vertices.

Type *MeshVertices*

volume ()

Return volume of the closed model surface.

Returns Model volume.

Return type float

class *PhotoScan*.**NetworkClient**

NetworkClient class provides access to the network processing server and allows to create and manage tasks.

The following example connects to the server and lists active tasks:

```
>>> import PhotoScan
>>> client = PhotoScan.NetworkClient()
>>> client.connect('127.0.0.1')
>>> client.batchList()
```

abortBatch (*batch_id*)

Abort batch.

Parameters **batch_id** (*int*) – Batch id.

Returns Success of operation.

Return type boolean

abortNode (*node_id*)

Abort node.

Parameters **node_id** (*int*) – Node id.

Returns Success of operation.

Return type boolean

batchList ()

Get list of active batches.

Returns List of batches.

Return type list

batchStatus (*batch_id*)

Get batch status.

Parameters **batch_id** (*int*) – Batch id.

Returns Batch status.

Return type dict

connect (*host*, *port=5840*)

Connect to the server.

Parameters

- **host** (*string*) – Server hostname.
- **port** (*int*) – Communication port.

Returns Success of operation.

Return type boolean

createBatch (*path*, *tasks*)

Create new batch.

Parameters

- **path** (*string*) – Project path relative to root folder.
- **tasks** (list of [NetworkTask](#)) – Project path relative to root folder.

Returns Batch id.

Return type int

disconnect ()

Disconnect from the server.

findBatch (*path*)

Get batch id based on project path.

Parameters **path** (*string*) – Project path relative to root folder.

Returns Batch id.

Return type int

nodeList ()

Get list of active nodes.

Returns List of nodes.

Return type list

pauseBatch (*batch_id*)

Pause batch.

Parameters **batch_id** (*int*) – Batch id.

Returns Success of operation.

Return type boolean

pauseNode (*node_id*)

Pause node.

Parameters **node_id** (*int*) – Node id.

Returns Success of operation.

Return type boolean

resumeBatch (*batch_id*)

Resume batch.

Parameters **batch_id** (*int*) – Batch id.

Returns Success of operation.

Return type boolean

resumeNode (*node_id*)

Resume node.

Parameters **node_id** (*int*) – Node id.

Returns Success of operation.

Return type boolean

setBatchPriority (*batch_id, priority*)

Set batch priority.

Parameters **batch_id** (*int*) – Batch id.

Returns Success of operation.

Return type boolean

setNodePriority (*node_id, priority*)

Set node priority.

Parameters **node_id** (*int*) – Node id.

Returns Success of operation.

Return type boolean

class PhotoScan.**NetworkTask**

NetworkTask class contains information about network task and its parameters.

The following example creates a new processing task and submits it to the server:

```
>>> import PhotoScan
>>> task = PhotoScan.NetworkTask()
>>> task.name = 'MatchPhotos'
>>> task.params['keypoint_limit'] = 40000
>>> client = PhotoScan.NetworkClient()
>>> client.connect('127.0.0.1')
>>> batch_id = client.createBatch('processing/project.psx', [task])
>>> client.resumeBatch(batch_id)
```

chunks

List of chunks.

Type list

frames

List of frames.

Type list

name

Task name.

Type string

params

Task parameters.

Type dict

class PhotoScan.**Orthomosaic**

Orthomosaic data.

bottom

Y coordinate of the bottom side.

Type float

crs

Coordinate system of orthomosaic.

Type `CoordinateSystem`

height

Orthomosaic height.

Type int

left

X coordinate of the left side.

Type float

meta

Orthomosaic meta data.

Type `MetaData`

removeOrthophotos ()

Remove orthorectified images from orthomosaic.

resolution

Orthomosaic resolution in meters.

Type float

right

X coordinate of the right side.

Type float

top

Y coordinate of the top side.

Type float

width

Orthomosaic width.

Type int

class PhotoScan.**Photo**

Photo instance

alpha ()

Returns alpha channel data.

Returns Alpha channel data.

Return type `Image`

copy()

Returns a copy of the photo.

Returns Copy of the photo.

