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
CHAPTER TWO

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:

```python
>>> import PhotoScan
>>> doc = PhotoScan.app.document
>>> 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 [HighAccuracy, MediumAccuracy, LowAccuracy]

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:

```python
>>> 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")
```

`addMenuItem(label, func[, shortcut])`
Create a new menu entry.
Parameters

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

**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.

**Parameters**

**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]

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

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

fx  
    X focal length component.  
    Type float

fy  
    Y focal length component.  
    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'].

_Returns_ success of operation
_Return type_ boolean

**p1**
Tangential distortion coefficient P1.

_Type_ float

**p2**
Tangential distortion coefficient P2.

_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'].
- **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

**skew**
Skew coefficient.
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.

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

frames

Camera frames.

Type list of Camera

group

Camera group.

Type CameraGroup

dkey

Camera identifier.

Type int

dlabel

Camera label.

Type string

dmask

Camera mask.

Type Mask

dmeta

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

photo
    Camera photo.
    Type Photo

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.<code>selected</code>
Current selection state.

Type boolean

CameraGroup.<code>type</code>
Camera group type.

Type CameraGroup.Type

class PhotoScan.<code>CameraOffset</code>
CameraOffset contains camera position relative to GPS antenna.

<code>location</code>
Camera coordinates.

Type Vector

<code>rotation</code>
Camera rotation angles.

Type Vector

class PhotoScan.<code>CameraReference</code>
CameraReference object contains measured camera location data.

<code>enabled</code>
Enabled flag.

Type boolean

<code>location</code>
Camera coordinates.

Type Vector

<code>rotation</code>
Camera rotation angles.

Type Vector

class PhotoScan.<code>Chunk</code>
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 float

accuracy_markers
    Expected accuracy of marker coordinates in meters.
    Type float

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)
    Add a list of photos to the chunk.
    Parameters filenames (list of string) – A list of file paths.
    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`)
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.

**Returns** Success of operation.

**Return type** boolean

**buildDenseCloud** (quality=MediumQuality, `filter`=AggressiveFiltering, `cameras`[], `keep_depth`=False, `reuse_depth`=False)
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.

**Returns** Success of operation.

**Return type** boolean

**buildModel** (surface=Arbitrary, `interpolation`=EnabledInterpolation, `face_count`=MediumFaceCount[, `source`][, `classes`])
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.PointsSource)* – 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.

**Returns** Success of operation.

**Return type** boolean
**buildPoints** (*error=*10, *min_image*)
Rebuild point cloud for the chunk.

**Parameters**
- *error* (*float*) – Reprojection error threshold.
- *min_image* (*int*) – Minimum number of point projections.

**Returns** Success of operation.

**Return type** boolean

**buildTexture** (*blending=*MosaicBlending, *color_correction=False, size=2048, camera*)
Generate texture for the chunk.

**Parameters**
- *blending* (*PhotoScan.BlendingMode*) – Texture blending mode.
- *color_correction* (*boolean*) – Enables color correction.
- *size* (*int*) – Texture size.
- *camera* (*Camera*) – Generates texture from a single camera only if specified.

**Returns** Success of operation.

**Return type** boolean

**buildUV** (*mapping=*GenericMapping, *count=1, camera*)
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.

**Returns** Success of operation.

**Return type** boolean

**camera_groups**
List of camera groups in the chunk.

**Type** list of CameraGroup

**camera_offset**
Camera correction data.

**Type** CameraOffset

**cameras**
List of cameras in the chunk.

**Type** list of Camera

**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)
Decimate the model to the specified face count.

Parameters face_count (int) – Target face count.

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)
Create markers from coded targets.

Parameters
  • type (PhotoScan.TargetType) – Type of targets.
  • tolerance (int) – Detector tolerance (0 - 100).

Returns Success of operation.

Return type boolean
enabled
Enables/disables the chunk.

Type boolean
estimateImageQuality([cameras])
Estimate image quality.

Parameters cameras (list of Camera) – Optional list of cameras to be processed.

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', 'opk', 'path', 'bingo', 'aerosys', 'inpho'].
  • projection (Matrix or CoordinateSystem) – Sets output projection.
  • rotation_order (string) – Rotation order (CHAN format only) in ['xyz', 'xzy', 'yxz', 'yzx', 'zyx', 'zxy']

Returns Success of operation.
Return type boolean

