

---

# Metashape Python Reference

*Release 1.8.0*

**Agisoft LLC**

**Dec 31, 2021**



## CONTENTS

<b>1 Overview</b>	<b>3</b>
<b>2 Application Modules</b>	<b>5</b>
<b>3 Python API Change Log</b>	<b>201</b>
<b>Python Module Index</b>	<b>231</b>



Copyright (c) 2021 Agisoft LLC.



## OVERVIEW

### 1.1 Introduction to Python scripting in Metashape Professional

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

---

**Note:** Python scripting is supported only in Metashape Professional edition.

---

Metashape Professional uses Python 3.8 as a scripting engine.

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

- From Metashape “Console” pane using it as standard Python console.
- From the “Tools” menu using “Run script...” command.
- From command line using “-r” argument and passing the path to the script as an argument.

**The following Metashape functionality can be accessed from Python scripts:**

- Open/save/create Metashape 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.
- Start and control network processing tasks.





## APPLICATION MODULES

Metashape 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 Metashape project. Multiple Document instances can be created simultaneously if needed. Besides that a currently opened project in the application can be accessed using `Metashape.app.document` property.

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

```
>>> import Metashape
>>> doc = Metashape.app.document
>>> doc.open("project.psz")
>>> chunk = doc.chunk
>>> chunk.matchPhotos(downscale=1, generic_preselection=True, reference_
↳preselection=False)
>>> chunk.alignCameras()
>>> chunk.buildDepthMaps(downscale=4, filter_mode=Metashape.AggressiveFiltering)
>>> chunk.buildDenseCloud()
>>> chunk.buildModel(surface_type=Metashape.Arbitrary, interpolation=Metashape.
↳EnabledInterpolation)
>>> chunk.buildUV(mapping_mode=Metashape.GenericMapping)
>>> chunk.buildTexture(blending_mode=Metashape.MosaicBlending, texture_size=4096)
>>> doc.save()
```

### **class** Metashape.Antenna

GPS antenna position relative to camera.

#### **copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *Antenna*

#### **fixed**

Fix antenna flag.

**Type** bool

#### **location**

Antenna coordinates.

**Type** *Vector*

#### **location\_acc**

Antenna location accuracy.

Type *Vector*

**location\_covariance**

Antenna location covariance.

Type *Matrix*

**location\_ref**

Antenna location reference.

Type *Vector*

**rotation**

Antenna rotation angles.

Type *Vector*

**rotation\_acc**

Antenna rotation accuracy.

Type *Vector*

**rotation\_covariance**

Antenna rotation covariance.

Type *Matrix*

**rotation\_ref**

Antenna rotation reference.

Type *Vector*

**class Metashape.Application**

Application class provides access to several global application attributes, such as document currently loaded in the user interface, software version and GPU 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 Metashape.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 Metashape
>>> doc = Metashape.app.document
>>> crs = Metashape.app.getCoordinateSystem("Select Coordinate System", doc.chunk.
↳ crs)
>>> doc.chunk.crs = crs
>>> path = Metashape.app.getSaveFileName("Save Project As")
>>> try:
...     doc.save(path)
... except RuntimeError:
...     Metashape.app.messageBox("Can't save project")
```

**class ConsolePane**

ConsolePane class provides access to the console pane

**clear()**

Clear console pane.

**contents**

Console pane contents.

**Type** string

**class ModelView**

ModelView class provides access to the model view

**class DenseCloudViewMode**

Dense cloud view mode in [DenseCloudViewColor, DenseCloudViewClasses, DenseCloudViewConfidence]

**class ModelViewMode**

Model view mode in [ModelViewShaded, ModelViewSolid, ModelViewWireframe, ModelViewConfidence, ModelViewTextured]

**class PointCloudViewMode**

Point cloud view mode in [PointCloudViewColor, PointCloudViewVariance]

**class TiledModelViewMode**

Tiled model view mode in [TiledModelViewTextured, TiledModelViewSolid, TiledModelViewWireframe]

**captureView**([width][, height][, transparent][, hide\_items])

Capture image from model view.

**Parameters**

- **width** (*int*) – Image width.
- **height** (*int*) – Image height.
- **transparent** (*bool*) – Sets transparent background.
- **hide\_items** (*bool*) – Hides all items.

**Returns** Captured image.

**Return type** *Image*

**dense\_cloud\_view\_mode**

Dense cloud view mode.

**Type** *DenseCloudViewMode*

**model\_view\_mode**

Model view mode.

**Type** *ModelViewMode*

**point\_cloud\_view\_mode**

Point cloud view mode.

**Type** *PointCloudViewMode*

**texture\_view\_mode**

Texture view mode.

**Type** *TextureViewMode*

**tiled\_model\_view\_mode**

Tiled model view mode.

**Type** *TiledModelViewMode*

**view\_mode**

View mode.

**Type** *DataSource*

**viewpoint**

Viewpoint in the model view.

**Type** *Viewpoint*

**class OrthoView**

OrthoView class provides access to the ortho view

**captureView**([*width* ][, *height* ][, *transparent* ][, *hide\_items* ])

Capture image from ortho view.

**Parameters**

- **width** (*int*) – Image width.
- **height** (*int*) – Image height.
- **transparent** (*bool*) – Sets transparent background.
- **hide\_items** (*bool*) – Hides all items.

**Returns** Captured image.

**Return type** *Image*

**view\_mode**

View mode.

**Type** *DataSource*

**class PhotosPane**

PhotosPane class provides access to the photos pane

**resetFilter**()

Reset photos pane filter.

**setFilter**(*items*)

Set photos pane filter.

**Parameters** *items* (list of *Camera* or *Marker*) – filter to apply.

**class Settings**

PySettings()

Application settings

**language**

User interface language.

**Type** string

**load**()

Load settings from disk.

**log\_enable**

Enable writing log to file.

**Type** bool

**log\_path**

Log file path.

**Type** string

**network\_enable**

Network processing enabled flag.

**Type** bool

**network\_host**

Network server host name.

**Type** string

**network\_path**

Network data root path.

**Type** string

**network\_port**

Network server control port.

**Type** int

**project\_absolute\_paths**

Store absolute image paths in project files.

**Type** bool

**project\_compression**

Project compression level.

**Type** int

**save()**

Save settings on disk.

**setValue(key, value)**

Set settings value. :arg key: Key. :type key: string :arg value: Value. :type value: object

**value(key)**

Return settings value. :arg key: Key. :type key: string :return: Settings value. :rtype: object

**activated**

Metashape activation status.

**Type** bool

**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\_pane**

Console pane.

**Type** *ConsolePane*

**cpu\_enable**

Use CPU when GPU is active.

**Type** bool

**document**

Main application document object.

**Type** *Document*

**enumGPUDevices()**

Enumerate installed GPU devices.

**Returns** A list of devices.

**Return type** list

**getBool(label="")**

Prompt user for the boolean value.

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

**Returns** Boolean value selected by the user.

**Return type** bool

**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 ]*[, *dir* ])

Prompt user for the existing folder.

**Parameters**

- **hint** (*string*) – Optional text label for the dialog.
- **dir** (*string*) – Optional default folder.

**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 ]*[, *dir* ][, *filter* ])

Prompt user for the existing file.

**Parameters**

- **hint** (*string*) – Optional text label for the dialog.
- **dir** (*string*) – Optional default folder.
- **filter** (*string*) – Optional file filter, e.g. “Text file (*.txt*)” or “.txt”. Multiple filters are separated with “;”.

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

**Return type** string

**getOpenFileNames**(*[hint ]*[, *dir ]*[, *filter ]*)

Prompt user for one or more existing files.

**Parameters**

- **hint** (*string*) – Optional text label for the dialog.
- **dir** (*string*) – Optional default folder.
- **filter** (*string*) – Optional file filter, e.g. “Text file (.txt)” or “.txt”. Multiple filters are separated with “;”.

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

**Return type** list

**getSaveFileName**(*[hint ]*[, *dir ]*[, *filter ]*)

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

**Parameters**

- **hint** (*string*) – Optional text label for the dialog.
- **dir** (*string*) – Optional default folder.
- **filter** (*string*) – Optional file filter, e.g. “Text file (.txt)” or “.txt”. Multiple filters are separated with “;”.

**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.

**model\_view**

Model view.

**Type** *ModelView*

**ortho\_view**

Ortho view.

**Type** *OrthoView*

**photos\_pane**

Photos pane.

**Type** *PhotosPane*

**quit()**

Exit application.

**releaseFreeMemory()**

Call `malloc_trim` on Linux (does nothing on other OS).

**removeMenuItem(*label*)**

Remove menu entry with given label (if exists). If there are multiple entries with given label - all of them will be removed.

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

**settings**

Application settings.

**Type** *Settings*

**title**

Application name.

**Type** *string*

**update()**

Update user interface during long operations.

**version**

Metashape version.

**Type** *string*

**class Metashape.AttachedGeometry**

Attached geometry data.

**GeometryCollection(*geometries*)**

Create a GeometryCollection geometry.

**Parameters** **geometries** (list of *Geometry*) – Child geometries.

**Returns** A GeometryCollection geometry.

**Return type** *Geometry*

**LineString(*coordinates*)**

Create a LineString geometry.

**Parameters** **coordinates** (list of *Vector*) – List of vertex coordinates.

**Returns** A LineString geometry.

**Return type** *Geometry*

**MultiLineString(*geometries*)**

Create a MultiLineString geometry.

**Parameters** **geometries** (list of *Geometry*) – Child line strings.

**Returns** A point geometry.

**Return type** *Geometry*

**MultiPoint(*geometries*)**

Create a MultiPoint geometry.



**Parameters** `geometries` (list of *Geometry*) – Child points.

**Returns** A point geometry.

**Return type** *Geometry*

**MultiPolygon**(*geometries*)

Create a MultiPolygon geometry.

**Parameters** `geometries` (list of *Geometry*) – Child polygons.

**Returns** A point geometry.

**Return type** *Geometry*

**Point**(*vector*)

Create a Point geometry.

**Parameters** `vector` (*Vector* or list of floats) – Point coordinates.

**Returns** A point geometry.

**Return type** *Geometry*

**Polygon**(*exterior\_ring*[, *interior\_rings* ])

Create a Polygon geometry.

**Parameters**

- **exterior\_ring** (list of *Vector*) – Point coordinates.
- **interior\_rings** (list of *Vector*) – Point coordinates.

**Returns** A Polygon geometry.

**Return type** *Geometry*

**coordinates**

List of vertex coordinates.

**Type** *Vector*

**geometries**

List of child geometries.

**Type** *Geometry*

**type**

Geometry type.

**Type** *Geometry.Type*

**class** `Metashape.BBox`

Axis aligned bounding box

**copy**()

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *BBox*

**max**

Maximum bounding box extent.

**Type** *Vector*

**min**  
Minimum bounding box extent.

**Type** *Vector*

**size**  
Bounding box dimension.

**Type** int

**class** Metashape.**BlendingMode**  
Blending mode in [AverageBlending, MosaicBlending, MinBlending, MaxBlending, DisabledBlending]

**class** Metashape.**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

**copy()**  
Return a copy of the object.  
**Returns** A copy of the object.  
**Return type** *Calibration*

**covariance\_matrix**  
Covariance matrix.  
**Type** *Matrix*

**covariance\_params**  
Covariance matrix parameters.  
**Type** list of string

**cx**  
Principal point X coordinate.  
**Type** float

**cy**  
Principal point Y coordinate.  
**Type** float

**error**(*point, proj*)  
Return 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=CalibrationFormatXML*)  
Loads calibration from file.  
**Parameters**
- **path** (*string*) – path to calibration file
  - **format** (*CalibrationFormat*) – Calibration format.
- p1**  
Decentering distortion coefficient P1.  
**Type** float
- p2**  
Decentering distortion coefficient P2.  
**Type** float
- p3**  
Decentering distortion coefficient P3.  
**Type** float
- p4**  
Decentering distortion coefficient P4.  
**Type** float
- project**(*point*)  
Return projected pixel coordinates of the point.  
**Parameters** **point** (*Vector*) – Coordinates of the point to be projected.  
**Returns** 2D projected point coordinates.  
**Return type** *Vector*

**rpc**

RPC model.

Type *RPCModel***save**(*path*, *format=CalibrationFormatXML* [, *label*] [, *pixel\_size*] [, *focal\_length*], *cx = 0*, *cy = 0*)  
Saves calibration to file.**Parameters**

- **path** (*string*) – path to calibration file
- **format** (*CalibrationFormat*) – Calibration format.
- **label** (*string*) – Calibration label used in Australis, CalibCam and CalCam formats.
- **pixel\_size** (*Vector*) – Pixel size in mm used to convert normalized calibration coefficients to Australis and CalibCam coefficients.
- **focal\_length** (*float*) – Focal length (Grid calibration format only).
- **cx** (*float*) – X principal point coordinate (Grid calibration format only).
- **cy** (*float*) – Y principal point coordinate (Grid calibration format only).

**type**

Camera model.

Type *Sensor.Type***unproject**(*point*)

Return 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 Metashape.CalibrationFormat**

Calibration format in [CalibrationFormatXML, CalibrationFormatAustralis, CalibrationFormatAustralisV7, CalibrationFormatPhotoModeler, CalibrationFormatCalibCam, CalibrationFormatCalCam, CalibrationFormatInpho, CalibrationFormatUSGS, CalibrationFormatPix4D, CalibrationFormatOpenCV, CalibrationFormatPhotomod, CalibrationFormatGrid]

**class Metashape.Camera**

Camera instance

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

The following example describes how to create multispectral camera layout:

```

>>> import Metashape
>>> doc = Metashape.app.document
>>> chunk = doc.chunk
>>> rgb = ["RGB_0001.JPG", "RGB_0002.JPG", "RGB_0003.JPG"]
>>> nir = ["NIR_0001.JPG", "NIR_0002.JPG", "NIR_0003.JPG"]
>>> images = [[rgb[0], nir[0]], [rgb[1], nir[1]], [rgb[2], nir[2]]]
>>> chunk.addPhotos(images, Metashape.MultiplaneLayout)

```

**class Reference**

Camera reference data.

**accuracy**

Camera location accuracy.

**Type** *Vector*

**enabled**

Location enabled flag.

**Type** bool

**location**

Camera coordinates.

**Type** *Vector*

**location\_accuracy**

Camera location accuracy.

**Type** *Vector*

**location\_enabled**

Location enabled flag.

**Type** bool

**rotation**

Camera rotation angles.

**Type** *Vector*

**rotation\_accuracy**

Camera rotation accuracy.

**Type** *Vector*

**rotation\_enabled**

Rotation enabled flag.

**Type** bool

**class Type**

Camera type in [Regular, Keyframe]

**calibration**

Adjusted camera calibration including photo-invariant parameters.

**Type** *Calibration*

**center**

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

**Type** *Vector*

**chunk**

Chunk the camera belongs to.

**Type** *Chunk*

**enabled**

Enables/disables the photo.

**Type** bool

**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*

**image**()

Returns image data.

**Returns** Image data.

**Return type** *Image*

**key**

Camera identifier.

**Type** int

**label**

Camera label.

**Type** string

**layer\_index**

Camera layer index.

**Type** int

**location\_covariance**

Camera location covariance.

**Type** *Matrix*

**mask**

Camera mask.

**Type** *Mask*

**master**

Master camera.

**Type** *Camera*

**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.

**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** *Vector*

**reference**

Camera reference data.

Type *CameraReference*

**rotation\_covariance**

Camera rotation covariance.

Type *Matrix*

**selected**

Selects/deselects the photo.

Type *bool*

**sensor**

Camera sensor.

Type *Sensor*

**shutter**

Camera shutter.

Type *Shutter*

**thumbnail**

Camera thumbnail.

Type *Thumbnail*

**transform**

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

Type *Matrix*

**type**

Camera type.

Type *Camera.Type*

**unproject**(*point*)

Returns coordinates of the point which will have specified projected coordinates.

**Parameters** **point** (*Vector*) – Projection coordinates.

**Returns** 3D point coordinates.

**Return type** *Vector*

**vignetting**

Vignetting for each band.

Type list of *Vignetting*

**class** `Metashape.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]

**label**

Camera group label.

Type string

**selected**

Current selection state.

Type bool

**type**

Camera group type.

Type *CameraGroup.Type*

**class** `Metashape.CameraTrack`

Camera track.

**chunk**

Chunk the camera track belongs to.

Type *Chunk*

**duration**

Animation duration.

Type float

**field\_of\_view**

Vertical field of view in degrees.

Type float

**keyframes**

Camera track keyframes.



**Type** list of *Camera*

**label**

Animation label.

**Type** string

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

Load camera track from file.

**Parameters**

- **path** (*string*) – Path to camera track file
- **projection** (*CoordinateSystem*) – Camera track coordinate system.

**meta**

Camera track meta data.

**Type** *MetaData*

**save**(*path*[, *file\_format*, *max\_waypoints*, *projection* ])

Save camera track to file.

**Parameters**

- **path** (*string*) – Path to camera track file
- **file\_format** (*string*) – File format. “deduce”: - Deduce from extension, “path”: Path, “earth”: Google Earth KML, “pilot”: DJI Pilot KML, “trinity”: Asctec Trinity CSV, “autopilot”: Asctec Autopilot CSV, “litchi”: Litchi CSV
- **max\_waypoints** (*int*) – Max waypoints per flight
- **projection** (*CoordinateSystem*) – Camera track coordinate system.

**class Metashape.CamerasFormat**

Camera orientation format in [CamerasFormatXML, CamerasFormatCHAN, CamerasFormatBoujou, CamerasFormatBundler, CamerasFormatOPK, CamerasFormatPATB, CamerasFormatBINGO, CamerasFormatORIMA, CamerasFormatAeroSys, CamerasFormatInpho, CamerasFormatSummit, CamerasFormatBlocksExchange, CamerasFormatRZML, CamerasFormatVisionMap, CamerasFormatABC, CamerasFormatFBX, CamerasFormatNVM]

**class Metashape.Chunk**

A Chunk object:

- provides access to all chunk components (sensors, cameras, camera groups, markers, scale bars)
- 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 Metashape
>>> chunk = Metashape.app.document.chunk
>>> for frame in chunk.frames:
...     frame.matchPhotos(yscale=1)
>>> chunk.alignCameras()
```

**addCamera**([*sensor* ])

Add new camera to the chunk.

**Parameters** *sensor* (*Sensor*) – Sensor to be assigned to this camera.

**Returns** Created camera.

**Return type** *Camera*

**addCameraGroup**()

Add new camera group to the chunk.

**Returns** Created camera group.

**Return type** *CameraGroup*

**addCameraTrack**()

Add new camera track to the chunk.

**Returns** Created camera track.

**Return type** *CameraTrack*

**addDenseCloud**()

Add new dense cloud to the chunk.

**Returns** Created dense cloud.

**Return type** *DenseCloud*

**addDepthMaps**()

Add new depth maps set to the chunk.

**Returns** Created depth maps set.

**Return type** *DepthMaps*

**addElevation**()

Add new elevation model to the chunk.

**Returns** Created elevation model.

**Return type** *Elevation*

**addFrame**()

Add new frame to the chunk.

**Returns** Created frame.

**Return type** *Frame*

**addFrames**([*chunk* ][, *frames* ], *copy\_depth\_maps*=True, *copy\_dense\_cloud*=True, *copy\_model*=True, *copy\_tiled\_model*=True, *copy\_elevation*=True, *copy\_orthomosaic*=True[, *progress* ])

Add frames from specified chunk.

**Parameters**

- **chunk** (*int*) – Chunk to copy frames from.
- **frames** (*list of int*) – List of frame keys to copy.