Return type `Photo`

image()

Returns image data.

Returns Image data.

Return type `Image`

layer

Layer index in the image file.

Type `int`

meta

Frame meta data.

Type `MetaData`

open (*path* [, *layer*])

Loads specified image file.

Parameters

- **path** (*string*) – Path to the image file to be loaded.
- **layer** (*int*) – Optional layer index in case of multipage files.

Returns Success of operation.

Return type `boolean`

path

Path to the image file.

Type `string`

thumbnail (*width=192, height=192*)

Creates new thumbnail with specified dimensions.

Returns Thumbnail data.

Return type `Thumbnail`

class `PhotoScan.PointCloud`

Sparse point cloud instance

copy()

Returns a copy of the point cloud.

Returns Copy of the point cloud.

Return type `PointCloud`

cropSelectedPoints()

Crop selected points.

cropSelectedTracks()

Crop selected tie points.

export (*path, format='obj'* [, *projection*])

Export point cloud.

Parameters

- **path** (*string*) – Path to output file.
- **format** (*string*) – Export format in ['obj', 'ply'].
- **projection** (*Matrix* or *CoordinateSystem*) – Sets output projection.

Returns Success of operation.

Return type boolean

meta

Point cloud meta data.

Type *MetaData*

points

List of points.

Type *PointCloudPoints*

projections

Point projections for each photo.

Type *PointCloudProjections*

removeSelectedPoints ()

Remove selected points.

removeSelectedTracks ()

Remove selected tie points.

tracks

List of tracks.

Type *PointCloudTracks*

class *PhotoScan.PointCloudCameras*

Collection of *PointCloudProjections* objects indexed by corresponding cameras

class *PhotoScan.PointCloudPoint*

3D point in the point cloud

coord

Point coordinates.

Type *Vector*

selected

Point selection flag.

Type boolean

track_id

Track index.

Type int

valid

Point valid flag.

Type boolean

class *PhotoScan.PointCloudPoints*

Collection of 3D points in the point cloud

class PhotoScan.**PointCloudProjection**

Projection of the 3D point on the photo

coord

Projection coordinates.

Type tuple of 2 float

size

Point size.

Type float

track_id

Track index.

Type int

class PhotoScan.**PointCloudProjections**

Collection of `PointCloudProjection` for the camera

class PhotoScan.**PointCloudTrack**

Track in the point cloud

color

Track color.

Type tuple of 3 int

class PhotoScan.**PointCloudTracks**

Collection of tracks in the point cloud

class PhotoScan.**Preselection**

Image pair preselection in [ReferencePreselection, GenericPreselection, NoPreselection]

class PhotoScan.**Quality**

Dense point cloud quality in [UltraQuality, HighQuality, MediumQuality, LowQuality, LowestQuality]

class PhotoScan.**RasterTransform**

Raster transform definition.

calibrateRange ()

Auto detect range based on orthomosaic histogram. :return: Success of operation. :rtype: boolean

enabled

Enable flag.

Type boolean

formula

Raster calculator expression.

Type string

interpolation

Interpolation enable flag.

Type boolean

palette

Color palette.

Type dict

range

Palette mapping range.

Type tuple

reset ()

Reset raster transform.

class PhotoScan.**RasterTransformType**

Raster transformation type in [RasterTransformNone, RasterTransformValue, RasterTransformPalette]

class PhotoScan.**Region**

Region parameters

center

Region center coordinates.

Type Vector

rot

Region rotation matrix.

Type Matrix

size

Region size.

Type Vector

class PhotoScan.**Scalebar**

Scalebar instance

frames

Scalebar frames.

Type list of Scalebar

key

Scalebar identifier.

Type int

label

Scalebar label.

Type string

meta

Scalebar meta data.

Type MetaData

point0

Start of the scalebar.

Type Marker

point1

End of the scalebar.

Type Marker

reference

Scalebar reference data.

Type ScalebarReference

selected

Selects/deselects the scalebar.

Type boolean

class PhotoScan.**ScalebarReference**

Scalebar reference data

accuracy

Scalebar length accuracy.