`exportDem(path, format='tif', projection=None, region=None, dx=1, dy=1, blockw=1, blockh=1, nodata=-32767, crop_borders=True, write_kml=False, write_world=False)`

Export digital elevation model.

Parameters

- **path** (string) – Path to output DEM.
- **format** (string) – Export format in ['tif', ‘asc’, ‘bil’, ‘xyz’].
- **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.
- **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.
- **crop_borders** (boolean) – Enables/disables cropping invalid dem regions.

Returns Success of operation.

Return type boolean

`exportModel(path, binary=True, precision=6, texture_format='jpg', texture=True, normals=True, colors=True, cameras=True, comment=None, format=None, projection=None, shift=None)`

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.
- **comment** (string) – Optional comment (if supported by selected format).
- **projection** (CoordinateSystem) – Output coordinate system.
- **shift** (3-element vector) – Optional shift to be applied to vertex coordinates.

Returns Success of operation.

Return type boolean
exportOrthophoto(path, format='tif', blending=MosaicBlending, color_correction=False, projection=[], region=[], dx=[], dy=[], blockw=[], blockh=[], write_kml=False, write_world=False)

Export orthophoto for the chunk.

Parameters

- path (string) – Path to output orthophoto.
- format (string) – Export format in ['tif', 'jpg', 'png', 'kmz'].
- 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.
- 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.

Returns Success of operation.

Return type boolean

exportPoints(path, binary=True, precision=6, normals=True, colors=True, source=None, comment=None, format=['obj', 'ply', 'xyz', 'las', 'u3d', 'pdf', 'e57', 'potree', 'oc3'], projection=None, shift=None)

Export point cloud.

Parameters

- path (string) – Path to output file.
- 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.
- source (PhotoScan.PointsSource) – Selects between dense point cloud and sparse point cloud. If not specified, uses dense cloud if available.
- comment (string) – Optional comment (if supported by selected format).
- format (string) – Export format in ['obj', 'ply', 'xyz', 'las', 'u3d', 'pdf', 'e57', 'potree', 'oc3'].
- projection (CoordinateSystem) – Output coordinate system.
- shift (3-element vector) – Optional shift to be applied to vertex coordinates.

Returns Success of operation.

Return type boolean

exportReport(path)

Export processing report in PDF format.
Parameters **path** *(string)* – Path to output report.

Returns Success of operation.

Return type boolean

**frame**

Current frame index.

Type int

**frames**

List of frames in the chunk.

Type list of Frame

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

Import camera positions.

Parameters

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

Returns Success of operation.

Return type boolean

**importMasks**(path='', method='alpha', tolerance=10, cameras)

Import masks for multiple cameras.

Parameters

- **path** *(string)* – Mask file name template.
- **tolerance** *(int)* – Background masking tolerance.
- **cameras** (list of Camera) – Optional list of cameras to be processed.

Returns Success of operation.

Return type boolean

**importModel**(path, format, projection, shift)

Import model from file.

Parameters

- **path** *(string)* – Path to model.
- **projection** *(CoordinateSystem)* – Model coordinate system.
- **shift** *(3-element vector)* – Optional shift to be applied to vertex coordinates.

Returns Success of operation.

Return type boolean

**key**

Chunk identifier.

Type int
**label**

Chunk label.

Type string

**loadReference** (*path*, *format*)

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']

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=1000*)

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.

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_cxy=True, fit_aspect=True, fit_skew=True, fit_k1k2k3=True, fit_p1p2=True, fit_k4=False*)

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_aspect (boolean)` – Enables optimization of aspect ratio.
- `fit_skew (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.

Returns Success of operation.

Return type boolean

point_cloud
Generated sparse point cloud.

Type PointCloud

refineMatches (filter_mask=False, point_limit=40000)
Perform precise matching.

Parameters

- `filter_mask (boolean)` – Filter points by mask.
- `point_limit (int)` – Maximum number of points for each photo.

Returns Success of operation.

Return type boolean

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)
Export reference data to the specified file.

Parameters

- `path (string)` – Path to the output file.
- `format (string)` – Export format in ['xml', 'tel', 'csv'].

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

smoothModel (passes = 3)
Smooth mesh using Laplacian smoothing algorithm.

Parameters  passes (int) – Number of smoothing passes to perform.

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])
Track marker projections through the frame sequence.

Parameters

•  start (int) – Starting frame index.

•  end (int) – Ending frame index.

Returns  Success of operation.

Return type  boolean

transform
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.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.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

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


```
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.