- **copy\_depth\_maps** (*bool*) – Copy depth maps.
- **copy\_dense\_cloud** (*bool*) – Copy dense cloud.
- **copy\_model** (*bool*) – Copy model.
- **copy\_tiled\_model** (*bool*) – Copy tiled model.
- **copy\_elevation** (*bool*) – Copy DEM.
- **copy\_orthomosaic** (*bool*) – Copy orthomosaic.
- **progress** (*Callable[[float], None]*) – Progress callback.

**addMarker**(*[point]*, *visibility=False*)

Add new marker to the chunk.

**Parameters**

- **point** (*Vector*) – Point to initialize marker projections.
- **visibility** (*bool*) – Enables visibility check during projection assignment.

**Returns** Created marker.

**Return type** *Marker*

**addMarkerGroup**()

Add new marker group to the chunk.

**Returns** Created marker group.

**Return type** *MarkerGroup*

**addModel**()

Add new model to the chunk.

**Returns** Created model.

**Return type** *Model*

**addOrthomosaic**()

Add new orthomosaic to the chunk.

**Returns** Created orthomosaic.

**Return type** *Orthomosaic*

**addPhotos**(*[filenames]*, *[filegroups]*, *layout=UndefinedLayout*, *[group]*, *strip\_extensions=True*, *load\_reference=True*, *load\_xmp\_calibration=True*, *load\_xmp\_orientation=True*, *load\_xmp\_accuracy=False*, *load\_xmp\_antenna=True*, *load\_rpc\_txt=False*, *[progress]*)

Add a list of photos to the chunk.

**Parameters**

- **filenames** (*list of string*) – List of files to add.
- **filegroups** (*list of int*) – List of file groups.
- **layout** (*ImageLayout*) – Image layout.
- **group** (*int*) – Camera group key.
- **strip\_extensions** (*bool*) – Strip file extensions from camera labels.
- **load\_reference** (*bool*) – Load reference coordinates.
- **load\_xmp\_calibration** (*bool*) – Load calibration from XMP meta data.

- **load\_xmp\_orientation** (*bool*) – Load orientation from XMP meta data.
- **load\_xmp\_accuracy** (*bool*) – Load accuracy from XMP meta data.
- **load\_xmp\_antenna** (*bool*) – Load GPS/INS offset from XMP meta data.
- **load\_rpc\_txt** (*bool*) – Load satellite RPC data from auxiliary TXT files.
- **progress** (*Callable[[float], None]*) – Progress callback.

**addScalebar**(*point1*, *point2*)

Add new scale bar to the chunk.

**Parameters**

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

**Returns** Created scale bar.

**Return type** *Scalebar*

**addScalebarGroup**()

Add new scale bar group to the chunk.

**Returns** Created scale bar group.

**Return type** *ScalebarGroup*

**addSensor**([*source*])

Add new sensor to the chunk.

**Parameters** **source** (*Sensor*) – Sensor to copy parameters from.

**Returns** Created sensor.

**Return type** *Sensor*

**addTiledModel**()

Add new tiled model to the chunk.

**Returns** Created tiled model.

**Return type** *TiledModel*

**alignCameras**([*cameras*], *min\_image=2*, *adaptive\_fitting=False*, *reset\_alignment=False*,  
*subdivide\_task=True*[, *progress*])

Perform photo alignment for the chunk.

**Parameters**

- **cameras** (*list of int*) – List of cameras to align.
- **min\_image** (*int*) – Minimum number of point projections.
- **adaptive\_fitting** (*bool*) – Enable adaptive fitting of distortion coefficients.
- **reset\_alignment** (*bool*) – Reset current alignment.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **progress** (*Callable[[float], None]*) – Progress callback.

**analyzePhotos**([*cameras*], *filter\_mask=False*[, *progress*])

Estimate image quality.

**Parameters**

- **cameras** (*list of int*) – List of cameras to be analyzed.
- **filter\_mask** (*bool*) – Constrain analyzed image region by mask.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildContours**(*source\_data=ElevationData, interval=1, min\_value=-1e+10, max\_value=1e+10, prevent\_intersections=True[, progress]*)

Build contours for the chunk.

#### Parameters

- **source\_data** (*DataSource*) – Source data for contour generation.
- **interval** (*float*) – Contour interval.
- **min\_value** (*float*) – Minimum value of contour range.
- **max\_value** (*float*) – Maximum value of contour range.
- **prevent\_intersections** (*bool*) – Prevent contour intersections.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildDem**(*source\_data=DenseCloudData, interpolation=EnabledInterpolation[, projection][[, region][[, classes], flip\_x=False, flip\_y=False, flip\_z=False, resolution=0, subdivide\_task=True, workitem\_size\_tiles=10, max\_workgroup\_size=100[, progress]*)

Build elevation model for the chunk.

#### Parameters

- **source\_data** (*DataSource*) – Selects between dense point cloud and tie points.
- **interpolation** (*Interpolation*) – Interpolation mode.
- **projection** (*OrthoProjection*) – Output projection.
- **region** (*BBox*) – Region to be processed.
- **classes** (*list of int*) – List of dense point classes to be used for surface extraction.
- **flip\_x** (*bool*) – Flip X axis direction.
- **flip\_y** (*bool*) – Flip Y axis direction.
- **flip\_z** (*bool*) – Flip Z axis direction.
- **resolution** (*float*) – Output resolution in meters.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **workitem\_size\_tiles** (*int*) – Number of tiles in a workitem.
- **max\_workgroup\_size** (*int*) – Maximum workgroup size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildDenseCloud**(*point\_colors=True, point\_confidence=False, keep\_depth=True, max\_neighbors=100, subdivide\_task=True, workitem\_size\_cameras=20, max\_workgroup\_size=100[, progress]*)

Generate dense cloud for the chunk.

#### Parameters

- **point\_colors** (*bool*) – Enable point colors calculation.
- **point\_confidence** (*bool*) – Enable point confidence calculation.
- **keep\_depth** (*bool*) – Enable store depth maps option.

- **max\_neighbors** (*int*) – Maximum number of neighbor images to use for depth map filtering.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **workitem\_size\_cameras** (*int*) – Number of cameras in a workitem.
- **max\_workgroup\_size** (*int*) – Maximum workgroup size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildDepthMaps**(*downscale=4, filter\_mode=MildFiltering*[, *cameras* ], *reuse\_depth=False, max\_neighbors=16, subdivide\_task=True, workitem\_size\_cameras=20, max\_workgroup\_size=100*[, *progress* ])

Generate depth maps for the chunk.

#### Parameters

- **downscale** (*int*) – Depth map quality.
- **filter\_mode** (*FilterMode*) – Depth map filtering mode.
- **cameras** (*list of int*) – List of cameras to process.
- **reuse\_depth** (*bool*) – Enable reuse depth maps option.
- **max\_neighbors** (*int*) – Maximum number of neighbor images to use for depth map generation.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **workitem\_size\_cameras** (*int*) – Number of cameras in a workitem.
- **max\_workgroup\_size** (*int*) – Maximum workgroup size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildModel**(*surface\_type=Arbitrary, interpolation=EnabledInterpolation, face\_count=HighFaceCount, face\_count\_custom=200000, source\_data=DenseCloudData, vertex\_colors=True, vertex\_confidence=True, volumetric\_masks=False, keep\_depth=True, trimming\_radius=10*[, *cameras* ] [, *classes* ], *subdivide\_task=True, workitem\_size\_cameras=20, max\_workgroup\_size=100*[, *progress* ])

Generate model for the chunk frame.

#### Parameters

- **surface\_type** (*SurfaceType*) – Type of object to be reconstructed.
- **interpolation** (*Interpolation*) – Interpolation mode.
- **face\_count** (*FaceCount*) – Target face count.
- **face\_count\_custom** (*int*) – Custom face count.
- **source\_data** (*DataSource*) – Selects between dense point cloud, tie points and depth maps.
- **vertex\_colors** (*bool*) – Enable vertex colors calculation.
- **vertex\_confidence** (*bool*) – Enable vertex confidence calculation.
- **volumetric\_masks** (*bool*) – Enable strict volumetric masking.
- **keep\_depth** (*bool*) – Enable store depth maps option.
- **trimming\_radius** (*int*) – Trimming radius (no trimming if zero).
- **cameras** (*list of int*) – List of cameras to process.

- **classes** (*list of int*) – List of dense point classes to be used for surface extraction.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **workitem\_size\_cameras** (*int*) – Number of cameras in a workitem.
- **max\_workgroup\_size** (*int*) – Maximum workgroup size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildOrthomosaic**(*surface\_data=ModelData, blending\_mode=MosaicBlending, fill\_holes=True, ghosting\_filter=False, cull\_faces=False, refine\_seamlines=False*[[, *projection*]][, *region*], *resolution=0, resolution\_x=0, resolution\_y=0, flip\_x=False, flip\_y=False, flip\_z=False, subdivide\_task=True, workitem\_size\_cameras=20, workitem\_size\_tiles=10, max\_workgroup\_size=100*[[, *progress*]])

Build orthomosaic for the chunk.

#### Parameters

- **surface\_data** (*DataSource*) – Orthorectification surface.
- **blending\_mode** (*BlendingMode*) – Orthophoto blending mode.
- **fill\_holes** (*bool*) – Enable hole filling.
- **ghosting\_filter** (*bool*) – Enable ghosting filter.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **refine\_seamlines** (*bool*) – Refine seamlines based on image content.
- **projection** (*OrthoProjection*) – Output projection.
- **region** (*BBox*) – Region to be processed.
- **resolution** (*float*) – Pixel size in meters.
- **resolution\_x** (*float*) – Pixel size in the X dimension in projected units.
- **resolution\_y** (*float*) – Pixel size in the Y dimension in projected units.
- **flip\_x** (*bool*) – Flip X axis direction.
- **flip\_y** (*bool*) – Flip Y axis direction.
- **flip\_z** (*bool*) – Flip Z axis direction.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **workitem\_size\_cameras** (*int*) – Number of cameras in a workitem.
- **workitem\_size\_tiles** (*int*) – Number of tiles in a workitem.
- **max\_workgroup\_size** (*int*) – Maximum workgroup size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildPanorama**(*blending\_mode=MosaicBlending, ghosting\_filter=False*[[, *rotation*]][, *region*], *width=0, height=0*[[, *camera\_groups*]][, *frames*]][, *progress*])

Generate spherical panoramas from camera stations.

#### Parameters

- **blending\_mode** (*BlendingMode*) – Panorama blending mode.
- **ghosting\_filter** (*bool*) – Enable ghosting filter.
- **rotation** (*Matrix*) – Panorama 3x3 orientation matrix.

- **region** (*BBox*) – Region to be generated.
- **width** (*int*) – Width of output panorama.
- **height** (*int*) – Height of output panorama.
- **camera\_groups** (*list of int*) – List of camera groups to process.
- **frames** (*list of int*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildSeamlines**(*epsilon=1.5*[, *progress* ])

Generate shapes for orthomosaic seamlines.

#### Parameters

- **epsilon** (*float*) – Contour simplification threshold.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildTexture**(*blending\_mode=MosaicBlending*, *texture\_size=8192*, *fill\_holes=True*, *ghosting\_filter=True*[, *cameras* ], *texture\_type=DiffuseMap*[, *source\_model* ], *transfer\_texture=True*[, *progress* ])

Generate texture for the chunk.

#### Parameters

- **blending\_mode** (*BlendingMode*) – Texture blending mode.
- **texture\_size** (*int*) – Texture page size.
- **fill\_holes** (*bool*) – Enable hole filling.
- **ghosting\_filter** (*bool*) – Enable ghosting filter.
- **cameras** (*list of int*) – A list of cameras to be used for texturing.
- **texture\_type** (*Model.TextureType*) – Texture type.
- **source\_model** (*int*) – Source model.
- **transfer\_texture** (*bool*) – Transfer texture.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildTiledModel**(*pixel\_size=0*, *tile\_size=256*, *source\_data=DenseCloudData*, *face\_count=20000*, *ghosting\_filter=False*, *transfer\_texture=False*, *keep\_depth=True*, *merge=False*[, *operand\_chunk* ][, *operand\_frame* ][, *operand\_asset* ][, *classes* ], *subdivide\_task=True*, *workitem\_size\_cameras=20*, *max\_workgroup\_size=100*[, *progress* ])

Build tiled model for the chunk.

#### Parameters

- **pixel\_size** (*float*) – Target model resolution in meters.
- **tile\_size** (*int*) – Size of tiles in pixels.
- **source\_data** (*DataSource*) – Selects between dense point cloud and mesh.
- **face\_count** (*int*) – Number of faces per megapixel of texture resolution.
- **ghosting\_filter** (*bool*) – Enable ghosting filter.
- **transfer\_texture** (*bool*) – Transfer source model texture to tiled model.
- **keep\_depth** (*bool*) – Enable store depth maps option.
- **merge** (*bool*) – Merge tiled model flag.



- **operand\_chunk** (*int*) – Operand chunk key.
- **operand\_frame** (*int*) – Operand frame key.
- **operand\_asset** (*int*) – Operand asset key.
- **classes** (*list of int*) – List of dense point classes to be used for surface extraction.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **workitem\_size\_cameras** (*int*) – Number of cameras in a workitem.
- **max\_workgroup\_size** (*int*) – Maximum workgroup size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildUV**(*mapping\_mode=GenericMapping, page\_count=1, texture\_size=8192* [, *camera*] [, *progress* ])  
Generate uv mapping for the model.

#### Parameters

- **mapping\_mode** (*MappingMode*) – Texture mapping mode.
- **page\_count** (*int*) – Number of texture pages to generate.
- **texture\_size** (*int*) – Expected size of texture page at texture generation step.
- **camera** (*int*) – Camera to be used for texturing in MappingCamera mode.
- **progress** (*Callable[[float], None]*) – Progress callback.

**calculatePointNormals**(*point\_neighbors=28* [, *progress* ])  
Calculate dense cloud normals.

#### Parameters

- **point\_neighbors** (*int*) – Number of point neighbors to use for normal estimation.
- **progress** (*Callable[[float], None]*) – Progress callback.

**calibrateColors**(*source\_data=ModelData, white\_balance=False* [, *cameras*] [, *progress* ])  
Perform radiometric calibration.

#### Parameters

- **source\_data** (*DataSource*) – Source data for calibration.
- **white\_balance** (*bool*) – Calibrate white balance.
- **cameras** (*list of int*) – List of cameras to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**calibrateReflectance**(*use\_reflectance\_panels=True, use\_sun\_sensor=False* [, *progress* ])  
Calibrate reflectance factors based on calibration panels and/or sun sensor.

#### Parameters

- **use\_reflectance\_panels** (*bool*) – Use calibrated reflectance panels.
- **use\_sun\_sensor** (*bool*) – Apply irradiance sensor measurements.
- **progress** (*Callable[[float], None]*) – Progress callback.

**camera\_crs**

Coordinate system used for camera reference data.

Type *CoordinateSystem*

**camera\_groups**

List of camera groups in the chunk.

Type list of *CameraGroup*

**camera\_location\_accuracy**

Expected accuracy of camera coordinates in meters.

Type *Vector*

**camera\_rotation\_accuracy**

Expected accuracy of camera orientation angles in degrees.

Type *Vector*

**camera\_track**

Camera track.

Type *CameraTrack*

**camera\_tracks**

List of camera tracks in the chunk.

Type list of *CameraTrack*

**cameras**

List of Regular and Keyframe cameras in the chunk.

Type list of *Camera*

**cir\_transform**

CIR calibration matrix.

Type *CirTransform*

**colorizeDenseCloud**(*source\_data=ImagesData*[, *progress* ])

Calculate point colors for the dense cloud.

**Parameters**

- **source\_data** (*DataSource*) – Source data to extract colors from.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**colorizeModel**(*source\_data=ImagesData*[, *progress* ])

Calculate vertex colors for the model.

**Parameters**

- **source\_data** (*DataSource*) – Source data to extract colors from.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**copy**([*frames* ][, *items* ], *keypoints=True*[, *progress* ])

Make a copy of the chunk.

**Parameters**

- **frames** (list of *Frame*) – Optional list of frames to be copied.
- **items** (list of *DataSource*) – A list of items to copy.
- **keypoints** (*bool*) – copy key points data.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**Returns** Copy of the chunk.

**Return type** *Chunk*

**crs**

Coordinate system used for reference data.

**Type** *CoordinateSystem*

**decimateModel**(*face\_count=200000*[], *asset*[], *apply\_to\_selection=False*[], *progress*[])

Decimate the model to the specified face count.

**Parameters**

- **face\_count** (*int*) – Target face count.
- **asset** (*int*) – Model to process.
- **apply\_to\_selection** (*bool*) – Apply to selection.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**dense\_cloud**

Default dense point cloud for the current frame.

**Type** *DenseCloud*

**dense\_clouds**

List of dense clouds for the current frame.

**Type** list of *DenseCloud*

**depth\_maps**

Default depth maps set for the current frame.

**Type** *DepthMaps*

**depth\_maps\_sets**

List of depth maps sets for the current frame.

**Type** list of *DepthMaps*

**detectFiducials**(*generate\_masks=False*[], *cameras*][[], *frames*][[], *progress*])

Detect fiducial marks on film cameras.

**Parameters**

- **generate\_masks** (*bool*) – Generate background masks.
- **cameras** (*list of int*) – List of cameras to process.
- **frames** (*list of int*) – List of frames to process.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**detectMarkers**(*target\_type=CircularTarget12bit*, *tolerance=50*, *filter\_mask=False*, *inverted=False*, *noparity=False*, *maximum\_residual=5*, *minimum\_size=0*, *minimum\_dist=5*[], *cameras*][[], *frames*][[], *progress*])

Create markers from coded targets.

**Parameters**

- **target\_type** (*TargetType*) – Type of targets.
- **tolerance** (*int*) – Detector tolerance (0 - 100).
- **filter\_mask** (*bool*) – Ignore masked image regions.
- **inverted** (*bool*) – Detect markers on black background.

- **noparity** (*bool*) – Disable parity checking.
- **maximum\_residual** (*float*) – Maximum residual for non-coded targets in pixels.
- **minimum\_size** (*int*) – Minimum target radius in pixels to be detected (CrossTarget type only).
- **minimum\_dist** (*int*) – Minimum distance between targets in pixels (CrossTarget type only).
- **cameras** (*list of int*) – List of cameras to process.
- **frames** (*list of int*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**elevation**

Default elevation model for the current frame.

Type *Elevation*

**elevations**

List of elevation models for the current frame.

Type list of *Elevation*

**enabled**

Enables/disables the chunk.

Type *bool*

**euler\_angles**

Euler angles triplet used for rotation reference.

Type *EulerAngles*

**exportCameras**(*path=""*, *format=CamerasFormatXML*[, *crs* ], *save\_points=True*, *save\_markers=False*, *save\_invalid\_matches=False*, *use\_labels=False*, *use\_initial\_calibration=False*, *image\_orientation=0*, *chan\_rotation\_order=RotationOrderXYZ*, *binary=False*, *bundler\_save\_list=True*, *bundler\_path\_list='list.txt'*, *bingo\_save\_image=True*, *bingo\_save\_itera=True*, *bingo\_save\_geoin=True*, *bingo\_save\_gps=False*, *bingo\_path\_itera='itera.dat'*, *bingo\_path\_image='image.dat'*, *bingo\_path\_geoin='geoin.dat'*, *bingo\_path\_gps='gps-imu.dat'*[, *progress* ])

Export point cloud and/or camera positions.