Type float

distance

Scalebar length.

Type float

enabled

Enabled flag.

Type boolean

class PhotoScan.**Sensor**

Sensor instance

class **Type**

Sensor type in [Frame, Fisheye, Spherical]

Sensor.**antenna**

GPS antenna correction.

Type [Antenna](#)

Sensor.**calibration**

Refined calibration of the photo.

Type [Calibration](#)

Sensor.**fixed**

Fix calibration flag.

Type boolean

Sensor.**focal_length**

Focal length in mm.

Type float

Sensor.**height**

Image height.

Type int

Sensor.**key**

Sensor identifier.

Type int

Sensor.**label**

Camera label.

Type string

Sensor.**pixel_height**

Pixel height in mm.

Type float

Sensor.pixel_size
Pixel size in mm.
Type `Vector`

Sensor.pixel_width
Pixel width in mm.
Type `float`

Sensor.plane_count
Number of image planes.
Type `int`

Sensor.planes
Sensor planes.
Type list of `Sensor`

Sensor.type
Sensor projection model.
Type `Sensor.Type`

Sensor.user_calib
Custom calibration used as initial calibration during photo alignment.
Type `Calibration`

Sensor.width
Image width.
Type `int`

class PhotoScan.Shape
Shape data.

class BoundaryType
Shape boundary type in [NoBoundary, OuterBoundary, InnerBoundary]

class Shape.Type
Shape type in [Point, Polyline, Polygon]

Shape.boundary
Shape boundary type.
Type `Shape.BoundaryType`

Shape.type
Shape type.
Type `Shape.Type`

class PhotoScan.Shapes
A set of shapes for a chunk frame.

crs
Shapes coordinate system.
Type `CoordinateSystem`

items ()
List of items.

meta
Shapes meta data.

Type `MetaData`

class `PhotoScan.SurfaceType`
Surface type in [Arbitrary, HeightField]

class `PhotoScan.TargetType`
Target type in [CircularTarget12bit, CircularTarget16bit, CircularTarget20bit, CircularTarget, CrossTarget]

class `PhotoScan.Thumbnail`
Thumbnail instance

copy ()
Returns a copy of thumbnail.

Returns Copy of thumbnail.

Return type `Thumbnail`

image ()
Returns image data.

Returns Image data.

Return type `Image`

load (*path* [, *layer*])
Loads thumbnail from file.

Parameters

- **path** (*string*) – Path to the image file to be loaded.
- **layer** (*int*) – Optional layer index in case of multipage files.

Returns Success of operation.

Return type `boolean`

setImage (*image*)

Parameters **image** (`Image`) – Image object with thumbnail data.

class `PhotoScan.Utils`
Utility functions.

createDifferenceMask (*image*, *background*, *tolerance=10*, *fit_colors=True*)
Creates mask from a pair of images or an image and specified color.

Parameters

- **image** (`Image`) – Image to be masked.
- **background** (`Image` or color tuple) – Background image or color value.
- **tolerance** (*int*) – Tolerance value.
- **fit_colors** (*boolean*) – Enables white balance correction.

Returns Resulting mask.

Return type `Image`

estimateImageQuality (*image*)
Estimates image sharpness.

Parameters `image` (`Image`) – Image to be analyzed.

Returns Quality metric.

Return type `float`

mat2opk (`R`)

Calculate omega, phi, kappa from world to camera rotation matrix.

Parameters `R` (`Matrix`) – Rotation matrix.

Returns Omega, phi, kappa angles in degrees.

Return type `Vector`

mat2ypr (`R`)

Calculate yaw, pitch, roll from camera to world rotation matrix.

Parameters `R` (`Matrix`) – Rotation matrix.

Returns Yaw, pitch roll angles in degrees.

Return type `Vector`

opk2mat (`angles`)

Calculate world to camera rotation matrix from omega, phi, kappa angles.

Parameters `angles` (`Vector`) – Omega, phi, kappa angles in degrees.

Returns Rotation matrix.