   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:

```python
>>> import PhotoScan
>>> doc = PhotoScan.app.document
>>> doc2 = PhotoScan.Document()
>>> doc2.addChunk(doc.chunk.copy())
>>> doc2.save("project.psz")
```

**addChunk**

Add chunk to the document. If chunk is not specified, an empty one is created.

**Parameters**

- **chunk** *(Chunk)* – A chunk to be added.

**Returns**

Added chunk.

**Return type**

Chunk

**alignChunks** *(chunks, reference, method='points', fix_scale=False, accuracy='high', 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** *(string)* – Alignment accuracy in ['high', 'medium', 'low'].
- **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], compression = 6, absolute_paths = False)
Save document to the specified file.

Parameters

  • path (string) – optional path to the file.
  • compression (int) – project compression level.
  • absolute_paths (boolean) – store absolute image paths.

Returns Success of operation.

Return type boolean

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

    channels
        Channel mapping for the image.
        Type string

    cn
        Number of color channels.
        Type int

cconvert (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’, ‘F32’]

    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’, ‘F32’]

    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', 'F32']
  - *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
• `calib` ([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.

  **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

```python
>>> 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.
renderNormalMap \( \text{(transform, calibration)} \)

Render image with model normals for specified viewpoint.

**Parameters**

- \text{transform (Matrix)} – Camera location.
- \text{calibration (Calibration)} – Camera calibration.

**Returns**

Rendered image.

**Return type** Image

saveTexture \( \text{(path)} \)

Save texture to the specified file.

**Parameters**

- \text{path (string)} – Path to the image file.

**Returns**

Success of operation.

**Return type** boolean

setTexture \( \text{(image, page=0)} \)

Initialize texture from image data.

**Parameters**

- \text{image (Image)} – Texture image.
- \text{page (int)} – Texture index for multitextured models.

**Returns**

Success of operation.

**Return type** boolean

tex_vertices

Collection of mesh texture vertices.

**Type** MeshTexVertices

texture \( \text{(page=0)} \)

Return texture image.

**Parameters**

- \text{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\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

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

groups
Points for each camera group.

Type **PointCloudGroups**

points
List of points.

Type **PointCloudPoints**

projections
Point projections for each photo.

Type **PointCloudProjections**

tracks
List of tracks.

Type **PointCloudTracks**

class **PhotoScan.PointCloudCameras**
Collection of **PointCloudProjections** objects indexed by corresponding cameras

class **PhotoScan.PointCloudGroups**
Collection of **PointCloudPoints** objects indexed by corresponding camera groups

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

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.PointsSource
Points source in [SparsePoints, DensePoints]

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.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

distance
  Scalebar length.
    Type float

enabled
  Enabled flag.
    Type boolean

class PhotoScan.Sensor
  Sensor instance

class Type
  Sensor type in [Frame, Fisheye, Spherical]

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.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.SurfaceType
Surface type in [Arbitrary, HeightField]

class PhotoScan.TargetType
Target type in [CircularTarget12bit, CircularTarget16bit, CircularTarget20bit, 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

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**
3.1 PhotoScan version 1.1.0 build 2004

- Added CameraOffset and ConsolePane classes
- Added Application.console attribute
- Added Application.addMenuSeparator() method
- Added Chunk.import Masks() 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

3.2 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

3.3 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

3.4 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.addMenuItem() method
• Added absolute_paths parameter to Document.save() method
• Added fit_f, fit_cxcy, fit_k1k2k3 and fit_k4 parameters to Chunk.optimizePhotos() method
3.5 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.addMenuItem() method
- Added Model.closeHoles() and Model.fixTopology() methods

3.6 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
3.7 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

3.8 PhotoScan version 0.8.4 build 1289

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

3.9 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

3.10 PhotoScan version 0.8.3 build 1154

Initial version of PhotoScan Python API
p
PhotoScan, 5