**Parameters**

- **path** (*string*) – Path to output file.
- **format** (*CamerasFormat*) – Export format.
- **crs** (*CoordinateSystem*) – Output coordinate system.
- **save\_points** (*bool*) – Enables/disables export of automatic tie points.
- **save\_markers** (*bool*) – Enables/disables export of manual matching points.
- **save\_invalid\_matches** (*bool*) – Enables/disables export of invalid image matches.
- **use\_labels** (*bool*) – Enables/disables label based item identifiers.
- **use\_initial\_calibration** (*bool*) – Transform image coordinates to initial calibration.
- **image\_orientation** (*int*) – Image coordinate system (0 - X right, 1 - X up, 2 - X left, 3 - X down).
- **chan\_rotation\_order** (*RotationOrder*) – Rotation order (CHAN format only).

- **binary** (*bool*) – Enables/disables binary encoding for selected format (if applicable).
- **bundler\_save\_list** (*bool*) – Enables/disables export of Bundler image list file.
- **bundler\_path\_list** (*string*) – Path to Bundler image list file.
- **bingo\_save\_image** (*bool*) – Enables/disables export of BINGO IMAGE COORDINATE file.
- **bingo\_save\_itera** (*bool*) – Enables/disables export of BINGO ITERA file.
- **bingo\_save\_geoin** (*bool*) – Enables/disables export of BINGO GEO INPUT file.
- **bingo\_save\_gps** (*bool*) – Enables/disables export of BINGO GPS/IMU data.
- **bingo\_path\_itera** (*string*) – Path to BINGO ITERA file.
- **bingo\_path\_image** (*string*) – Path to BINGO IMAGE COORDINATE file.
- **bingo\_path\_geoin** (*string*) – Path to BINGO GEO INPUT file.
- **bingo\_path\_gps** (*string*) – Path to BINGO GPS/IMU file.
- **progress** (*Callable[[float], None]*) – Progress callback.

**exportMarkers**(*path*="[ , crs ], *binary*=False[ , *progress* ])

Export markers.

#### Parameters

- **path** (*string*) – Path to output file.
- **crs** (*CoordinateSystem*) – Output coordinate system.
- **binary** (*bool*) – Enables/disables binary encoding for selected format (if applicable).
- **progress** (*Callable[[float], None]*) – Progress callback.

**exportModel**(*path*=" , *binary*=True, *precision*=6, *texture\_format*=ImageFormatJPEG, *save\_texture*=True, *save\_uv*=True, *save\_normals*=True, *save\_colors*=True, *save\_confidence*=False, *save\_cameras*=True, *save\_markers*=True, *save\_udim*=False, *save\_alpha*=False, *embed\_texture*=False, *strip\_extensions*=False, *raster\_transform*=RasterTransformNone, *colors\_rgb\_8bit*=True, *comment*=" , *save\_comment*=True, *format*=ModelFormatNone[ , *crs* ] [ , *shift* ], *clip\_to\_boundary*=True[ , *viewpoint* ] [ , *progress* ])

Export generated model for the chunk.

#### Parameters

- **path** (*string*) – Path to output model.
- **binary** (*bool*) – Enables/disables binary encoding (if supported by format).
- **precision** (*int*) – Number of digits after the decimal point (for text formats).
- **texture\_format** (*ImageFormat*) – Texture format.
- **save\_texture** (*bool*) – Enables/disables texture export.
- **save\_uv** (*bool*) – Enables/disables uv coordinates export.
- **save\_normals** (*bool*) – Enables/disables export of vertex normals.
- **save\_colors** (*bool*) – Enables/disables export of vertex colors.
- **save\_confidence** (*bool*) – Enables/disables export of vertex confidence.
- **save\_cameras** (*bool*) – Enables/disables camera export.
- **save\_markers** (*bool*) – Enables/disables marker export.

- **save\_udim** (*bool*) – Enables/disables UDIM texture layout.
- **save\_alpha** (*bool*) – Enables/disables alpha channel export.
- **embed\_texture** (*bool*) – Embeds texture inside the model file (if supported by format).
- **strip\_extensions** (*bool*) – Strips camera label extensions during export.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.
- **colors\_rgb\_8bit** (*bool*) – Convert colors to 8 bit RGB.
- **comment** (*string*) – Optional comment (if supported by selected format).
- **save\_comment** (*bool*) – Enables/disables comment export.
- **format** (*ModelFormat*) – Export format.
- **crs** (*CoordinateSystem*) – Output coordinate system.
- **shift** (*Vector*) – Optional shift to be applied to vertex coordinates.
- **clip\_to\_boundary** (*bool*) – Clip model to boundary shapes.
- **viewpoint** (*Viewpoint*) – Default view.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
exportOrthophotos(path='{filename}.tif'[, cameras ], raster_transform=RasterTransformNone[, projection
    ][, region ], resolution=0, resolution_x=0, resolution_y=0, save_kml=False,
    save_world=False, save_alpha=True[, image_compression ], white_background=True,
    north_up=True[, progress ])
```

Export orthophotos for the chunk.

#### Parameters

- **path** (*string*) – Path to output orthophoto.
- **cameras** (*list of int*) – List of cameras to process.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.
- **projection** (*OrthoProjection*) – Output projection.
- **region** (*BBox*) – Region to be exported.
- **resolution** (*float*) – Output resolution in meters.
- **resolution\_x** (*float*) – Pixel size in the X dimension in projected units.
- **resolution\_y** (*float*) – Pixel size in the Y dimension in projected units.
- **save\_kml** (*bool*) – Enable kml file generation.
- **save\_world** (*bool*) – Enable world file generation.
- **save\_alpha** (*bool*) – Enable alpha channel generation.
- **image\_compression** (*ImageCompression*) – Image compression parameters.
- **white\_background** (*bool*) – Enable white background.
- **north\_up** (*bool*) – Use north-up orientation for export.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
exportPoints(path="", source_data=DenseCloudData, binary=True, save_normals=True,
save_colors=True, save_classes=True, save_confidence=True,
raster_transform=RasterTransformNone, colors_rgb_8bit=True, comment="",
save_comment=True, format=PointsFormatNone, image_format=ImageFormatJPEG[, crs
][, shift ][, region ], clip_to_boundary=True, block_width=1000, block_height=1000,
split_in_blocks=False[, classes ], save_images=False, compression=True,
screen_space_error=16[, viewpoint ], subdivide_task=True[, progress ])
```

Export point cloud.

### Parameters

- **path** (*string*) – Path to output file.
- **source\_data** (*DataSource*) – Selects between dense point cloud and tie points. If not specified, uses dense cloud if available.
- **binary** (*bool*) – Enables/disables binary encoding for selected format (if applicable).
- **save\_normals** (*bool*) – Enables/disables export of point normals.
- **save\_colors** (*bool*) – Enables/disables export of point colors.
- **save\_classes** (*bool*) – Enables/disables export of point classes.
- **save\_confidence** (*bool*) – Enables/disables export of point confidence.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.
- **colors\_rgb\_8bit** (*bool*) – Convert colors to 8 bit RGB.
- **comment** (*string*) – Optional comment (if supported by selected format).
- **save\_comment** (*bool*) – Enable comment export.
- **format** (*PointsFormat*) – Export format.
- **image\_format** (*ImageFormat*) – Image data format.
- **crs** (*CoordinateSystem*) – Output coordinate system.
- **shift** (*Vector*) – Optional shift to be applied to point coordinates.
- **region** (*BBox*) – Region to be exported.
- **clip\_to\_boundary** (*bool*) – Clip point cloud to boundary shapes.
- **block\_width** (*float*) – Block width in meters.
- **block\_height** (*float*) – Block height in meters.
- **split\_in\_blocks** (*bool*) – Enable tiled export.
- **classes** (*list of int*) – List of dense point classes to be exported.
- **save\_images** (*bool*) – Enable image export.
- **compression** (*bool*) – Enable compression (Cesium format only).
- **screen\_space\_error** (*float*) – Target screen space error (Cesium format only).
- **viewpoint** (*Viewpoint*) – Default view.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
exportRaster(path="",format=RasterFormatTiles,image_format=ImageFormatNone,
raster_transform=RasterTransformNone[, projection ][, region ],resolution=0,
resolution_x=0,resolution_y=0,block_width=10000,block_height=10000,
split_in_blocks=False,width=0,height=0[, world_transform ],nodata_value=-32767,
save_kml=False,save_world=False,save_scheme=False,save_alpha=True,
image_description="[, image_compression ],network_links=True,min_zoom_level=-1,
max_zoom_level=-1,white_background=True,clip_to_boundary=True,title='Orthomosaic',
description='Generated by Agisoft Metashape',source_data=OrthomosaicData,
north_up=True,tile_width=256,tile_height=256[, progress ])
```

Export DEM or orthomosaic to file.

### Parameters

- **path** (*string*) – Path to output orthomosaic.
- **format** (*RasterFormat*) – Export format.
- **image\_format** (*ImageFormat*) – Tile format.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.
- **projection** (*OrthoProjection*) – Output projection.
- **region** (*BBox*) – Region to be exported.
- **resolution** (*float*) – Output resolution in meters.
- **resolution\_x** (*float*) – Pixel size in the X dimension in projected units.
- **resolution\_y** (*float*) – Pixel size in the Y dimension in projected units.
- **block\_width** (*int*) – Raster block width in pixels.
- **block\_height** (*int*) – Raster block height in pixels.
- **split\_in\_blocks** (*bool*) – Split raster in blocks.
- **width** (*int*) – Raster width.
- **height** (*int*) – Raster height.
- **world\_transform** (*Matrix*) – 2x3 raster-to-world transformation matrix.
- **nodata\_value** (*float*) – No-data value (DEM export only).
- **save\_kml** (*bool*) – Enable kml file generation.
- **save\_world** (*bool*) – Enable world file generation.
- **save\_scheme** (*bool*) – Enable tile scheme files generation.
- **save\_alpha** (*bool*) – Enable alpha channel generation.
- **image\_description** (*string*) – Optional description to be added to image files.
- **image\_compression** (*ImageCompression*) – Image compression parameters.
- **network\_links** (*bool*) – Enable network links generation for KMZ format.
- **min\_zoom\_level** (*int*) – Minimum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).
- **max\_zoom\_level** (*int*) – Maximum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).
- **white\_background** (*bool*) – Enable white background.
- **clip\_to\_boundary** (*bool*) – Clip raster to boundary shapes.



- **title** (*string*) – Export title.
- **description** (*string*) – Export description.
- **source\_data** (*DataSource*) – Selects between DEM and orthomosaic.
- **north\_up** (*bool*) – Use north-up orientation for export.
- **tile\_width** (*int*) – Tile width in pixels.
- **tile\_height** (*int*) – Tile height in pixels.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**exportReference**(*path*="", *format*=*ReferenceFormatNone*, *items*=*ReferenceItemsCameras*, *columns*="", *delimiter*=' ', *precision*=6[, *progress* ])

Export reference data to the specified file.

#### Parameters

- **path** (*string*) – Path to the output file.
- **format** (*ReferenceFormat*) – Export format.
- **items** (*ReferenceItems*) – Items to export in CSV format.
- **columns** (*string*) – Column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, u/v/w - estimated coordinates, U/V/W - coordinate errors, d/e/f - estimated orientation angles, D/E/F - orientation errors, p/q/r - estimated coordinates variance, i/j/k - estimated orientation angles variance, [] - group of multiple values, | - column separator within group).
- **delimiter** (*string*) – Column delimiter in csv format.
- **precision** (*int*) – Number of digits after the decimal point (for CSV format).
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**exportReport**(*path*="", *title*="", *description*="", *font\_size*=12, *page\_numbers*=*True*, *include\_system\_info*=*True*[, *user\_settings* ][, *progress* ])

Export processing report in PDF format.

#### Parameters

- **path** (*string*) – Path to output report.
- **title** (*string*) – Report title.
- **description** (*string*) – Report description.
- **font\_size** (*int*) – Font size (pt).
- **page\_numbers** (*bool*) – Enable page numbers.
- **include\_system\_info** (*bool*) – Include system information.
- **user\_settings** (*list of (string, string) tuples*) – A list of user defined settings to include on the Processing Parameters page.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**exportShapes**(*path*="", *save\_points*=*False*, *save\_polylines*=*False*, *save\_polygons*=*False*[, *groups* ], *format*=*ShapesFormatNone*[, *crs* ][, *shift* ], *polygons\_as\_polylines*=*False*, *save\_labels*=*True*, *save\_attributes*=*True*[, *progress* ])

Export shapes layer to file.

#### Parameters

- **path** (*string*) – Path to shape file.
- **save\_points** (*bool*) – Export points.
- **save\_polylines** (*bool*) – Export polylines.
- **save\_polygons** (*bool*) – Export polygons.
- **groups** (*list of int*) – A list of shape groups to export.
- **format** (*ShapesFormat*) – Export format.
- **crs** (*CoordinateSystem*) – Output coordinate system.
- **shift** (*Vector*) – Optional shift to be applied to vertex coordinates.
- **polygons\_as\_polylines** (*bool*) – Save polygons as polylines.
- **save\_labels** (*bool*) – Export labels.
- **save\_attributes** (*bool*) – Export attributes.
- **progress** (*Callable[[float], None]*) – Progress callback.

**exportTexture**(*path=""*, *texture\_type=DiffuseMap*, *save\_alpha=False* [, *progress* ])  
Export model texture to file.

#### Parameters

- **path** (*string*) – Path to output file.
- **texture\_type** (*Model.TextureType*) – Texture type.
- **save\_alpha** (*bool*) – Enable alpha channel export.
- **progress** (*Callable[[float], None]*) – Progress callback.

**exportTiledModel**(*path=""*, *format=TiledModelFormatNone*, *model\_format=ModelFormatCOLLADA*,  
*texture\_format=ImageFormatJPEG*, *raster\_transform=RasterTransformNone* [,  
*image\_compression* ] [, *crs* ], *clip\_to\_boundary=True*, *model\_compression=True*,  
*screen\_space\_error=16* [, *progress* ])

Export generated tiled model for the chunk.

#### Parameters

- **path** (*string*) – Path to output model.
- **format** (*TiledModelFormat*) – Export format.
- **model\_format** (*ModelFormat*) – Model format for zip export.
- **texture\_format** (*ImageFormat*) – Texture format.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.
- **image\_compression** (*ImageCompression*) – Image compression parameters.
- **crs** (*CoordinateSystem*) – Output coordinate system.
- **clip\_to\_boundary** (*bool*) – Clip tiled model to boundary shapes.
- **model\_compression** (*bool*) – Enable mesh compression (Cesium format only).
- **screen\_space\_error** (*float*) – Target screen space error (Cesium format only).
- **progress** (*Callable[[float], None]*) – Progress callback.

**filterDenseCloud**(*point\_spacing=0* [, *asset* ] [, *progress* ])  
Reduce dense cloud points number.

**Parameters**

- **point\_spacing** (*float*) – Desired point spacing (m).
- **asset** (*int*) – Dense cloud key to filter.
- **progress** (*Callable[[float], None]*) – Progress callback.

**findCamera**(*key*)

Find camera by its key.

**Returns** Found camera.

**Return type** *Camera*

**findCameraGroup**(*key*)

Find camera group by its key.

**Returns** Found camera group.

**Return type** *CameraGroup*

**findCameraTrack**(*key*)

Find camera track by its key.

**Returns** Found camera track.

**Return type** *CameraTrack*

**findDenseCloud**(*key*)

Find dense cloud by its key.

**Returns** Found dense cloud.

**Return type** *DenseCloud*

**findDepthMaps**(*key*)

Find depth maps by its key.

**Returns** Found depth maps.

**Return type** *DepthMaps*

**findElevation**(*key*)

Find elevation model by its key.

**Returns** Found elevation model.

**Return type** *Elevation*

**findFrame**(*key*)

Find frame by its key.

**Returns** Found frame.

**Return type** *Chunk*

**findMarker**(*key*)

Find marker by its key.

**Returns** Found marker.

**Return type** *Marker*

**findMarkerGroup**(*key*)

Find marker group by its key.

**Returns** Found marker group.

**Return type** *MarkerGroup*

**findModel**(*key*)

Find model by its key.

**Returns** Found model.

**Return type** *Model*

**findOrthomosaic**(*key*)

Find orthomosaic by its key.

**Returns** Found orthomosaic.

**Return type** *Orthomosaic*

**findScalebar**(*key*)

Find scalebar by its key.

**Returns** Found scalebar.

**Return type** *Scalebar*

**findScalebarGroup**(*key*)

Find scalebar group by its key.

**Returns** Found scalebar group.

**Return type** *ScalebarGroup*

**findSensor**(*key*)

Find sensor by its key.

**Returns** Found sensor.

**Return type** *Sensor*

**findTiledModel**(*key*)

Find tiled model by its key.

**Returns** Found tiled model.

**Return type** *TiledModel*

**frame**

Current frame index.

**Type** int

**frames**

List of frames in the chunk.

**Type** list of Frame

**generateMasks**(*path*='{filename}\_mask.png', *masking\_mode*=*MaskingModeAlpha*,  
*mask\_operation*=*MaskOperationReplacement*, *tolerance*=10[, *cameras* ],  
*mask\_defocus*=*False*, *fix\_coverage*=*True*, *blur\_threshold*=3,  
*depth\_threshold*=3.40282e+38[, *progress* ])

Generate masks for multiple cameras.

**Parameters**

- **path** (*string*) – Mask file name template.
- **masking\_mode** (*MaskingMode*) – Mask generation mode.
- **mask\_operation** (*MaskOperation*) – Mask operation.

- **tolerance** (*int*) – Background masking tolerance.
- **cameras** (*list of int*) – Optional list of cameras to be processed.
- **mask\_defocus** (*bool*) – Mask defocus areas.
- **fix\_coverage** (*bool*) – Extend masks to cover whole mesh (only if mask\_defocus=True).
- **blur\_threshold** (*float*) – Allowed blur radius on a photo in pix (only if mask\_defocus=True).
- **depth\_threshold** (*float*) – Maximum depth of masked areas in meters (only if mask\_defocus=False).
- **progress** (*Callable[[float], None]*) – Progress callback.