Return type `Matrix`

ypr2mat (`angles`)

Calculate camera to world rotation matrix from yaw, pitch, roll angles.

Parameters `angles` (`Vector`) – Yaw, pitch, roll angles in degrees.

Returns Rotation matrix.

Return type `Matrix`

class `PhotoScan.Vector`

n-component vector

```
>>> import PhotoScan
>>> vect = PhotoScan.Vector( (1, 2, 3) )
>>> vect2 = vect.copy()
>>> vect2.size = 4
>>> vect2.w = 5
>>> vect2 *= -1.5
>>> vect.size = 4
>>> vect.normalize()
>>> PhotoScan.app.messageBox("Scalar product is " + str(vect2 * vect))
```

copy ()

Return a copy of the vector.

Returns A copy of the vector.

Return type `Vector`

norm ()

Return norm of the vector.

norm2 ()

Return squared norm of the vector.

normalize()

Normalize vector to the unit length.

normalized()

Return a new, normalized vector.

Returns a normalized copy of the vector

Return type `Vector`

size

Vector dimensions.

Type `int`

w

Vector W component.

Type `float`

x

Vector X component.

Type `float`

y

Vector Y component.

Type `float`

z

Vector Z component.

Type `float`

zero()

Set all elements to zero.

class `PhotoScan.Viewpoint`

`Viewpoint(app)`

Represents viewpoint in the model view

center

Camera center.

Type `Vector`

coo

Center of orbit.

Type `Vector`

fov

Camera vertical field of view in degrees.

Type `float`

height

OpenGL window height.

Type `int`

mag

Camera magnification defined by distance to the center of rotation.

Type `float`

rot

Camera rotation matrix.

Type `Matrix`

width

OpenGL window width.

Type `int`

PYTHON API CHANGE LOG

3.1 PhotoScan version 1.2.4 build 2399

- Added NetworkClient and NetworkTask classes
- Added Calibration.f, Calibration.b1, Calibration.b2 attributes
- Added Chunk.exportMatches() method
- Added DenseCloud.compactPoints() method
- Added Orthomosaic.removeOrthophotos() method
- Added fit_b1 and fit_b2 parameters to Chunk.optimizeCameras() method
- Added tiff_big parameter to Chunk.exportOrthomosaic(), Chunk.exportDem() and Chunk.exportOrthophotos() methods
- Added classes parameter to Chunk.exportPoints() method
- Added progress parameter to processing methods
- Removed Calibration.fx, Calibration.fy, Calibration.skew attributes

3.2 PhotoScan version 1.2.3 build 2331

- Added tiff_compression parameter to Chunk.exportOrthomosaic() and Chunk.exportOrthophotos() methods

3.3 PhotoScan version 1.2.2 build 2294

- Added Camera.orientation attribute
- Added chunks parameter to Document.save() method

3.4 PhotoScan version 1.2.1 build 2278

- Renamed RasterTransform enum to RasterTransformType
- Added CirTransform and RasterTransform classes
- Added Chunk.cir_transform and Chunk.raster_transform attributes
- Added Chunk.exportOrthophotos() method

- Added `udim` parameter to `Chunk.exportModel()` method

3.5 PhotoScan version 1.2.0 build 2198

- Added `Elevation` and `Orthomosaic` classes
- Added `Shape` and `Shapes` classes
- Added `Antenna` class
- Added `DataSource` enum
- Added `Camera.error()` method
- Added `Chunk.buildContours()` and `Chunk.exportContours()` methods
- Added `Chunk.importShapes()` and `Chunk.exportShapes()` methods
- Added `Chunk.importDem()` method
- Added `Utils.mat2opk()`, `Utils.mat2ypr()`, `Utils.opk2mat()` and `Utils.ypr2mat()` methods
- Added `Chunk.elevation`, `Chunk.orthomosaic` and `Chunk.shapes` attributes
- Added `Chunk.accuracy_cameras_ypr` attribute
- Added `Sensor.antenna`, `Sensor.plane_count` and `Sensor.planes` attributes
- Added `Calibration.p3` and `Calibration.p4` attributes
- Added `Camera.planes` attribute
- Added `CameraReference.accuracy_ypr` attribute
- Added `fit_p3` and `fit_p4` parameters to `Chunk.optimizeCameras()` method
- Removed `OrthoSurface` and `PointsSource` enums
- Removed `PointCloud.groups` attribute
- Removed `Chunk.camera_offset` attribute