```
generatePrescriptionMap(class_count=4, cell_size=1,
                        classification_method=JenksNaturalBreaksClassification[,
                        boundary_shape_group ][, breakpoints ][, rates ][, progress ])
```

Generate prescription map for orthomosaic.

#### Parameters

- **class\_count** (*int*) – Number of classes.
- **cell\_size** (*float*) – Step of prescription grid, meters.
- **classification\_method** (*ClassificationMethod*) – Index values classification method.
- **boundary\_shape\_group** (*int*) – Boundary shape group.
- **breakpoints** (*list of float*) – Classification breakpoints.
- **rates** (*list of float*) – Fertilizer rate for each class.
- **progress** (*Callable[[float], None]*) – Progress callback.

#### **image\_brightness**

Image brightness as percentage.

**Type** float

#### **image\_contrast**

Image contrast as percentage.

**Type** float

```
importCameras(path="", format=CamerasFormatXML[, crs ], image_orientation=0, image_list='list.txt',
                load_image_list=False[, progress ])
```

Import camera positions.

#### Parameters

- **path** (*string*) – Path to the file.
- **format** (*CamerasFormat*) – File format.
- **crs** (*CoordinateSystem*) – Ground coordinate system.
- **image\_orientation** (*int*) – Image coordinate system (0 - X right, 1 - X up, 2 - X left, 3 - X down).
- **image\_list** (*string*) – Path to image list file (Bundler format only).
- **load\_image\_list** (*bool*) – Enable Bundler image list import.
- **progress** (*Callable[[float], None]*) – Progress callback.

**importLaserScans**(*format=PointsFormatNone*[, *filenames* ], *image\_path*=""[, *progress* ])  
Import cameras with depth data.

**Parameters**

- **format** (*PointsFormat*) – Point cloud format.
- **filenames** (*list of string*) – List of files to import.
- **image\_path** (*string*) – Path template to output files.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**importMarkers**(*path*=""[, *progress* ])  
Import markers.

**Parameters**

- **path** (*string*) – Path to the file.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**importModel**(*path*="", *format=ModelFormatNone*[, *crs* ][, *shift* ], *decode\_udim=True*[, *progress* ])  
Import model from file.

**Parameters**

- **path** (*string*) – Path to model.
- **format** (*ModelFormat*) – Model format.
- **crs** (*CoordinateSystem*) – Model coordinate system.
- **shift** (*Vector*) – Optional shift to be applied to vertex coordinates.
- **decode\_udim** (*bool*) – Load UDIM texture layout.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**importPoints**(*path*="", *format=PointsFormatNone*, *calculate\_normals=True*[, *crs* ][, *shift* ],  
*point\_neighbors=28*[, *progress* ])  
Import point cloud from file.

**Parameters**

- **path** (*string*) – Path to point cloud.
- **format** (*PointsFormat*) – Point cloud format.
- **calculate\_normals** (*bool*) – Calculate point normals.
- **crs** (*CoordinateSystem*) – Point cloud coordinate system.
- **shift** (*Vector*) – Optional shift to be applied to point coordinates.
- **point\_neighbors** (*int*) – Number of point neighbors to use for normal estimation.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**importRaster**(*path*=""[, *crs* ], *raster\_type=ElevationData*, *nodata\_value=-32767*,  
*has\_nodata\_value=False*[, *progress* ])  
Import DEM or orthomosaic from file.

**Parameters**

- **path** (*string*) – Path to elevation model in GeoTIFF format.
- **crs** (*CoordinateSystem*) – Default coordinate system if not specified in GeoTIFF file.

- **raster\_type** (*DataSource*) – Type of raster layer to import.
- **nodata\_value** (*float*) – No-data value.
- **has\_nodata\_value** (*bool*) – No-data value valid flag.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
importReference(path="", format=ReferenceFormatCSV, columns="", delimiter="", group_delimiters=False,
                skip_rows=0[, items][[, crs ]], ignore_labels=False, create_markers=False,
                threshold=0.1, shutter_lag=0[, progress ])
```

Import reference data from the specified file.

#### Parameters

- **path** (*string*) – Path to the file with reference data.
- **format** (*ReferenceFormat*) – File format.
- **columns** (*string*) – Column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, [] - group of multiple values, | - column separator within group).
- **delimiter** (*string*) – Column delimiter in csv format.
- **group\_delimiters** (*bool*) – Combine consecutive delimiters in csv format.
- **skip\_rows** (*int*) – Number of rows to skip in (csv format only).
- **items** (*ReferenceItems*) – List of items to load reference for (csv format only).
- **crs** (*CoordinateSystem*) – Reference data coordinate system (csv format only).
- **ignore\_labels** (*bool*) – Matches reference data based on coordinates alone (csv format only).
- **create\_markers** (*bool*) – Create markers for missing entries (csv format only).
- **threshold** (*float*) – Error threshold in meters used when ignore\_labels is set (csv format only).
- **shutter\_lag** (*float*) – Shutter lag in seconds (APM format only).
- **progress** (*Callable[[float], None]*) – Progress callback.

```
importShapes(path="", replace=False, boundary_type=NoBoundary, format=ShapesFormatNone,
              columns='nxyzd', delimiter=', ', group_delimiters=False, skip_rows=0[, crs ][, progress ])
```

Import shapes layer from file.

#### Parameters

- **path** (*string*) – Path to shape file.
- **replace** (*bool*) – Replace current shapes with new data.
- **boundary\_type** (*Shape.BoundaryType*) – Boundary type to be applied to imported shapes.
- **format** (*ShapesFormat*) – Shapes format.
- **columns** (*string*) – Column order in csv format (n - label, x/y/z - coordinates, d - description, [] - group of multiple values, | - column separator within group).
- **delimiter** (*string*) – Column delimiter in csv format.
- **group\_delimiters** (*bool*) – Combine consecutive delimiters in csv format.
- **skip\_rows** (*int*) – Number of rows to skip in (csv format only).

- **crs** (*CoordinateSystem*) – Reference data coordinate system (csv format only).
- **progress** (*Callable[[float], None]*) – Progress callback.

**importTiledModel**(*path*="[, *progress* ]

Import tiled model from file.

**Parameters**

- **path** (*string*) – Path to tiled model.
- **progress** (*Callable[[float], None]*) – Progress callback.

**importVideo**(*path, image\_path, frame\_step=CustomFrameStep, custom\_frame\_step=1, time\_start=0, time\_end=-1*)

Imports video to active chunk.

**Parameters**

- **path** (*string*) – Path to source video.
- **image\_path** (*string*) – Path to directory where to save frames with filename template. For example: `/path/to/dir/frame{filenum}.png`.
- **frame\_step** (*FrameStep*) – Frame step type.
- **custom\_frame\_step** (*int*) – Every `custom_frame_step`'th frame will be saved. Used for `frame_step=CustomFrameStep`.
- **time\_start** (*int*) – The starting point for importing video, in milliseconds.
- **time\_end** (*int*) – The endpoint for importing video, in milliseconds.

**key**

Chunk identifier.

**Type** *int*

**label**

Chunk label.

**Type** *string*

**loadReferenceExif**(*load\_rotation=False, load\_accuracy=False*)

Import camera locations from EXIF meta data.

**Parameters**

- **load\_rotation** (*bool*) – load yaw, pitch and roll orientation angles.
- **load\_accuracy** (*bool*) – load camera location accuracy.

**loadReflectancePanelCalibration**(*path[, cameras]*)

Load reflectance panel calibration from CSV file.

**Parameters**

- **path** (*string*) – Path to calibration file.
- **cameras** (list of *Camera*) – List of cameras to process.

**locateReflectancePanels**(*[progress]*)

Locate reflectance panels based on QR-codes.

**Parameters** **progress** (*Callable[[float], None]*) – Progress callback.

**marker\_crs**

Coordinate system used for marker reference data.



Type *CoordinateSystem*

**marker\_groups**

List of marker groups in the chunk.

Type list of *MarkerGroup*

**marker\_location\_accuracy**

Expected accuracy of marker coordinates in meters.

Type *Vector*

**marker\_projection\_accuracy**

Expected accuracy of marker projections in pixels.

Type float

**markers**

List of Regular, Vertex and Fiducial markers in the chunk.

Type list of *Marker*

**masks**

Image masks.

Type *Masks*

**matchPhotos**(*downscale=1, generic\_preselection=True, reference\_preselection=True, reference\_preselection\_mode=ReferencePreselectionSource, filter\_mask=False, mask\_tiepoints=True, filter\_stationary\_points=True, keypoint\_limit=40000, keypoint\_limit\_per\_mpx=1000, tiepoint\_limit=4000, keep\_keypoints=False[, pairs][[, cameras]*, *guided\_matching=False, reset\_matches=False, subdivide\_task=True, workitem\_size\_cameras=20, workitem\_size\_pairs=80, max\_workgroup\_size=100[, progress]*)

Perform image matching for the chunk frame.

**Parameters**

- **downscale** (*int*) – Image alignment accuracy.
- **generic\_preselection** (*bool*) – Enable generic preselection.
- **reference\_preselection** (*bool*) – Enable reference preselection.
- **reference\_preselection\_mode** (*ReferencePreselectionMode*) – Reference preselection mode.
- **filter\_mask** (*bool*) – Filter points by mask.
- **mask\_tiepoints** (*bool*) – Apply mask filter to tie points.
- **filter\_stationary\_points** (*bool*) – Exclude tie points which are stationary across images.
- **keypoint\_limit** (*int*) – Key point limit.
- **keypoint\_limit\_per\_mpx** (*int*) – Key point limit per megapixel.
- **tiepoint\_limit** (*int*) – Tie point limit.
- **keep\_keypoints** (*bool*) – Store keypoints in the project.
- **pairs** (*list of (int, int) tuples*) – User defined list of camera pairs to match.
- **cameras** (*list of int*) – List of cameras to match.
- **guided\_matching** (*bool*) – Enable guided image matching.

- **reset\_matches** (*bool*) – Reset current matches.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **workitem\_size\_cameras** (*int*) – Number of cameras in a workitem.
- **workitem\_size\_pairs** (*int*) – Number of image pairs in a workitem.
- **max\_workgroup\_size** (*int*) – Maximum workgroup size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**meta**

Chunk meta data.

Type *MetaData*

**model**

Default model for the current frame.

Type *Model*

**models**

List of models for the current frame.

Type list of *Model*

**modified**

Modified flag.

Type *bool*

**optimizeCameras**(*fit\_f=True, fit\_cx=True, fit\_cy=True, fit\_b1=False, fit\_b2=False, fit\_k1=True, fit\_k2=True, fit\_k3=True, fit\_k4=False, fit\_p1=True, fit\_p2=True, fit\_corrections=False, adaptive\_fitting=False, tiepoint\_covariance=False[, progress]*)

Perform optimization of point cloud / camera parameters.

**Parameters**

- **fit\_f** (*bool*) – Enable optimization of focal length coefficient.
- **fit\_cx** (*bool*) – Enable optimization of X principal point coordinates.
- **fit\_cy** (*bool*) – Enable optimization of Y principal point coordinates.
- **fit\_b1** (*bool*) – Enable optimization of aspect ratio.
- **fit\_b2** (*bool*) – Enable optimization of skew coefficient.
- **fit\_k1** (*bool*) – Enable optimization of k1 radial distortion coefficient.
- **fit\_k2** (*bool*) – Enable optimization of k2 radial distortion coefficient.
- **fit\_k3** (*bool*) – Enable optimization of k3 radial distortion coefficient.
- **fit\_k4** (*bool*) – Enable optimization of k3 radial distortion coefficient.
- **fit\_p1** (*bool*) – Enable optimization of p1 tangential distortion coefficient.
- **fit\_p2** (*bool*) – Enable optimization of p2 tangential distortion coefficient.
- **fit\_corrections** (*bool*) – Enable optimization of additional corrections.
- **adaptive\_fitting** (*bool*) – Enable adaptive fitting of distortion coefficients.
- **tiepoint\_covariance** (*bool*) – Estimate tie point covariance matrices.
- **progress** (*Callable[[float], None]*) – Progress callback.

**orthomosaic**

Default orthomosaic for the current frame.

Type *Orthomosaic*

**orthomosaics**

List of orthomosaics for the current frame.

Type list of *Orthomosaic*

**point\_cloud**

Generated tie point cloud.

Type *PointCloud*

**primary\_channel**

Primary channel index (-1 for default).

Type *int*

**publishData**(*service, source, export\_point\_colors=False, with\_camera\_track=False* [, *title*] [, *description*] [, *token*] [, *is\_draft=False, is\_private=False*] [, *password* ])

Publish generated data online.

**Parameters**

- **service** (*ServiceType*) – Service to upload on.
- **source** (*DataSource*) – Item to upload.
- **with\_camera\_track** (*bool*) – If model should be uploaded with camera track. Can be used only with *DataSource.ModelData*.
- **export\_point\_colors** (*bool*) – If Point Cloud should be uploaded with point colors.
- **title** (*string*) – Title of uploading model.
- **description** (*string*) – Description of uploading model.
- **token** (*string*) – Token used to upload data.
- **is\_draft** (*bool*) – If model should be uploaded as draft.
- **is\_private** (*bool*) – If model should have private access.
- **password** (*string*) – Password to access model if uploaded as private.

**raster\_transform**

Raster transform.

Type *RasterTransform*

**reduceOverlap**(*overlap=3, use\_selection=False* [, *progress* ])

Disable redundant cameras.

**Parameters**

- **overlap** (*int*) – Target number of cameras observing each point of the surface.
- **use\_selection** (*bool*) – Focus on model selection.
- **progress** (*Callable[[float], None]*) – Progress callback.

**refineMarkers**(*markers*] [, *progress* ])

Refine markers based on images content.

**Parameters**

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

**refineMesh**(*downscale=4, iterations=10, smoothness=0.5*[, *cameras*][, *progress*])  
Generate model for the chunk frame.

#### Parameters

- **downscale** (*int*) – Refinement quality.
- **iterations** (*int*) – Number of refinement iterations.
- **smoothness** (*float*) – Smoothing strength. Should be in range [0, 1].
- **cameras** (*list of int*) – List of cameras to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

#### region

Reconstruction volume selection.

Type *Region*

#### remove(*items*)

Remove items from the chunk.

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

**removeLighting**(*color\_mode=False, internal\_blur=1.5, mesh\_noise\_suppression=1, ambient\_occlusion\_path=""*, *ambient\_occlusion\_multiplier=1.5*[, *progress*])  
Generate model for the chunk frame.

#### Parameters

- **color\_mode** (*bool*) – Enable multi-color processing mode.
- **internal\_blur** (*float*) – Internal blur. Should be in range [0, 4].
- **mesh\_noise\_suppression** (*float*) – Mesh normals noise suppression strength. Should be in range [0, 4].
- **ambient\_occlusion\_path** (*string*) – Path to ambient occlusion texture atlas. Can be empty.
- **ambient\_occlusion\_multiplier** (*float*) – Ambient occlusion multiplier. Should be in range [0.25, 4].
- **progress** (*Callable[[float], None]*) – Progress callback.

**renderPreview**(*width = 2048, height = 2048*[, *transform* ], *point\_size=1*[, *progress*])  
Generate preview image for the chunk.

#### Parameters

- **width** (*int*) – Preview image width.
- **height** (*int*) – Preview image height.
- **transform** (*Matrix*) – 4x4 viewpoint transformation matrix.
- **point\_size** (*int*) – Point size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**Returns** Preview image.

**Return type** *Image*

**resetRegion()**

Reset reconstruction volume selector to default position.

**samplePoints**(*source\_data=ModelData, uniform\_sampling=True, points\_spacing=0.1*, [*asset*], [*progress*])

Sample point cloud from the model.

**Parameters**

- **source\_data** (*DataSource*) – Source data to extract points from.
- **uniform\_sampling** (*bool*) – Sampling method
- **points\_spacing** (*float*) – Desired point spacing (m).
- **asset** (*int*) – Model to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**scalebar\_accuracy**

Expected scale bar accuracy in meters.

**Type** *float*

**scalebar\_groups**

List of scale bar groups in the chunk.

**Type** list of *ScalebarGroup*

**scalebars**

List of scale bars in the chunk.

**Type** list of *Scalebar*

**selected**

Selects/deselects the chunk.

**Type** *bool*

**sensors**

List of sensors in the chunk.

**Type** list of *Sensor*

**shapes**

Shapes for the current frame.

**Type** *Shapes*

**smoothModel**(*strength=3, apply\_to\_selection=False, fix\_borders=True, preserve\_edges=False*, [*progress*])

Smooth mesh using Laplacian smoothing algorithm.

**Parameters**

- **strength** (*float*) – Smoothing strength.
- **apply\_to\_selection** (*bool*) – Apply to selected faces.
- **fix\_borders** (*bool*) – Fix borders.
- **preserve\_edges** (*bool*) – Preserve edges.
- **progress** (*Callable[[float], None]*) – Progress callback.

**sortCameras()**

Sorts cameras by their labels.

**sortMarkers()**

Sorts markers by their labels.

**sortScalebars()**

Sorts scalebars by their labels.

**thinPointCloud(*point\_limit=1000*)**

Remove excessive tracks from the point cloud.

**Parameters** **point\_limit** (*int*) – Maximum number of points for each photo.

**thumbnails**

Image thumbnails.

**Type** *Thumbnails*

**tiepoint\_accuracy**

Expected tie point accuracy in pixels.

**Type** float

**tiled\_model**

Default tiled model for the current frame.

**Type** *TiledModel*

**tiled\_models**

List of tiled models for the current frame.

**Type** list of *TiledModel*

**trackMarkers(*first\_frame=0, last\_frame=0* [, *progress* ])**

Track marker projections through the frame sequence.

**Parameters**

- **first\_frame** (*int*) – Starting frame index.
- **last\_frame** (*int*) – Ending frame index.
- **progress** (*Callable* [[*float*], *None*]) – Progress callback.

**transform**

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

**Type** *ChunkTransform*

**transformRaster(*data\_source=ElevationData* [, *asset* ], *subtract=False* [, *operand\_chunk* ] [, *operand\_frame* ] [, *operand\_asset* ], *width=0, height=0* [, *world\_transform* ], *resolution=0, resolution\_x=0, resolution\_y=0, nodata\_value=-32767, north\_up=True* [, *region* ] [, *projection* ] [, *progress* ])**

Transform DEM or orthomosaic.

**Parameters**

- **data\_source** (*DataSource*) – Selects between DEM and orthomosaic.
- **asset** (*int*) – Asset key to transform.
- **subtract** (*bool*) – Subtraction flag.
- **operand\_chunk** (*int*) – Operand chunk key.
- **operand\_frame** (*int*) – Operand frame key.
- **operand\_asset** (*int*) – Operand asset key.

- **width** (*int*) – Raster width.
- **height** (*int*) – Raster height.
- **world\_transform** (*Matrix*) – 2x3 raster-to-world transformation matrix.
- **resolution** (*float*) – Output resolution in meters.
- **resolution\_x** (*float*) – Pixel size in the X dimension in projected units.
- **resolution\_y** (*float*) – Pixel size in the Y dimension in projected units.
- **nodata\_value** (*float*) – No-data value (DEM export only).
- **north\_up** (*bool*) – Use north-up orientation for export.
- **region** (*BBox*) – Region to be processed.
- **projection** (*OrthoProjection*) – Output projection.
- **progress** (*Callable[[float], None]*) – Progress callback.

**triangulatePoints**(*max\_error=10, min\_image=2[, progress]*)

Rebuild point cloud for the chunk.

**Parameters**

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

**updateTransform**()

Update chunk transformation based on reference data.

**world\_crs**

Coordinate system used as world coordinate system.

**Type** *CoordinateSystem*

**class** Metashape.**ChunkTransform**

Transformation between chunk and world coordinates systems.

**copy**()

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *ChunkTransform*

**matrix**

Transformation matrix.

**Type** *Matrix*

**rotation**

Rotation component.

**Type** *Matrix*

**scale**

Scale component.

**Type** float

**translation**

Translation component.

Type *Vector*

**class** `Metashape.CirTransform`

CIR calibration matrix.

**calibrate()**

Calibrate CIR matrix based on orthomosaic histogram.

**coeffs**

Color matrix.

Type *Matrix*

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *CirTransform*

**reset()**

Reset CIR calibration matrix.

**class** `Metashape.ClassificationMethod`

Index values classification method in [EqualIntervalsClassification, JenksNaturalBreaksClassification]

**class** `Metashape.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 Metashape
>>> chunk = Metashape.app.document.chunk
>>> chunk.crs = Metashape.CoordinateSystem("EPSG::32641")
>>> chunk.importReference("gcp.txt", Metashape.ReferenceFormatCSV)
>>> chunk.updateTransform()
```

**addGeoid(*path*)**

Register geoid model.

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

**authority**

Authority identifier of the coordinate system.

**Type** *string*

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *CoordinateSystem*

**datumTransform(*source*, *target*)**

Coordinate transformation from source to target coordinate system datum.

**Parameters**

- **source** (*CoordinateSystem*) – Source coordinate system.
- **target** (*CoordinateSystem*) – Target coordinate system.

**Returns** 4x4 transformation matrix.



**Return type** *Matrix*

**geoccs**

Base geocentric coordinate system.

**Type** *CoordinateSystem*

**geogcs**

Base geographic coordinate system.

**Type** *CoordinateSystem*

**geoid\_height**

Fixed geoid height to be used instead of interpolated values.

**Type** float

**init**(*crs*)

Initialize projection based on specified WKT definition or authority identifier.

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

**listBuiltinCRS**()

Returns a list of builtin coordinate systems.

**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*

**name**

Name of the coordinate system.

**Type** string

**proj4**

Coordinate system definition in PROJ.4 format.

**Type** string

**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*

**towgs84**

TOWGS84 transformation parameters (dx, dy, dz, rx, ry, rz, scale).

**Type** list of float

**transform**(*point, source, target*)

Transform point coordinates between coordinate systems.

**Parameters**

- **point** (2 or 3 component *Vector*) – Point coordinates.
- **source** (*CoordinateSystem*) – Source coordinate system.
- **target** (*CoordinateSystem*) – Target coordinate system.

**Returns** Transformed point coordinates.

**Return type** *Vector*

**transformationMatrix**(*point*, *source*, *target*)

Local approximation of coordinate transformation from source to target coordinate system at the given point.

**Parameters**

- **point** (3 component *Vector*) – Point coordinates.
- **source** (*CoordinateSystem*) – Source coordinate system.
- **target** (*CoordinateSystem*) – Target coordinate system.

**Returns** 4x4 transformation matrix.

**Return type** *Matrix*

**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**

Coordinate system definition in WKT format.

**Type** string

**wkt2**

Coordinate system definition in WKT format, version 2.

**Type** string

**class** `Metashape.DataSource`

Data source in [PointCloudData, DenseCloudData, DepthMapsData, ModelData, TiledModelData, ElevationData, OrthomosaicData, ImagesData]

**class** `Metashape.DataType`

Data type in [DataTypeUndefined, DataType8i, DataType8u, DataType16i, DataType16u, DataType16f, DataType32i, DataType32u, DataType32f, DataType64i, DataType64u, DataType64f]

**class** `Metashape.DenseCloud`

Dense point cloud data.

**assignClass**(*target=0*[, *source* ][, *progress* ])

Assign class to points.

**Parameters**

- **target** (*PointClass*) – Target class.
- **source** (*PointClass* or list of *PointClass*) – Classes of points to be replaced.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**assignClassToSelection**(*target=0*[, *source* ][, *progress* ])

Assign class to selected points.

**Parameters**

- **target** (*PointClass*) – Target class.

- **source** (*PointClass* or list of *PointClass*) – Classes of points to be replaced.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**bands**

List of color bands.

**Type** list of string

**classifyGroundPoints**(*max\_angle=15.0*, *max\_distance=1.0*, *cell\_size=50.0*, *erosion\_radius=0.0*[[, *source* ]], *progress* ])

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).
- **erosion\_radius** (*float*) – Erosion radius (meters).
- **source** (*PointClass*) – Class of points to be re-classified.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**classifyPoints**([[*source* ]], [*target* ], *confidence=0.0*[[, *progress* ]])

Multiclass classification of points.

**Parameters**

- **source** (*PointClass*) – Class of points to be re-classified.
- **target** (list of *PointClass*) – Target point classes for classification.
- **confidence** (*float*) – Required confidence level from 0.0 to 1.0.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**clear()**

Clears dense cloud data.

**compactPoints**([[*progress* ]])

Permanently removes deleted points from dense cloud.

**Parameters** **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**copy()**

Create a copy of the dense cloud.

**Returns** Copy of the dense cloud.

**Return type** *DenseCloud*

**cropSelectedPoints**([[*point\_classes* ]], [*progress* ])

Crop selected points.

**Parameters**

- **point\_classes** (*PointClass* or list of *PointClass*) – Classes of points to be removed.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**crs**

Reference coordinate system.

**Type** *CoordinateSystem* or *None*

**data\_type**

Data type used to store color values.

Type *DataType*

**key**

Dense cloud identifier.

Type *int*

**label**

Dense cloud label.

Type *string*

**meta**

Dense cloud meta data.

Type *MetaData*

**modified**

Modified flag.

Type *bool*

**pickPoint**(*origin, target, endpoints=1*)

Returns ray intersection with the point cloud (point on the ray nearest to some point).

**Parameters**

- **origin** (*Vector*) – Ray origin.
- **target** (*Vector*) – Point on the ray.
- **endpoints** (*int*) – Number of endpoints to check for (0 - line, 1 - ray, 2 - segment).

**Returns** Coordinates of the intersection point.

**Return type** *Vector*

**point\_count**

Number of points in dense cloud.

Type *int*

**removePoints**(*point\_classes*[, *progress* ])

Remove points.

**Parameters**

- **point\_classes** (*PointClass* or list of *PointClass*) – Classes of points to be removed.
- **progress** (*Callable*[*float*], *None*) – Progress callback.

**removeSelectedPoints**([*point\_classes* ][, *progress* ])

Remove selected points.

**Parameters**

- **point\_classes** (*PointClass* or list of *PointClass*) – Classes of points to be removed.
- **progress** (*Callable*[*float*], *None*) – Progress callback.

**renderDepth**(*transform, calibration, point\_size=1, resolution=1, cull\_points=False, add\_alpha=True*)

Render dense cloud depth image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point\_size** (*int*) – Point size.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_points** (*bool*) – Enable normal based culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** *Image*

**renderImage**(*transform, calibration, point\_size=1, resolution=1, cull\_points=False, add\_alpha=True, raster\_transform=RasterTransformNone*)

Render dense cloud image for specified viewpoint.

#### Parameters

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point\_size** (*int*) – Point size.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_points** (*bool*) – Enable normal based culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.

**Returns** Rendered image.

**Return type** *Image*

**renderMask**(*transform, calibration, point\_size=1, resolution=1, cull\_points=False*)

Render dense cloud mask image for specified viewpoint.

#### Parameters

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point\_size** (*int*) – Point size.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_points** (*bool*) – Enable normal based culling.

**Returns** Rendered image.

**Return type** *Image*

**renderNormalMap**(*transform, calibration, point\_size=1, resolution=1, cull\_points=False, add\_alpha=True*)

Render image with dense cloud normals for specified viewpoint.

#### Parameters

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point\_size** (*int*) – Point size.

- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_points** (*bool*) – Enable normal based culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** *Image*

**renderPreview**(*width = 2048, height = 2048*[, *transform* ], *point\_size=1*[, *progress* ])  
Generate dense cloud preview image.

**Parameters**

- **width** (*int*) – Preview image width.
- **height** (*int*) – Preview image height.
- **transform** (*Matrix*) – 4x4 viewpoint transformation matrix.
- **point\_size** (*int*) – Point size.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**Returns** Preview image.

**Return type** *Image*

**resetFilters**()

Reset filters.

**restorePoints**([*point\_classes* ][, *progress* ])

Restore deleted points.

**Parameters**

- **point\_classes** (*PointClass* or list of *PointClass*) – Classes of points to be restored.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**selectMaskedPoints**(*cameras, softness=4*[, *progress* ])

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.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**selectPointsByColor**(*color, tolerance=10, channels='RGB'*[, *progress* ])

Select dense points based on point colors.

**Parameters**

- **color** (*list of int*) – Color to select.
- **tolerance** (*int*) – Color tolerance.
- **channels** (*string*) – Combination of color channels to compare in ['R', 'G', 'B', 'H', 'S', 'V'].
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**selectPointsByShapes**([*shapes* ][, *progress* ])

Select dense points based on shapes.

**Parameters**

- **shapes** (list of *Shape*) – A list of shapes to use for selection (selected shapes if not specified).
- **progress** (*Callable[[float], None]*) – Progress callback.

**setClassesFilter**(*point\_classes*)

Set filter by point classes.

**Parameters** **point\_classes** (*PointClass* or list of *PointClass*) – List of point classes.

**setConfidenceFilter**(*min\_confidence*, *max\_confidence*)

Set filter by confidence.

**Parameters**

- **min\_confidence** (*int*) – Minimum confidence value.
- **max\_confidence** (*int*) – Maximum confidence value.

**setSelectionFilter**()

Set filter by selection.

**transform**

4x4 dense cloud transformation matrix.

**Type** *Matrix*

**updateStatistics**([*progress*])

Updates dense cloud statistics.

**Parameters** **progress** (*Callable[[float], None]*) – Progress callback.

**class** `Metashape.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*

**getCalibration**(*level=0*)

Returns calibration data.

**Parameters** **level** (*int*) – Level index.

**Returns** Calibration data.

**Return type** *Calibration*

**image**([*level*])

Returns image data.

**Parameters** **level** (*int*) – Level index.

**Returns** Image data.

**Return type** *Image*

**setCalibration**(*calibration*, *level=0*)

**Parameters**

- **calibration** (*Calibration*) – Calibration data.
- **level** (*int*) – Level index.

**setImage**(*image*, *level=0*)

**Parameters**

- **image** (*Image*) – Image object with depth map data.
- **level** (*int*) – Level index.

**class Metashape.DepthMaps**

A set of depth maps generated for a chunk frame.

**clear**()

Clears depth maps data.

**copy**()

Create a copy of the depth maps.

**Returns** Copy of the depth maps.

**Return type** *DepthMaps*

**items**()

List of items.

**key**

Depth maps identifier.

**Type** *int*

**keys**()

List of item keys.

**label**

Depth maps label.

**Type** *string*

**meta**

Depth maps meta data.

**Type** *MetaData*

**modified**

Modified flag.

**Type** *bool*

**values**()

List of item values.

**class Metashape.Document**

Metashape 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 Metashape window can be accessed using `Metashape.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 Metashape
>>> doc = Metashape.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*=0, *fit\_scale*=True, *downscale*=1, *generic\_preselection*=False, *filter\_mask*=False, *mask\_tiepoints*=False, *keypoint\_limit*=40000[, *markers*][, *progress*])

Align specified set of chunks.

#### **Parameters**

- **chunks** (*list of int*) – List of chunks to be aligned.
- **reference** (*int*) – Chunk to be used as a reference.
- **method** (*int*) – Alignment method (0 - point based, 1 - marker based, 2 - camera based).
- **fit\_scale** (*bool*) – Fit chunk scale during alignment.
- **downscale** (*int*) – Alignment accuracy.
- **generic\_preselection** (*bool*) – Enables image pair preselection.
- **filter\_mask** (*bool*) – Filter points by mask.
- **mask\_tiepoints** (*bool*) – Apply mask filter to tie points.
- **keypoint\_limit** (*int*) – Maximum number of points for each photo.
- **markers** (*list of int*) – List of markers to be used for marker based alignment.
- **progress** (*Callable[[float], None]*) – Progress callback.

**append**(*document*[, *chunks*][, *progress*])

Append the specified Document object to the current document.

#### **Parameters**

- **document** (*Document*) – Document object to be appended.
- **chunks** (*list of Chunk*) – List of chunks to append.
- **progress** (*Callable[[float], None]*) – Progress callback.

### **chunk**

Active chunk.

**Type** *Chunk*

### **chunks**

List of chunks in the document.

**Type** Chunks

### **clear()**

Clear the contents of the Document object.

**copy()**

Return a copy of the document.

**Returns** A copy of the document.

**Return type** *Document*

**findChunk(key)**

Find chunk by its key.

**Returns** Found chunk.

**Return type** *Chunk*

**mergeChunks**(*merge\_markers=False, merge\_tiepoints=False, merge\_depth\_maps=False, merge\_dense\_clouds=True, merge\_models=False, merge\_elevations=False, merge\_orthomosaics=False[, chunks][, progress]*)

Merge specified set of chunks.

**Parameters**

- **merge\_markers** (*bool*) – Merge markers.
- **merge\_tiepoints** (*bool*) – Merge tie points.
- **merge\_depth\_maps** (*bool*) – Merge depth maps.
- **merge\_dense\_clouds** (*bool*) – Merge dense clouds.
- **merge\_models** (*bool*) – Merge models.
- **merge\_elevations** (*bool*) – Merge DEMs.
- **merge\_orthomosaics** (*bool*) – Merge orthomosaics.
- **chunks** (*list of int*) – List of chunks to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**meta**

Document meta data.

**Type** *MetaData*

**modified**

Modified flag.

**Type** *bool*

**open**(*path, read\_only=False, ignore\_lock=False*)

Load document from the specified file.

**Parameters**

- **path** (*string*) – Path to the file.
- **read\_only** (*bool*) – Open document in read-only mode.
- **ignore\_lock** (*bool*) – Ignore lock state for project modifications.

**path**

Path to the document file.

**Type** *string*

**read\_only**

Read only status.

**Type** bool

**remove**(*items*)

Remove a set of items from the document.

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

**save**([*path*][, *chunks*][, *version*])

Save document to the specified file.

**Parameters**

- **path** (*string*) – Optional path to the file.
- **chunks** (list of *Chunk*) – List of chunks to be saved.
- **version** (*string*) – Project version to save.

**class** Metashape.Elevation

Digital elevation model.

**altitude**(*point*)

Return elevation value at the specified point.

**Parameters** *point* (*Vector*) – Point coordinates in the elevation coordinate system.

**Returns** Elevation value.

**Return type** float

**bottom**

Y coordinate of the bottom side.

**Type** float

**clear**()

Clears elevation model data.

**copy**()

Create a copy of the elevation model.

**Returns** Copy of the elevation model.

**Return type** *Elevation*

**crs**

Coordinate system of elevation model.

**Type** *CoordinateSystem*

**height**

Elevation model height.

**Type** int

**key**

Elevation model identifier.

**Type** int

**label**

Elevation model label.

**Type** string

**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

**modified**

Modified flag.

**Type** bool

**palette**

Color palette.

**Type** dict

**projection**

Projection of elevation model.

**Type** *OrthoProjection*

**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** `Metashape.EulerAngles`

Euler angles in [EulerAnglesYPR, EulerAnglesOPK, EulerAnglesPOK, EulerAnglesANK]

**class** `Metashape.FaceCount`

Face count in [LowFaceCount, MediumFaceCount, HighFaceCount, CustomFaceCount]

**class** `Metashape.FilterMode`

Depth filtering mode in [NoFiltering, MildFiltering, ModerateFiltering, AggressiveFiltering]

**class** `Metashape.FrameStep`

Frame step size for video import in [CustomFrameStep, SmallFrameStep, MediumFrameStep, LargeFrameStep]

**class** `Metashape.Geometry`

Geometry data.

**GeometryCollection**(*geometries*)

Create a GeometryCollection geometry.

**Parameters** **geometries** (list of *Geometry*) – Child geometries.

**Returns** A GeometryCollection geometry.

**Return type** *Geometry*

**LineString**(*coordinates*)

Create a LineString geometry.

**Parameters** **coordinates** (list of *Vector*) – List of vertex coordinates.

**Returns** A LineString geometry.

**Return type** *Geometry*

**MultiLineString**(*geometries*)

Create a MultiLineString geometry.

**Parameters** **geometries** (list of *Geometry*) – Child line strings.

**Returns** A point geometry.

**Return type** *Geometry*

**MultiPoint**(*geometries*)

Create a MultiPoint geometry.

**Parameters** **geometries** (list of *Geometry*) – Child points.

**Returns** A point geometry.

**Return type** *Geometry*

**MultiPolygon**(*geometries*)

Create a MultiPolygon geometry.

**Parameters** **geometries** (list of *Geometry*) – Child polygons.

**Returns** A point geometry.

**Return type** *Geometry*

**Point**(*vector*)

Create a Point geometry.

**Parameters** **vector** (*Vector* or list of floats) – Point coordinates.

**Returns** A point geometry.

**Return type** *Geometry*

**Polygon**(*exterior\_ring*[, *interior\_rings* ])

Create a Polygon geometry.

**Parameters**

- **exterior\_ring** (list of *Vector*) – Point coordinates.
- **interior\_rings** (list of *Vector*) – Point coordinates.

**Returns** A Polygon geometry.

**Return type** *Geometry*

**class Type**

Geometry type in [PointType, LineStringType, PolygonType, MultiPointType, MultiLineStringType, MultiPolygonType, GeometryCollectionType]

**coordinates**

List of vertex coordinates.

**Type** *Vector*

**geometries**

List of child geometries.

**Type** *Geometry*

**is\_3d**

Is 3D flag.

**Type** bool

**type**

Geometry type.

**Type** *Geometry.Type*

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

n-channel image

**Parameters**

- **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', 'F16', 'F32', 'F64']

**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', 'F16', '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', 'F16', 'F32', 'F64']

**Returns** Created image.

**Return type** *Image*

**gaussianBlur**(*radius*)

Smooth image with a gaussian filter.

**Parameters** **radius** (*float*) – smoothing radius.

**Returns** Smoothed image.

**Return type** *Image*

**height**

Image height.

**Type** int

**open**(*path*, *layer*=0, *datatype*='U8'[, *channels*][[, *x*][[, *y*][[, *w*][[, *h*]])

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', 'F16', 'F32', 'F64']
- **channels** (*string*) – color channels to be loaded, e.g. 'RGB', 'RGBA', etc.
- **x** (*int*) – x offset of image region.
- **y** (*int*) – y offset of image region.
- **w** (*int*) – width of image region.
- **h** (*int*) – height of image region.

**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*[, *compression* ])

Save image to the file.

**Parameters**

- **path** (*string*) – path to the image file
- **compression** (*ImageCompression*) – compression options

**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** (*bool*) – moves principal point to the image center
- **square\_pixels** (*bool*) – create image with square pixels

**Returns** undistorted image

**Return type** *Image*

**uniformNoise**(*amplitude*)

Add uniform noise with specified amplitude.

**Parameters** **amplitude** (*float*) – noise amplitude.

**Returns** Image with added noise.

**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 Metashape.ImageCompression**

Image compression parameters

**class TiffCompression**

Tiff compression in [TiffCompressionNone, TiffCompressionLZW, TiffCompressionJPEG, TiffCompressionPackbits, TiffCompressionDeflate]

**copy()**

Return a copy of the object.

**Returns** A copy of the object.**Return type** *Viewpoint***jpeg\_quality**

JPEG quality.

**Type** int**tiff\_big**

Enable BigTIFF compression for TIFF files.

**Type** bool**tiff\_compression**

Tiff compression.

**Type** int**tiff\_overviews**

Enable image pyramid deneneration for TIFF files.

**Type** bool**tiff\_tiled**

Export tiled TIFF.

**Type** bool**class Metashape.ImageFormat**

Image format in [ImageFormatNone, ImageFormatJPEG, ImageFormatTIFF, ImageFormatPNG, ImageFormatBMP, ImageFormatEXR, ImageFormatPNM, ImageFormatSGI, ImageFormatCR2, ImageFormatSEQ, ImageFormatBIL, ImageFormatXYZ, ImageFormatARA, ImageFormatTGA, ImageFormatDDS, ImageFormatJP2, ImageFormatWebP, ImageFormatJXL]

**class Metashape.ImageLayout**

Image layout in [UndefinedLayout, FlatLayout, MultiframeLayout, MultiplaneLayout]

**class Metashape.Interpolation**

Interpolation mode in [DisabledInterpolation, EnabledInterpolation, Extrapolated]

**class Metashape.License**

License information.

**activate(*license\_key*)**

Activate software online using a license key.

**Parameters** **key** (*string*) – Activation key.**activateOffline(*license\_key*)**

Create a request for offline activation.

**Parameters** **key** (*string*) – Activation key.**Returns** Activation request.

**Return type** string

**deactivate()**

Deactivate software online.

**deactivateOffline()**

Create a request for offline deactivation.

**Returns** Deactivation request.

**Return type** string

**valid**

Metashape activation status.

**Type** bool

**class** Metashape.**MappingMode**

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

**class** Metashape.**Marker**

Marker instance

**class** **Projection**

Marker data().

**coord**

Point coordinates in pixels.

**Type** *Vector*

**pinned**

Pinned flag.

**Type** bool

**valid**

Valid flag.

**Type** bool

**class** **Projections**

Collection of projections specified for the marker

**items()**

List of items.

**keys()**

List of item keys.

**values()**

List of item values.

**class** **Reference**

Marker reference data.

**accuracy**

Marker location accuracy.

**Type** *Vector*

**enabled**

Enabled flag.

**Type** bool

**location**

Marker coordinates.

**Type** *Vector*

**class Type**

Marker type in [Regular, Vertex, Fiducial]

**chunk**

Chunk the marker belongs to.

**Type** *Chunk*

**enabled**

Enables/disables the marker.

**Type** bool

**frames**

Marker frames.

**Type** list of *Marker*

**group**

Marker group.

**Type** *MarkerGroup*

**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*

**position\_covariance**

Marker position covariance.

**Type** *Matrix*

**projections**

List of marker projections.

**Type** *MarkerProjections*

**reference**

Marker reference data.

**Type** *MarkerReference*

**selected**

Selects/deselects the marker.

**Type** bool

**sensor**

Fiducial mark sensor.

**Type** *Sensor*

**type**

Marker type.

**Type** *Marker.Type*

**class** Metashape.**MarkerGroup**

MarkerGroup objects define groups of multiple markers. The grouping is established by assignment of a MarkerGroup instance to the Marker.group attribute of participating markers.

**label**

Marker group label.

**Type** string

**selected**

Current selection state.

**Type** bool

**class** Metashape.**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*

**invert()**

Create inverted copy of the mask.

**Returns** Inverted copy of the mask.

**Return type** *Mask*

**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.

**setImage**(*image*)

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

**class** Metashape.**MaskOperation**

Mask operation in [MaskOperationReplacement, MaskOperationUnion, MaskOperationIntersection, MaskOperationDifference]

**class** Metashape.**MaskingMode**

Masking mode in [MaskingModeAlpha, MaskingModeFile, MaskingModeBackground, MaskingModeModel]

**class Metashape.Masks**

A set of masks for a chunk frame.

**items()**

List of items.

**keys()**

List of item keys.

**meta**

Thumbnails meta data.

**Type** *MetaData*

**modified**

Modified flag.

**Type** bool

**values()**

List of item values.

**class Metashape.Matrix**

m-by-n matrix