3.6 PhotoScan version 1.2.0 build 2126

- Added `Chunk.buildDem()` and `Chunk.buildOrthomosaic()` methods
- Added `Chunk.buildTiledModel()` method
- Added `Chunk.exportMarkers()` and `Chunk.importMarkers()` methods
- Renamed `Chunk.exportModelTiled()` to `Chunk.exportTiledModel()`
- Renamed `Chunk.exportOrthophoto()` to `Chunk.exportOrthomosaic()`
- Added `PointCloud.removeSelectedPoints()` and `PointCloud.cropSelectedPoints()` methods
- Added `Application.activated` attribute
- Added `Chunk.image_brightness` attribute
- Added `icon` parameter to `Application.addItem()` method
- Added `title` and `description` parameters to `Chunk.exportReport()` method
- Added `operation` parameter to `Chunk.importMasks()` method

- Added columns, delimiter, group_delimiters, skip_rows parameters to Chunk.loadReference() method
- Added items parameter to Chunk.saveReference() method
- Added CameraReference.accuracy, MarkerReference.accuracy and ScalebarReference.accuracy attributes

3.7 PhotoScan version 1.1.1 build 2009

- Added Chunk.exportModelTiles() method
- Added noparity parameter to Chunk.detectMarkers() method
- Added blockw and blockh parameters to Chunk.exportPoints() method

3.8 PhotoScan version 1.1.0 build 2004

- Added CameraOffset and ConsolePane classes
- Added Application.console attribute
- Added Application.addMenuSeparator() method
- Added Chunk.importMasks() method
- Added Chunk.master_channel and Chunk.camera_offset attributes
- Added DenseCloud.assignClass(), DenseCloud.assignClassToSelection(), DenseCloud.removePoints() methods
- Added DenseCloud.classifyGroundPoints() and DenseCloud.selectMaskedPoints() methods
- Added Model.renderNormalMap() method
- Added DenseCloud.meta and Model.meta attributes
- Added Image.toString() and Image.fromstring() methods
- Added classes parameter to Chunk.buildModel() method
- Added crop_borders parameter to Chunk.exportDem() method
- Added chunk parameter to Document.addChunk() method
- Added format parameter to Calibration.save() and Calibration.load() methods

3.9 PhotoScan version 1.1.0 build 1976

- Added CameraGroup, CameraReference, ChunkTransform, DepthMap, DepthMaps, MarkerReference, MarkerProjection, Mask, PointCloudGroups, PointCloudTrack, PointCloudTracks, ScalebarReference, Thumbnail classes
- Removed Cameras, Chunks, DenseClouds, Frame, Frames, GroundControl, GroundControlLocations, GroundControlLocation, Markers, MarkerPositions, Models, Scalebars, Sensors classes
- Converted string constants to enum objects
- Added Chunk.addSensor, Chunk.addCameraGroup, Chunk.addCamera, Chunk.addMarker, Chunk.addScalebar methods

- Added `Chunk.addPhotos`, `Chunk.addFrame` methods
- Added U16 data type support in `Image` class
- Added `Image.channels` property
- Moved OpenCL settings into `Application` class
- Added `Calibration.error` method
- Added `PointCloud.tracks`, `PointCloud.groups` attributes
- Added `Matrix.mulp` and `Matrix.mulv` methods
- Added `Chunk.key`, `Sensor.key`, `Camera.key`, `Marker.key` and `Scalebar.key` attributes

3.10 PhotoScan version 1.0.0 build 1795

- Added `DenseCloud` and `DenseClouds` classes
- Added `Chunk.exportModel()` and `Chunk.importModel()` methods
- Added `Chunk.estimateImageQuality()` method
- Added `Photo.thumbnail()` method
- Added `Image.resize()` method
- Added `Camera.meta`, `Marker.meta`, `Scalebar.meta` and `Photo.meta` attributes
- Added `Chunk.dense_cloud` and `Chunk.dense_clouds` attributes
- Added `page` parameter to `Model.setTexture()` and `Model.texture()` methods