```
>>> import Metashape
>>> m1 = Metashape.Matrix.Diag( (1,2,3,4) )
>>> m3 = Metashape.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:
...     Metashape.app.messageBox("Diagonal matrix dimensions: " + str(m3.size))
```

**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*

**Rotation**(*matrix*)

Create a rotation matrix.

**Parameters** **matrix** (*Matrix*) – The 3x3 rotation matrix.

**Returns** 4x4 matrix representing rotation.

**Return type** *Matrix*

**Scale**(*scale*)

Create a scale matrix.

**Parameters** **scale** (*Vector*) – The scale vector.

**Returns** A matrix representing scale.

**Return type** *Matrix*

**Translation**(*vector*)

Create a translation matrix.

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

**Returns** A matrix representing translation.

**Return type** *Matrix*

**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

**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*

**rotation**()

Returns rotation component of the 4x4 matrix.

**Returns** rotation component

**Return type** *Matrix*

**row**(*index*)

Returns row of the matrix.

**Returns** matrix row.

**Return type** *Vector*

**scale**()

Returns scale component of the 4x4 matrix.

**Returns** scale component

**Return type** float

**size**

Matrix dimensions.

**Type** tuple

**svd()**

Returns singular value decomposition of the matrix.

**Returns** u, s, v tuple where  $a = u * \text{diag}(s) * v$

**Return type** *Matrix Vector Matrix* tuple

**t()**

Return a new, transposed matrix.

**Returns** a transposed matrix

**Return type** *Matrix*

**translation()**

Returns translation component of the 4x4 matrix.

**Returns** translation component

**Return type** *Vector*

**zero()**

Set all matrix elements to zero.

**class** Metashape.**MetaData**(*object*)

Collection of object properties

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *MetaData*

**items()**

List of items.

**keys()**

List of item keys.

**values()**

List of item values.

**class** Metashape.**Model**

Triangular mesh model instance

**class** **Face**

Triangular face of the model

**hidden**

Face visibility flag.

**Type** bool

**selected**

Face selection flag.

**Type** bool

**tex\_index**

Texture page index.

**Type** int

**tex\_vertices**  
Texture vertex indices.  
**Type** tuple of 3 int

**vertices**  
Vertex indices.  
**Type** tuple of 3 int

**class Faces**  
Collection of model faces

**resize**(*count*)  
Resize faces list.  
**Parameters** **count** (*int*) – new face count

**class Statistics**  
Mesh statistics

**components**  
Number of connected components.  
**Type** int

**degenerate\_faces**  
Number of degenerate faces.  
**Type** int

**duplicate\_faces**  
Number of duplicate faces.  
**Type** int

**faces**  
Total number of faces.  
**Type** int

**flipped\_normals**  
Number of edges with flipped normals.  
**Type** int

**free\_vertices**  
Number of free vertices.  
**Type** int

**multiple\_edges**  
Number of edges connecting more than 2 faces.  
**Type** int

**open\_edges**  
Number of open edges.  
**Type** int

**out\_of\_range\_indices**  
Number of out of range indices.  
**Type** int

**similar\_vertices**  
Number of similar vertices.  
**Type** int

**vertices**  
Total number of vertices.



**Type** int

**zero\_faces**

Number of zero faces.

**Type** int

**class TexVertex**

Texture vertex of the model

**coord**

Vertex coordinates.

**Type** tuple of 2 float

**class TexVertices**

Collection of model texture vertices

**resize**(*count*)

Resize vertex list.

**Parameters** **count** (*int*) – new vertex count

**class Texture**

Model texture.

**image**(*page=0*)

Return texture image.

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

**Returns** Texture image.

**Return type** *Image*

**label**

Animation label.

**Type** string

**meta**

Camera track meta data.

**Type** *MetaData*

**model**

Model the texture belongs to.

**Type** *Model*

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

Initialize texture from image data.

**Parameters**

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

**type**

Texture type.

**Type** *Model.TextureType*

**class TextureType**

Texture type in [DiffuseMap, NormalMap, OcclusionMap]

**class Vertex**

Vertex of the model

**color**

Vertex color.

**Type** tuple of 3 int

**confidence**

Vertex confidence.

**Type** float

**coord**

Vertex coordinates.

**Type** *Vector*

**class Vertices**

Collection of model vertices

**resize**(*count*)

Resize vertex list.

**Parameters** **count** (*int*) – new vertex count

**addTexture**(*type=Model.DiffuseMap*)

Add new texture to the model.

**Parameters** **type** (*Model.TextureType*) – Texture type.

**Returns** Created texture.

**Return type** *Model.Texture*

**area**()

Return area of the model surface.

**Returns** Model area.

**Return type** float

**bands**

List of color bands.

**Type** list of string

**clear**()

Clears model data.

**closeHoles**(*level=30, apply\_to\_selection=False*)

Fill holes in the model surface.

**Parameters**

- **level** (*int*) – Hole size threshold in percents.
- **apply\_to\_selection** (*bool*) – Close holes within selection

**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.

**data\_type**

Data type used to store color values.

**Type** *DataType*

**faces**

Collection of mesh faces.

**Type** MeshFaces

**fixTopology()**

Remove polygons causing topological problems.

**getActiveTexture** (*type=Model.DiffuseMap*)

Return active texture.

**Parameters** **type** (*Model.TextureType*) – Texture type.

**Returns** Texture image.

**Return type** *Image*

**key**

Model identifier.

**Type** int

**label**

Model label.

**Type** string

**loadTexture** (*path*)

Load texture from the specified file.

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

**meta**

Model meta data.

**Type** *MetaData*

**modified**

Modified flag.

**Type** bool

**pickPoint** (*origin, target, endpoints=1*)

Return ray intersection with mesh.

**Parameters**

- **origin** (*Vector*) – Ray origin.
- **target** (*Vector*) – Point on the ray.
- **endpoints** (*int*) – Number of endpoints to check for (0 - line, 1 - ray, 2 - segment).

**Returns** Coordinates of the intersection point.

**Return type** *Vector*

**remove** (*items*)

Remove textures from the model.

**Parameters** **items** (list of *Model.Texture*) – A list of textures to be removed.

**removeComponents** (*size*)

Remove small connected components.

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

**removeSelection()**

Remove selected faces and free vertices from the mesh.

**renderDepth**(*transform*, *calibration*, *cull\_faces=True*, *add\_alpha=True*)  
Render model depth image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** *Image*

**renderImage**(*transform*, *calibration*, *cull\_faces=True*, *add\_alpha=True*,  
*raster\_transform=RasterTransformNone*)  
Render model image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.

**Returns** Rendered image.

**Return type** *Image*

**renderMask**(*transform*, *calibration*, *cull\_faces=True*)  
Render model mask image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **cull\_faces** (*bool*) – Enable back-face culling.

**Returns** Rendered image.

**Return type** *Image*

**renderNormalMap**(*transform*, *calibration*, *cull\_faces=True*, *add\_alpha=True*)  
Render image with model normals for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** *Image*

**renderPreview**(*width = 2048, height = 2048*[, *transform* ][, *progress* ])

Generate model preview image.

**Parameters**

- **width** (*int*) – Preview image width.
- **height** (*int*) – Preview image height.
- **transform** (*Matrix*) – 4x4 viewpoint transformation matrix.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**Returns** Preview image.

**Return type** *Image*

**saveTexture**(*path*)

Save texture to the specified file.

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

**setActiveTexture**(*texture, type=Model.DiffuseMap*)

Set active texture.

**Parameters**

- **texture** (*Model.Texture*) – Texture to set.
- **type** (*Model.TextureType*) – Texture type.

**statistics**([*progress* ])

Return mesh statistics.

**Parameters** **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**Returns** Mesh statistics.

**Return type** *Model.Statistics*

**tex\_vertices**

Collection of mesh texture vertices.

**Type** *MeshTexVertices*

**textures**

List of model textures.

**Type** list of *Model.Texture*

**transform**(*transform*)

Transform vertex coordinates.

**Parameters** **transform** (*Matrix*) – 4x4 transformation matrix.

**vertices**

Collection of mesh vertices.

**Type** *MeshVertices*

**volume**()

Return volume of the closed model surface.

**Returns** Model volume.

**Return type** *float*

**class Metashape.ModelFormat**

Model format in [ModelFormatNone, ModelFormatOBJ, ModelFormat3DS, ModelFormatVRML, ModelFormatPLY, ModelFormatCOLLADA, ModelFormatU3D, ModelFormatPDF, ModelFormatDXF, ModelFormatFBX, ModelFormatKMZ, ModelFormatCTM, ModelFormatSTL, ModelFormatDXF\_3DF, ModelFormatTLS, ModelFormatABC, ModelFormatOSGB, ModelFormatOSGT, ModelFormatGLTF, ModelFormatX3D, ModelFormatLandXML]

**class Metashape.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 Metashape
>>> client = Metashape.NetworkClient()
>>> client.connect('127.0.0.1')
>>> client.batchList()
```

**abortBatch**(*batch\_id*)

Abort batch.

**Parameters** **batch\_id** (*int*) – Batch id.

**abortNode**(*node\_id*)

Abort node.

**Parameters** **node\_id** (*int*) – Node id.

**batchList**(*revision=0*)

Get list of batches.

**Parameters** **revision** (*int*) – First revision to get.

**Returns** List of batches.

**Return type** dict

**batchStatus**(*batch\_id, revision=0*)

Get batch status.

**Parameters**

- **batch\_id** (*int*) – Batch id.
- **revision** (*int*) – First revision to get.

**Returns** Batch status.

**Return type** dict

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

Connect to the server.

**Parameters**

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

**createBatch**(*path, tasks[, meta]*)

Create new batch.

**Parameters**

- **path** (*string*) – Project path relative to root folder.
- **tasks** (list of *NetworkTask*) – List of processing tasks to execute.

- **meta** (*MetaData*) – Batch metadata.

**Returns** Batch id.

**Return type** int

**disconnect()**

Disconnect from the server.

**dumpBatches**(*[batch\_ids]*)

Dump current state of batches.

**Parameters** **batch\_ids** (*list of int*) – List of batch ids to dump.

**Returns** Batches data.

**Return type** string

**findBatch**(*path*)

Get batch id based on project path.

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

**Returns** Batch id.

**Return type** int

**loadBatches**(*data*)

Load batches from dump.

**Parameters** **data** (*string*) – Batches data.

**nodeList**(*revision=0*)

Get list of nodes.

**Parameters** **revision** (*int*) – First revision to get.

**Returns** List of nodes.

**Return type** dict

**nodeStatus**(*node\_id, revision=0*)

Get node status.

**Parameters**

- **node\_id** (*int*) – Node id.
- **revision** (*int*) – First revision to get.

**Returns** Node status.

**Return type** dict

**quitNode**(*node\_id*)

Quit node.

**Parameters** **node\_id** (*int*) – Node id.

**serverInfo**()

Get server information.

**Returns** Server information.

**Return type** dict

**serverStatus**(*revision=0*)

Get server status.

**Parameters** **revision** (*int*) – First revision to get.

**Returns** Server status.

**Return type** dict

**setBatchNodeLimit**(*batch\_id, node\_limit*)

Set node limit of the batch.

**Parameters**

- **batch\_id** (*int*) – Batch id.
- **node\_limit** (*int*) – Node limit of the batch (0 - unlimited).

**setBatchPaused**(*batch\_id, paused=True*)

Set batch paused state.

**Parameters**

- **batch\_id** (*int*) – Batch id.
- **paused** (*bool*) – Paused state.

**setBatchPriority**(*batch\_id, priority*)

Set batch priority.

**Parameters**

- **batch\_id** (*int*) – Batch id.
- **priority** (*int*) – Batch priority (2 - Highest, 1 - High, 0 - Normal, -1 - Low, -2 - Lowest).

**setMasterServer**(*[host]*)

Set or reset master server.

**Parameters** **host** (*string*) – Master server hostname.

**setNodeCPUEnable**(*node\_id, cpu\_enable*)

Set node CPU enable flag.

**Parameters**

- **node\_id** (*int*) – Node id.
- **cpu\_enable** (*bool*) – CPU enable flag.

**setNodeCapability**(*node\_id, capability*)

Set node capability.

**Parameters**

- **node\_id** (*int*) – Node id.
- **capability** (*int*) – Node capability (1 - CPU, 2 - GPU, 3 - Any).

**setNodeGPUMask**(*node\_id, gpu\_mask*)

Set node GPU mask.

**Parameters**

- **node\_id** (*int*) – Node id.
- **gpu\_mask** (*int*) – GPU device mask.

**setNodePaused**(*node\_id, paused=True*)

Set node paused state.

**Parameters**



- **node\_id** (*int*) – Node id.
- **paused** (*bool*) – Paused state.

**setNodePriority**(*node\_id, priority*)

Set node priority.

**Parameters**

- **node\_id** (*int*) – Node id.
- **priority** (*int*) – Node priority (2 - Highest, 1 - High, 0 - Normal, -1 - Low, -2 - Lowest).

**class Metashape.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 Metashape
>>> task = Metashape.NetworkTask()
>>> task.name = 'MatchPhotos'
>>> task.params['keypoint_limit'] = 40000
>>> client = Metashape.NetworkClient()
>>> client.connect('127.0.0.1')
>>> batch_id = client.createBatch('processing/project.psx', [task])
>>> client.setBatchPaused(batch_id, false)
```

**chunks**

List of chunks.

**Type** list

**encode()**

Create a dictionary with task parameters.

**frames**

List of frames.

**Type** list

**name**

Task name.

**Type** string

**params**

Task parameters.

**Type** dict

**supports\_gpu**

GPU support flag.

**Type** bool

**class Metashape.OrthoProjection**

Orthographic projection.

**class Type**

Projection type in [Planar, Cylindrical]

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *OrthoProjection*

**crs**

Base coordinate system.

**Type** *CoordinateSystem*

**matrix**

Ortho transformation matrix.

**Type** *Matrix*

**radius**

Cylindrical projection radius.

**Type** float

**transform**(*point, source, target*)

Transform point coordinates between coordinate systems.

**Parameters**

- **point** (2 or 3 component *Vector*) – Point coordinates.
- **source** (*OrthoProjection*) – Source coordinate system.
- **target** (*OrthoProjection*) – Target coordinate system.

**Returns** Transformed point coordinates.

**Return type** *Vector*

**type**

Projection type.

**Type** *OrthoProjection.Type*

**class** `Metashape.Orthomosaic`

Orthomosaic data.

The following sample assigns to the first shape in the chunk the image from the first camera for the orthomosaic patch and updates the mosaic:

```
>>> import Metashape
>>> chunk = Metashape.app.document.chunk
>>> ortho = chunk.orthomosaic
>>> camera = chunk.cameras[0]
>>> shape = chunk.shapes[0]
>>> patch = Metashape.Orthomosaic.Patch()
>>> patch.image_keys = [camera.key]
>>> ortho.patches[shape] = patch
>>> ortho.update()
```

**class** `Patch`

Orthomosaic patch.

**copy**()

Returns a copy of the patch.

**Returns** Copy of the patch.

**Return type** *Orthomosaic.Patch*

**excluded**

Excluded flag.

**Type** bool

**image\_keys**

Image keys.

**Type** list of int**class Patches**

A set of orthomosaic patches.

**items()**

List of items.

**keys()**

List of item keys.

**values()**

List of item values.

**bands**

List of color bands.

**Type** list of string**bottom**

Y coordinate of the bottom side.

**Type** float**clear()**

Clears orthomosaic data.

**copy()**

Create a copy of the orthomosaic.

**Returns** Copy of the orthomosaic.**Return type** *Orthomosaic***crs**

Coordinate system of orthomosaic.

**Type** *CoordinateSystem***data\_type**

Data type used to store color values.

**Type** *DataType***height**

Orthomosaic height.

**Type** int**key**

Orthomosaic identifier.

**Type** int**label**

Orthomosaic label.

**Type** string**left**

X coordinate of the left side.

**Type** float

**meta**

Orthomosaic meta data.

**Type** *MetaData*

**modified**

Modified flag.

**Type** bool

**patches**

Orthomosaic patches.

**Type** *Orthomosaic.Patches*

**projection**

Orthomosaic projection.

**Type** *OrthoProjection*

**removeOrthophotos()**

Remove orthorectified images from orthomosaic.

**renderPreview**(*width = 2048, height = 2048*[, *progress* ])

Generate orthomosaic preview image. :arg width: Preview image width. :type width: int :arg height: Preview image height. :type height: int :arg progress: Progress callback. :type progress: Callable[[float], None] :return: Preview image. :rtype: *Image*

**reset**([*progress* ])

Reset all edits to orthomosaic.

**Parameters** **progress** (*Callable[[float], None]*) – Progress callback.

**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

**update**([*progress* ])

Apply edits to orthomosaic.

**Parameters** **progress** (*Callable[[float], None]*) – Progress callback.

**width**

Orthomosaic width.

**Type** int

**class** *Metashape.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**(*channels*][, *datatype* ])

Returns image data.

**Parameters**

- **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']
- **channels** (*string*) – color channels to be loaded, e.g. 'RGB', 'RGBA', etc.

**Returns** Image data.

**Return type** *Image*

**imageMeta()**

Returns image meta data.

**Returns** Image meta data.

**Return type** *MetaData*

**layer**

Layer index in the image file.

**Type** *int*

**meta**

Frame meta data.

**Type** *MetaData*

**open**(*path*, *layer=0*)

Loads specified image file.

**Parameters**

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

**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** *Metashape.PointClass*

Point class in [Created, Unclassified, Ground, LowVegetation, MediumVegetation, HighVegetation, Building, LowPoint, ModelKeyPoint, Water, Rail, RoadSurface, OverlapPoints, WireGuard, WireConductor, TransmissionTower, WireConnector, BridgeDeck, HighNoise, Car, Manmade]

**class** *Metashape.PointCloud*

Tie point cloud instance

**class Cameras**

Collection of *PointCloud.Projections* objects indexed by corresponding cameras

**class Filter**

Tie point cloud filter

The following example selects all tie points from the active chunk that have reprojection error higher than defined threshold:

```
>>> chunk = Metashape.app.document.chunk # active chunk
>>> threshold = 0.5
>>> f = Metashape.PointCloud.Filter()
>>> f.init(chunk, criterion = Metashape.PointCloud.Filter.ReprojectionError)
>>> f.selectPoints(threshold)
```

**class Criterion**

Point filtering criterion in [ReprojectionError, ReconstructionUncertainty, ImageCount, ProjectionAccuracy]

**init**(*points*, *criterion*, *progress*)

Initialize point cloud filter based on specified criterion.

**Parameters**

- **points** (*PointCloud* or *Chunk*) – Point cloud to filter.
- **criterion** (*PointCloud.Filter.Criterion*) – Point filter criterion.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**max\_value**

Maximum value.

**Type** int or double

**min\_value**

Minimum value.

**Type** int or double

**removePoints**(*threshold*)

Remove points based on specified threshold.

**Parameters** **threshold** (*float*) – Criterion threshold.

**resetSelection**()

Reset previously made selection.

**selectPoints**(*threshold*)

Select points based on specified threshold.

**Parameters** **threshold** (*float*) – Criterion threshold.

**values**

List of values.

**Type** list of int or list of double

**class Point**

3D point in the point cloud

**coord**

Point coordinates.

**Type** *Vector*

**cov**

Point coordinates covariance matrix.

**Type** *Matrix*

---

**selected**  
Point selection flag.  
**Type** bool

**track\_id**  
Track index.  
**Type** int

**valid**  
Point valid flag.  
**Type** bool

**class Points**  
Collection of 3D points in the point cloud

**copy()**  
Returns a copy of points buffer.  
**Returns** Copy of points buffer.  
**Return type** *PointCloud.Points*

**resize(count)**  
Resize points list.  
**Parameters** **count** (*int*) – new point count

**class Projection**  
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 Projections**  
Collection of *PointCloud.Projection* for the camera

**copy()**  
Returns a copy of projections buffer.  
**Returns** Copy of projections buffer.  
**Return type** *PointCloud.Projections*

**resize(count)**  
Resize projections list.  
**Parameters** **count** (*int*) – new projections count

**class Track**  
Track in the point cloud

**color**  
Track color.  
**Type** tuple of 3 int

**class Tracks**  
Collection of tracks in the point cloud

**copy()**

Returns a copy of tracks buffer.

**Returns** Copy of tracks buffer.

**Return type** *PointCloud.Tracks*

**resize(count)**

Resize track list.

**Parameters** **count** (*int*) – new track count

**bands**

List of color bands.

**Type** list of string

**cleanup([progress])**

Remove points with insufficient number of projections.

**Parameters** **progress** (*Callable[[float], None]*) – Progress callback.

**copy(keypoints=True)**

Returns a copy of the point cloud.

**Parameters** **keypoints** (*bool*) – copy key points data.

**Returns** Copy of the point cloud.

**Return type** *PointCloud*

**cropSelectedPoints()**

Crop selected points.

**cropSelectedTracks()**

Crop selected tie points.

**data\_type**

Data type used to store color values.

**Type** *DataType*

**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.

**meta**

Point cloud meta data.

**Type** *MetaData*

**modified**

Modified flag.

**Type** bool

**pickPoint(origin, target, endpoints=1)**

Returns ray intersection with the point cloud (point on the ray nearest to some point).

**Parameters**

- **origin** (*Vector*) – Ray origin.



- **target** (*Vector*) – Point on the ray.
- **endpoints** (*int*) – Number of endpoints to check for (0 - line, 1 - ray, 2 - segment).

**Returns** Coordinates of the intersection point.

**Return type** *Vector*

#### **points**

List of points.

**Type** *PointCloud.Points*

#### **projections**

Point projections for each photo.

**Type** *PointCloud.Projections*

#### **removeKeypoints()**

Remove keypoints from point cloud.

#### **removeSelectedPoints()**

Remove selected points.

#### **removeSelectedTracks()**

Remove selected tie points.

#### **renderDepth**(*transform, calibration, point\_size=1, cull\_points=False, add\_alpha=True*)

Render point cloud depth image for specified viewpoint.

##### **Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point\_size** (*int*) – Point size.
- **cull\_points** (*bool*) – Enable normal based culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** *Image*

#### **renderImage**(*transform, calibration, point\_size=1, cull\_points=False, add\_alpha=True, raster\_transform=RasterTransformNone*)

Render point cloud image for specified viewpoint.

##### **Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point\_size** (*int*) – Point size.
- **cull\_points** (*bool*) – Enable normal based culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.

**Returns** Rendered image.

**Return type** *Image*

**renderMask**(*transform*, *calibration*, *point\_size=1*, *cull\_points=False*)

Render point cloud mask image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point\_size** (*int*) – Point size.
- **cull\_points** (*bool*) – Enable normal based culling.

**Returns** Rendered image.

**Return type** *Image*

**renderNormalMap**(*transform*, *calibration*, *point\_size=1*, *cull\_points=False*, *add\_alpha=True*)

Render image with point cloud normals for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point\_size** (*int*) – Point size.
- **cull\_points** (*bool*) – Enable normal based culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** *Image*

**renderPreview**(*width = 2048*, *height = 2048*[, *transform* ], *point\_size=1*[, *progress* ])

Generate point cloud preview image.

**Parameters**

- **width** (*int*) – Preview image width.
- **height** (*int*) – Preview image height.
- **transform** (*Matrix*) – 4x4 viewpoint transformation matrix.
- **point\_size** (*int*) – Point size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**Returns** Preview image.

**Return type** *Image*

**tracks**

List of tracks.

**Type** *PointCloud.Tracks*

**class Metashape.PointsFormat**

Point cloud format in [PointsFormatNone, PointsFormatOBJ, PointsFormatPLY, PointsFormatXYZ, PointsFormatLAS, PointsFormatExpe, PointsFormatU3D, PointsFormatPDF, PointsFormatE57, PointsFormatOC3, PointsFormatPotree, PointsFormatLAZ, PointsFormatCL3, PointsFormatPTS, PointsFormatPTX, PointsFormatDXF, PointsFormatCesium, PointsFormatPCD, PointsFormatSLPK]

**class Metashape.Preselection**

Image pair preselection in [NoPreselection, GenericPreselection, ReferencePreselection]

**class** Metashape.RPCModel

Rational polynomial model.

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *RPCModel*

**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*

**image\_offset**

Image coordinate offset.

**Type** *Vector*

**image\_scale**

Image coordinate scale.

**Type** *Vector*

**line\_den\_coeff**

Line denominator.

**Type** *Vector*

**line\_num\_coeff**

Line numerator.

**Type** *Vector*

**load**(*path*)

Load RPC model from file.

**Parameters** **path** (*string*) – path to RPC model file

**object\_offset**

Object coordinate offset.

**Type** *Vector*

**object\_scale**

Object coordinate scale.

**Type** *Vector*

**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*

**samp\_den\_coeff**

Sample denominator.

Type *Vector*

**samp\_num\_coeff**

Sample numerator.

Type *Vector*

**save(*path*)**

Save RPC model to file.

Parameters **path** (*string*) – path to RPC model file

**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*

**class Metashape.RasterFormat**

Raster format in [RasterFormatNone, RasterFormatTiles, RasterFormatKMZ, RasterFormatXYZ, RasterFormatMBTiles, RasterFormatWW, RasterFormatTMS, RasterFormatGeoPackage]

**class Metashape.RasterTransform**

Raster transform definition.

**calibrateRange()**

Auto detect range based on orthomosaic histogram.

**copy()**

Return a copy of the object.

Returns A copy of the object.

Return type *RasterTransform*

**enabled**

Enable flag.

Type *bool*

**false\_color**

False color channels.

Type *list*

**formula**

Raster calculator expression.

Type *string*

**interpolation**

Interpolation enable flag.

Type *bool*

**palette**

Color palette.

Type *dict*

**range**

Palette mapping range.

**Type** tuple

**reset()**

Reset raster transform.

**class Metashape.RasterTransformType**

Raster transformation type in [RasterTransformNone, RasterTransformValue, RasterTransformPalette]

**class Metashape.ReferenceFormat**

Reference format in [ReferenceFormatNone, ReferenceFormatXML, ReferenceFormatTEL, ReferenceFormatCSV, ReferenceFormatMavinci, ReferenceFormatBramor, ReferenceFormatAPM]

**class Metashape.ReferenceItems**

Reference items in [ReferenceItemsCameras, ReferenceItemsMarkers, ReferenceItemsScalebars]

**class Metashape.ReferencePreselectionMode**

Reference preselection mode in [ReferencePreselectionSource, ReferencePreselectionEstimated, ReferencePreselectionSequential]

**class Metashape.Region**

Region parameters

**center**

Region center coordinates.

**Type** *Vector*

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *Region*

**rot**

Region rotation matrix.

**Type** *Matrix*

**size**

Region size.

**Type** *Vector*

**class Metashape.RotationOrder**

Rotation order in [RotationOrderXYZ, RotationOrderXZY, RotationOrderYXZ, RotationOrderYZX, RotationOrderZXY, RotationOrderZYX]

**class Metashape.Scalebar**

Scale bar instance

**class Reference**

Scale bar reference data

**accuracy**

Scale bar length accuracy.

**Type** float

**distance**

Scale bar length.

**Type** float

**enabled**

Enabled flag.

**Type** bool

**chunk**

Chunk the scalebar belongs to.

**Type** *Chunk*

**frames**

Scale bar frames.

**Type** list of *Scalebar*

**group**

Scale bar group.

**Type** *ScalebarGroup*

**key**

Scale bar identifier.

**Type** int

**label**

Scale bar label.

**Type** string

**meta**

Scale bar meta data.

**Type** *MetaData*

**point0**

Start of the scale bar.

**Type** *Marker*

**point1**

End of the scale bar.

**Type** *Marker*

**reference**

Scale bar reference data.

**Type** *ScalebarReference*

**selected**

Selects/deselects the scale bar.

**Type** bool

**class Metashape.ScalebarGroup**

ScalebarGroup objects define groups of multiple scale bars. The grouping is established by assignment of a ScalebarGroup instance to the Scalebar.group attribute of participating scale bars.

**label**

Scale bar group label.

**Type** string

**selected**

Current selection state.

**Type** bool

**class** Metashape.Sensor

Sensor instance

**class** Reference

Sensor reference data.

**accuracy**

Sensor location accuracy.

**Type** *Vector*

**enabled**

Location enabled flag.

**Type** bool

**location**

Sensor coordinates.

**Type** *Vector*

**location\_accuracy**

Sensor location accuracy.

**Type** *Vector*

**location\_enabled**

Location enabled flag.

**Type** bool

**rotation**

Sensor rotation angles.

**Type** *Vector*

**rotation\_accuracy**

Sensor rotation accuracy.

**Type** *Vector*

**rotation\_enabled**

Rotation enabled flag.

**Type** bool

**class** Type

Sensor type in [Frame, Fisheye, Spherical, Cylindrical, RPC]

**antenna**

GPS antenna correction.

**Type** *Antenna*

**bands**

List of color bands.

**Type** list of string

**black\_level**

Black level for each band.

**Type** list of float

**calibrateFiducials**(*resolution=0.014*)

Fit fiducial coordinates to image measurements.

**Parameters** **resolution** (*float*) – Scanning resolution in mm/pix.

**calibration**

Adjusted calibration of the photo.

Type *Calibration*

**chunk**

Chunk the sensor belongs to.

Type *Chunk*

**data\_type**

Data type used to store color values.

Type *DataType*

**fiducials**

Fiducial marks.

Type list of *Marker*

**film\_camera**

Film camera flag.

Type bool

**fixed**

Fix calibration flag.

Type bool

**fixed\_calibration**

Fix calibration flag.

Type bool

**fixed\_location**

Fix location flag.

Type bool

**fixed\_params**

List of fixed calibration parameters.

Type list of string

**fixed\_rotation**

Fix rotation flag.

Type bool

**focal\_length**

Focal length in mm.

Type float

**height**

Image height.

Type int

**key**

Sensor identifier.

Type int

**label**

Sensor label.



**Type** string

**layer\_index**

Sensor layer index.

**Type** int

**location**

Sensor plane location.

**Type** *Vector*

**location\_covariance**

Sensor plane location covariance.

**Type** *Matrix*

**makeMaster()**

Make this sensor master in the multi-camera system.

**master**

Master sensor.

**Type** *Sensor*

**meta**

Sensor meta data.

**Type** *MetaData*

**normalize\_sensitivity**

Enable sensitivity normalization.

**Type** bool

**normalize\_to\_float**

Convert pixel values to floating point after normalization.

**Type** bool

**photo\_params**

List of image-variant calibration parameters.

**Type** list of string

**pixel\_height**

Pixel height in mm.

**Type** float

**pixel\_size**

Pixel size in mm.

**Type** *Vector*

**pixel\_width**

Pixel width in mm.

**Type** float

**planes**

Sensor planes.

**Type** list of *Sensor*

**reference**

Sensor reference data.

**Type** SensorReference

**rolling\_shutter**

Enable rolling shutter compensation.

**Type** bool

**rotation**

Sensor plane rotation.

**Type** *Matrix*

**rotation\_covariance**

Sensor plane rotation covariance.

**Type** *Matrix*

**sensitivity**

Sensitivity for each band.

**Type** list of float

**type**

Sensor projection model.

**Type** *Sensor.Type*

**user\_calib**

Custom calibration used as initial calibration during photo alignment.

**Type** *Calibration*

**vignetting**

Vignetting for each band.

**Type** list of *Vignetting*

**width**

Image width.

**Type** int

**class** Metashape.**ServiceType**

Service type in [ServiceSketchfab, ServiceMapbox, Service4DMapper, ServicePointscene, ServiceMelown, ServicePointbox, ServicePicterra, ServiceCesium]

**class** Metashape.**Shape**

Shape data.

**class** **BoundaryType**

Shape boundary type in [NoBoundary, OuterBoundary, InnerBoundary]

**class** **Vertices**

Collection of shape vertices

**area()**

Return area of the shape on DEM.

**Returns** Shape area.

**Return type** float

**attributes**

Shape attributes.

**Type** *MetaData*

**boundary\_type**

Shape boundary type.

**Type** *Shape.BoundaryType*

**geometry**

Shape geometry.

**Type** *Geometry* or *AttachedGeometry*

**group**

Shape group.

**Type** *ShapeGroup*

**is\_attached**

Attached flag.

**Type** bool

**key**

Shape identifier.

**Type** int

**label**

Shape label.

**Type** string

**perimeter2D()**

Return perimeter of the shape on DEM.

**Returns** Shape perimeter.

**Return type** float

**perimeter3D()**

Return perimeter of the shape.

**Returns** Shape perimeter.

**Return type** float

**selected**

Selects/deselects the shape.

**Type** bool

**volume(*level='bestfit'*)**

Return volume of the shape measured on DEM above and below best fit, mean level or custom level plane.

**Parameters** **level** (*float*) – Plane level: 'bestfit', 'mean' or custom value.

**Returns** Shape volumes.

**Return type** dict

**class Metashape.ShapeGroup**

ShapeGroup objects define groups of multiple shapes. The grouping is established by assignment of a ShapeGroup instance to the Shape.group attribute of participating shapes.

**color**

Shape group color.

**Type** tuple of 4 int

**enabled**

Enable flag.

**Type** bool

**key**

Shape group identifier.

**Type** int

**label**

Shape group label.

**Type** string

**meta**

Shape group meta data.

**Type** *MetaData*

**selected**

Current selection state.

**Type** bool

**show\_labels**

Shape labels visibility flag.

**Type** bool

**class** *Metashape*.**Shapes**

A set of shapes for a chunk frame.

**addGroup()**

Add new shape group to the set of shapes.

**Returns** Created shape group.

**Return type** *ShapeGroup*

**addShape()**

Add new shape to the set of shapes.

**Returns** Created shape.

**Return type** *Shape*

**crs**

Shapes coordinate system.

**Type** *CoordinateSystem*

**group**

Default shape group.

**Type** *ShapeGroup*

**groups**

List of shape groups.

**Type** list of *ShapeGroup*

**items()**

List of items.

**meta**

Shapes meta data.

Type *MetaData*

**modified**

Modified flag.

Type *bool*

**projection**

Shapes projection.

Type *OrthoProjection*

**remove(*items*)**

Remove items from the shape layer.

**Parameters** *items* (list of *Shape* or *ShapeGroup*) – A list of items to be removed.

**shapes**

List of shapes.

Type list of *Shape*

**updateAltitudes(*items*[, *progress*])**

Update altitudes for items.

**Parameters**

- **items** (list of *Shape* or *ShapeGroup*) – A list of items to be updated.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**class** *Metashape.ShapesFormat*

Shapes format in [*ShapesFormatNone*, *ShapesFormatSHP*, *ShapesFormatKML*, *ShapesFormatDXF*, *ShapesFormatGeoJSON*, *ShapesFormatGeoPackage*, *ShapesFormatCSV*]

**class** *Metashape.Shutter*

Shutter object contains estimated parameters of the rolling shutter correction model.

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *Shutter*

**rotation**

Rotation matrix of the rolling shutter model.

Type *Matrix*

**translation**

Translation vector of the rolling shutter model.

Type *Vector*

**class** *Metashape.SurfaceType*

Surface type in [*Arbitrary*, *HeightField*]

**class** *Metashape.Target*

Target parameters

**code**

Target code.

Type *int*

**coord**

Target location.

**Type** *Vector*

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *Target*

**radius**

Target radius.

**Type** float

**class Metashape.TargetType**

Target type in [CircularTarget12bit, CircularTarget14bit, CircularTarget16bit, CircularTarget20bit, CircularTarget, CrossTarget]

**class Metashape.Tasks**

Task classes.

**class AddFrames**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**chunk**

Chunk to copy frames from.

**Type** int

**copy\_dense\_cloud**

Copy dense cloud.

**Type** bool

**copy\_depth\_maps**

Copy depth maps.

**Type** bool

**copy\_elevation**

Copy DEM.

**Type** bool

**copy\_model**

Copy model.

**Type** bool

**copy\_orthomosaic**

Copy orthomosaic.

**Type** bool

**copy\_tiled\_model**

Copy tiled model.

**Type** bool

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**frames**

List of frame keys to copy.

**Type** list of int

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class AddPhotos**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**filegroups**

List of file groups.

**Type** list of int

**filenames**

List of files to add.

**Type** list of string

**group**

Camera group key.

**Type** int

**layout**

Image layout.

**Type** *ImageLayout*

**load\_reference**

Load reference coordinates.

**Type** bool

**load\_rpc\_txt**

Load satellite RPC data from auxiliary TXT files.

**Type** bool

**load\_xmp\_accuracy**

Load accuracy from XMP meta data.

**Type** bool

**load\_xmp\_antenna**

Load GPS/INS offset from XMP meta data.

**Type** bool

**load\_xmp\_calibration**

Load calibration from XMP meta data.

**Type** bool

**load\_xmp\_orientation**

Load orientation from XMP meta data.

**Type** bool

**name**

Task name.

**Type** string

**strip\_extensions**

Strip file extensions from camera labels.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int



**class AlignCameras**

Task class containing processing parameters.

**adaptive\_fitting**

Enable adaptive fitting of distortion coefficients.

**Type** bool

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**cameras**

List of cameras to align.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**min\_image**

Minimum number of point projections.

**Type** int

**name**

Task name.

**Type** string

**reset\_alignment**

Reset current alignment.

**Type** bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class AlignChunks**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**chunks**

List of chunks to be aligned.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**downscale**

Alignment accuracy.

**Type** int

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**filter\_mask**

Filter points by mask.

**Type** bool

**fit\_scale**

Fit chunk scale during alignment.

**Type** bool

**generic\_preselection**

Enables image pair preselection.

**Type** bool

**keypoint\_limit**

Maximum number of points for each photo.

**Type** int

**markers**

List of markers to be used for marker based alignment.

**Type** list of int

**mask\_tiepoints**

Apply mask filter to tie points.

**Type** bool

**method**

Alignment method (0 - point based, 1 - marker based, 2 - camera based).

**Type** int

**name**

Task name.

**Type** string

**reference**

Chunk to be used as a reference.

**Type** int

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class AnalyzePhotos**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of cameras to be analyzed.

**Type** list of int

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**filter\_mask**

Constrain analyzed image region by mask.

**Type** bool

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*objects* )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class BuildContours**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**interval**

Contour interval.

**Type** float

**max\_value**

Maximum value of contour range.

**Type** float

**min\_value**

Minimum value of contour range.

**Type** float

**name**

Task name.

**Type** string

**prevent\_intersections**

Prevent contour intersections.

**Type** bool

**source\_data**

Source data for contour generation.