3.11 PhotoScan version 1.0.0 build 1742

- Added `Chunk.buildDenseCloud()` and `Chunk.smoothModel()` methods
- Added `Application.enumOpenCLDevices()` method
- Added `Utils.estimateImageQuality()` method
- Removed `Chunk.buildDepth()` method
- Removed `Camera.depth()` and `Camera.setDepth()` methods
- Removed `Frame.depth()` and `Frame.setDepth()` methods
- Removed `Frame.depth_calib` attribute
- Changed parameters of `Chunk.buildModel()` and `Chunk.buildTexture()` methods
- Changed parameters of `Chunk.exportPoints()` method
- Changed parameters of `Model.save()` method
- Changed return value of `Chunks.add()` method
- Added shortcut parameter to `Application.addItem()` method
- Added `absolute_paths` parameter to `Document.save()` method
- Added `fit_f`, `fit_cxcy`, `fit_k1k2k3` and `fit_k4` parameters to `Chunk.optimizePhotos()` method

3.12 PhotoScan version 0.9.1 build 1703

- Added Sensor class
- Added Scalebar class
- Added Camera.sensor attribute
- Added Chunk.sensors attribute
- Added Calibration.width and Calibration.height attributes
- Added Chunk.refineMatches() method
- Added Model.area() and Model.volume() methods
- Added Model.renderDepth(), Model.renderImage() and Model.renderMask() methods
- Added MetaData class
- Added Chunk.meta and Document.meta attributes
- Added Calibration.project() and Calibration.unproject() methods
- Added Calibration.k4 attribute
- Added Application.addItem() method
- Added Model.closeHoles() and Model.fixTopology() methods

3.13 PhotoScan version 0.9.0 build 1586

- Added Camera class
- Added Frame class
- Added CoordinateSystem class
- Removed Photo class (deprecated)
- Removed GeoProjection class (deprecated)
- Added Chunk.exportReport() method
- Added Chunk.trackMarkers() and Chunk.detectMarkers() methods
- Added Chunk.extractFrames() and Chunk.removeFrames() methods
- Added Chunk.matchPhotos() method
- Added Chunk.buildDepth() method
- Added Chunk.resetDepth() method
- Revised Chunk.alignPhotos() method
- Revised Chunk.buildPoints() method
- Revised Chunk.buildModel() method
- Added Chunk.cameras property
- Removed Chunk.photos property (deprecated)
- Added Utils.createDifferenceMask() method

3.14 PhotoScan version 0.8.5 build 1423

- Added `Chunk.fix_calibration` property
- Removed “`fix_calibration`” parameter from `Chunk.alignPhotos()` method
- Added `Chunk.exportCameras()` method
- Added `Chunk.exportPoints()` method for dense/sparse point cloud export
- Moved `GroundControl.optimize()` method to `Chunk.optimize()`
- Added `accuracy_cameras`, `accuracy_markers` and `accuracy_projections` properties to the `GroundControl` class
- Added `Image.undistort()` method
- Added `PointCloudPoint.selected` and `PointCloudPoint.valid` properties
- Removed `GeoProjection.epsg` property
- Added `GeoProjection.authority` property
- Added `GeoProjection.init()` method

3.15 PhotoScan version 0.8.4 build 1289

- Added `GroundControl.optimize()` method
- Command line scripting support removed

3.16 PhotoScan version 0.8.3 build 1212

- Revised class: `Chunk`
- Added classes: `Model`, `PointCloud`, `Image`
- `alignPhotos()`, `buildModel()` and `buildTexture()` are now methods of `Chunk` class
- Added export support for point cloud, orthophoto and DEM
- Added `GroundControl` class

3.17 PhotoScan version 0.8.3 build 1154

Initial version of PhotoScan Python API

p

PhotoScan, 5