**Type** *DataSource*

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*objects* )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class BuildDem**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**classes**

List of dense point classes to be used for surface extraction.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**flip\_x**

Flip X axis direction.

**Type** bool

**flip\_y**

Flip Y axis direction.

**Type** bool

**flip\_z**

Flip Z axis direction.

**Type** bool

**interpolation**

Interpolation mode.

**Type** *Interpolation*

**max\_workgroup\_size**

Maximum workgroup size.

**Type** int

**name**

Task name.

**Type** string

**projection**

Output projection.

**Type** *OrthoProjection*

**region**

Region to be processed.

**Type** *BBox*

**resolution**

Output resolution in meters.

**Type** float

**source\_data**

Selects between dense point cloud and tie points.

**Type** *DataSource*

**subdivide\_task**

Enable fine-level task subdivision.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**workitem\_size\_tiles**

Number of tiles in a workitem.

**Type** int

**class BuildDenseCloud**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**keep\_depth**

Enable store depth maps option.

**Type** bool

**max\_neighbors**

Maximum number of neighbor images to use for depth map filtering.

**Type** int

**max\_workgroup\_size**

Maximum workgroup size.

**Type** int

**name**

Task name.

**Type** string

**point\_colors**

Enable point colors calculation.

**Type** bool

**point\_confidence**

Enable point confidence calculation.

**Type** bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**workitem\_size\_cameras**

Number of cameras in a workitem.

**Type** int

**class BuildDepthMaps**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(*json*)**

Initialize task parameters from a JSON string.

**downscale**

Depth map quality.

**Type** int

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**filter\_mode**

Depth map filtering mode.

**Type** *FilterMode*

**max\_neighbors**

Maximum number of neighbor images to use for depth map generation.

**Type** int

**max\_workgroup\_size**

Maximum workgroup size.

**Type** int

**name**

Task name.

**Type** string

**reuse\_depth**

Enable reuse depth maps option.

**Type** bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([*objects*])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**workitem\_size\_cameras**

Number of cameras in a workitem.

**Type** int

**class BuildModel**

Task class containing processing parameters.

**apply(*object*[, *workitem*][, *progress*])**

Apply task to specified object.



**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**classes**

List of dense point classes to be used for surface extraction.

**Type** list of int

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**face\_count**

Target face count.

**Type** *FaceCount*

**face\_count\_custom**

Custom face count.

**Type** int

**interpolation**

Interpolation mode.

**Type** *Interpolation*

**keep\_depth**

Enable store depth maps option.

**Type** bool

**max\_workgroup\_size**

Maximum workgroup size.

**Type** int

**name**

Task name.

**Type** string

**source\_data**

Selects between dense point cloud, tie points and depth maps.

**Type** *DataSource*

**subdivide\_task**

Enable fine-level task subdivision.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**surface\_type**

Type of object to be reconstructed.

**Type** *SurfaceType*

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**trimming\_radius**

Trimming radius (no trimming if zero).

**Type** *int*

**vertex\_colors**

Enable vertex colors calculation.

**Type** *bool*

**vertex\_confidence**

Enable vertex confidence calculation.

**Type** *bool*

**volumetric\_masks**

Enable strict volumetric masking.

**Type** *bool*

**workitem\_count**

Work item count.

**Type** *int*

**workitem\_size\_cameras**

Number of cameras in a workitem.

**Type** *int*

**class BuildOrthomosaic**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**blending\_mode**

Orthophoto blending mode.

**Type** *BlendingMode*

**cull\_faces**

Enable back-face culling.

**Type** *bool*

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**fill\_holes**

Enable hole filling.

**Type** bool

**flip\_x**

Flip X axis direction.

**Type** bool

**flip\_y**

Flip Y axis direction.

**Type** bool

**flip\_z**

Flip Z axis direction.

**Type** bool

**ghosting\_filter**

Enable ghosting filter.

**Type** bool

**max\_workgroup\_size**

Maximum workgroup size.

**Type** int

**name**

Task name.

**Type** string

**projection**

Output projection.

**Type** *OrthoProjection*

**refine\_seamlines**

Refine seamlines based on image content.

**Type** bool

**region**

Region to be processed.

**Type** *BBox*

**resolution**

Pixel size in meters.

**Type** float

**resolution\_x**

Pixel size in the X dimension in projected units.

**Type** float

**resolution\_y**

Pixel size in the Y dimension in projected units.

**Type** float

**subdivide\_task**

Enable fine-level task subdivision.

**Type** bool

**supports\_gpu**  
GPU support flag.  
**Type** bool

**surface\_data**  
Orthorectification surface.  
**Type** *DataSource*

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask**(*objects* )  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**workitem\_size\_cameras**  
Number of cameras in a workitem.  
**Type** int

**workitem\_size\_tiles**  
Number of tiles in a workitem.  
**Type** int

**class BuildPanorama**  
Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**blending\_mode**  
Panorama blending mode.  
**Type** *BlendingMode*

**camera\_groups**  
List of camera groups to process.  
**Type** list of int

**decode**(*dict*)  
Initialize task parameters with a dictionary.

**decodeJSON**(*json*)  
Initialize task parameters from a JSON string.

**encode**()  
Create a dictionary with task parameters.

**encodeJSON**()  
Create a JSON string with task parameters.

**frames**  
List of frames to process.  
**Type** list of int

**ghosting\_filter**

Enable ghosting filter.

**Type** bool

**height**

Height of output panorama.

**Type** int

**name**

Task name.

**Type** string

**region**

Region to be generated.

**Type** *BBox*

**rotation**

Panorama 3x3 orientation matrix.

**Type** *Matrix*

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**width**

Width of output panorama.

**Type** int

**workitem\_count**

Work item count.

**Type** int

**class BuildSeamlines**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**epsilon**

Contour simplification threshold.

**Type** float

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class BuildTexture**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**blending\_mode**

Texture blending mode.

**Type** *BlendingMode*

**cameras**

A list of cameras to be used for texturing.

**Type** list of int

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**fill\_holes**

Enable hole filling.

**Type** bool

**ghosting\_filter**

Enable ghosting filter.

**Type** bool

**name**

Task name.

**Type** string**source\_model**

Source model.

**Type** int**supports\_gpu**

GPU support flag.

**Type** bool**target**

Task target.

**Type** *Tasks.TargetType***texture\_size**

Texture page size.

**Type** int**texture\_type**

Texture type.

**Type** *Model.TextureType***toNetworkTask**(*[objects]*)Convert task to *NetworkTask* to be applied to specified objects.**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.**transfer\_texture**

Transfer texture.

**Type** bool**workitem\_count**

Work item count.

**Type** int**class BuildTiledModel**

Task class containing processing parameters.

**apply**(*object*[, *workitem*][, *progress*])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**classes**

List of dense point classes to be used for surface extraction.

**Type** list of int**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**face\_count**  
Number of faces per megapixel of texture resolution.  
**Type** int

**ghosting\_filter**  
Enable ghosting filter.  
**Type** bool

**keep\_depth**  
Enable store depth maps option.  
**Type** bool

**max\_workgroup\_size**  
Maximum workgroup size.  
**Type** int

**merge**  
Merge tiled model flag.  
**Type** bool

**name**  
Task name.  
**Type** string

**operand\_asset**  
Operand asset key.  
**Type** int

**operand\_chunk**  
Operand chunk key.  
**Type** int

**operand\_frame**  
Operand frame key.  
**Type** int

**pixel\_size**  
Target model resolution in meters.  
**Type** float

**source\_data**  
Selects between dense point cloud and mesh.  
**Type** *DataSource*

**subdivide\_task**  
Enable fine-level task subdivision.  
**Type** bool

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**tile\_size**  
Size of tiles in pixels.  
**Type** int



**toNetworkTask**(*objects* )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**transfer\_texture**

Transfer source model texture to tiled model.

**Type** bool

**workitem\_count**

Work item count.

**Type** int

**workitem\_size\_cameras**

Number of cameras in a workitem.

**Type** int

**class BuildUV**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**camera**

Camera to be used for texturing in MappingCamera mode.

**Type** int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**mapping\_mode**

Texture mapping mode.

**Type** *MappingMode*

**name**

Task name.

**Type** string

**page\_count**

Number of texture pages to generate.

**Type** int

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**texture\_size**

Expected size of texture page at texture generation step.

**Type** int

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class CalculatePointNormals**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**point\_neighbors**

Number of point neighbors to use for normal estimation.

**Type** int

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class CalibrateColors**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**source\_data**

Source data for calibration.

**Type** *DataSource*

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**white\_balance**

Calibrate white balance.

**Type** bool

**workitem\_count**

Work item count.

**Type** int

**class CalibrateLens**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**border**

Border size to ignore.

**Type** int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**fit\_b1**

Enable optimization of aspect ratio.

**Type** bool

**fit\_b2**

Enable optimization of skew coefficient.

**Type** bool

**fit\_cxcy**

Enable optimization of principal point coordinates.

**Type** bool

**fit\_f**

Enable optimization of focal length coefficient.

**Type** bool

**fit\_k1**

Enable optimization of k1 radial distortion coefficient.

**Type** bool

**fit\_k2**

Enable optimization of k2 radial distortion coefficient.

**Type** bool

**fit\_k3**

Enable optimization of k3 radial distortion coefficient.

**Type** bool

**fit\_k4**

Enable optimization of k4 radial distortion coefficient.

**Type** bool

**fit\_p1**

Enable optimization of p1 tangential distortion coefficient.

**Type** bool

**fit\_p2**

Enable optimization of p2 tangential distortion coefficient.

**Type** bool

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects ]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class CalibrateReflectance**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects ]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**use\_reflectance\_panels**

Use calibrated reflectance panels.

**Type** bool

**use\_sun\_sensor**

Apply irradiance sensor measurements.

**Type** bool

**workitem\_count**

Work item count.

**Type** int

**class ClassifyGroundPoints**

Task class containing processing parameters.

**apply**(*object*[, *workitem* ][, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**cell\_size**

Cell size (meters).

**Type** float

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**erosion\_radius**

Erosion radius (meters).

**Type** float

**max\_angle**

Maximum angle (degrees).

**Type** float

**max\_distance**

Maximum distance (meters).

**Type** float

**name**

Task name.

**Type** string

**source\_class**

Class of points to be re-classified.

**Type** int

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ClassifyPoints**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**confidence**

Required confidence level.

**Type** float

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**source\_class**

Class of points to be re-classified.

**Type** int

**subdivide\_task**

Enable fine-level task subdivision.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**target\_classes**

Target point classes for classification.

**Type** list of int

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class CloseHoles**

Task class containing processing parameters.

**apply**(*object*[, *workitem* ][, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**apply\_to\_selection**

Close holes within selection.

**Type** bool

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**level**

Hole size threshold in percents.

**Type** int

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ColorizeDenseCloud**

Task class containing processing parameters.

**apply**(*object*[, *workitem* ][, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.



**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**source\_data**

Source data to extract colors from.

**Type** *DataSource*

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ColorizeModel**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**source\_data**

Source data to extract colors from.

**Type** *DataSource*

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask**(*objects* )  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**class CompactDenseCloud**  
Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)  
Initialize task parameters with a dictionary.

**decodeJSON**(*json*)  
Initialize task parameters from a JSON string.

**encode**()  
Create a dictionary with task parameters.

**encodeJSON**()  
Create a JSON string with task parameters.

**name**  
Task name.  
**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask**(*objects* )  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**class ConvertImages**  
Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**color\_correction**

Apply color correction.

**Type** bool

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**fix\_pixel\_aspect**

Fix pixel aspect.

**Type** bool

**fix\_principal\_point**

Fix principal point.

**Type** bool

**image\_compression**

Image compression parameters.

**Type** *ImageCompression*

**merge\_planes**

Merge multispectral images.

**Type** bool

**name**

Task name.

**Type** string

**path**

Path to output file.

**Type** string

**remove\_distortions**

Remove distortions.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects ]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**update\_gps\_tags**

Update GPS tags.

**Type** bool

**use\_initial\_calibration**

Transform to initial calibration.

**Type** bool

**workitem\_count**

Work item count.

**Type** int

**class DecimateModel**

Task class containing processing parameters.

**apply**(*object*[, *workitem* ][, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**apply\_to\_selection**

Apply to selection.

**Type** bool

**asset**

Model to process.

**Type** int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**face\_count**

Target face count.

**Type** int

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects ]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class DetectFiducials**

Task class containing processing parameters.

**apply**(*object*[, *workitem* ][, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**frames**

List of frames to process.

**Type** list of int

**generate\_masks**

Generate background masks.

**Type** bool

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects ]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class DetectMarkers**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**filter\_mask**

Ignore masked image regions.

**Type** bool

**frames**

List of frames to process.

**Type** list of int

**inverted**

Detect markers on black background.

**Type** bool

**maximum\_residual**

Maximum residual for non-coded targets in pixels.

**Type** float

**minimum\_dist**

Minimum distance between targets in pixels (CrossTarget type only).

**Type** int

**minimum\_size**

Minimum target radius in pixels to be detected (CrossTarget type only).

**Type** int

**name**

Task name.

**Type** string

**noparity**

Disable parity checking.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**target\_type**

Type of targets.

**Type** *TargetType*

**toNetworkTask**(*objects* )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**tolerance**

Detector tolerance (0 - 100).

**Type** int

**workitem\_count**

Work item count.

**Type** int

**class DuplicateAsset**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**asset\_key**

Asset key.

**Type** int

**asset\_type**

Asset type.

**Type** *DataSource*

**clip\_to\_boundary**

Clip to boundary shapes.

**Type** bool

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects ]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class DuplicateChunk**

Task class containing processing parameters.

**apply**(*Object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**chunk**

Chunk to copy.

**Type** int

**copy\_dense\_clouds**

Copy dense clouds.

**Type** bool

**copy\_depth\_maps**

Copy depth maps.

**Type** bool

**copy\_elevations**

Copy DEMs.

**Type** bool

**copy\_keypoints**

Copy keypoints.

**Type** bool

**copy\_models**

Copy models.

**Type** bool

**copy\_orthomosaics**

Copy orthomosaics.

**Type** bool

**copy\_tiled\_models**

Copy tiled models.

**Type** bool

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.



**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**frames**

List of frame keys to copy.

**Type** list of int

**label**

New chunk label.

**Type** string

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ExportCameras**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**binary**

Enables/disables binary encoding for selected format (if applicable).

**Type** bool

**bingo\_path\_geoin**

Path to BINGO GEO INPUT file.

**Type** string

**bingo\_path\_gps**

Path to BINGO GPS/IMU file.

**Type** string

**bingo\_path\_image**

Path to BINGO IMAGE COORDINATE file.

**Type** string

**bingo\_path\_itera**  
Path to BINGO ITERA file.  
**Type** string

**bingo\_save\_geoin**  
Enables/disables export of BINGO GEO INPUT file.  
**Type** bool

**bingo\_save\_gps**  
Enables/disables export of BINGO GPS/IMU data.  
**Type** bool

**bingo\_save\_image**  
Enables/disables export of BINGO IMAGE COORDINATE file.  
**Type** bool

**bingo\_save\_itera**  
Enables/disables export of BINGO ITERA file.  
**Type** bool

**bundler\_path\_list**  
Path to Bundler image list file.  
**Type** string

**bundler\_save\_list**  
Enables/disables export of Bundler image list file.  
**Type** bool

**chan\_rotation\_order**  
Rotation order (CHAN format only).  
**Type** *RotationOrder*

**crs**  
Output coordinate system.  
**Type** *CoordinateSystem*

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**format**  
Export format.  
**Type** *CamerasFormat*

**image\_orientation**  
Image coordinate system (0 - X right, 1 - X up, 2 - X left, 3 - X down).  
**Type** int

**name**  
Task name.  
**Type** string

**path**  
Path to output file.

**Type** string

**save\_invalid\_matches**

Enables/disables export of invalid image matches.

**Type** bool

**save\_markers**

Enables/disables export of manual matching points.

**Type** bool

**save\_points**

Enables/disables export of automatic tie points.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**use\_initial\_calibration**

Transform image coordinates to initial calibration.

**Type** bool

**use\_labels**

Enables/disables label based item identifiers.

**Type** bool

**workitem\_count**

Work item count.

**Type** int

**class ExportDepth**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**export\_depth**

Enable export of depth map.

**Type** bool

**export\_diffuse**

Enable export of diffuse map.

**Type** bool

**export\_normals**

Enable export of normal map.

**Type** bool

**name**

Task name.

**Type** string

**path\_depth**

Path to depth map.

**Type** string

**path\_diffuse**

Path to diffuse map.

**Type** string

**path\_normals**

Path to normal map.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ExportMarkers**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**binary**

Enables/disables binary encoding for selected format (if applicable).

**Type** bool

**crs**  
Output coordinate system.  
**Type** *CoordinateSystem*

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**name**  
Task name.  
**Type** string

**path**  
Path to output file.  
**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask([objects])**  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**class ExportMasks**  
Task class containing processing parameters.

**apply(object[, workitem][, progress])**  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**  
List of cameras to process.  
**Type** list of int

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**path**

Path to output file.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ExportModel**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**binary**

Enables/disables binary encoding (if supported by format).

**Type** bool

**clip\_to\_boundary**

Clip model to boundary shapes.

**Type** bool

**colors\_rgb\_8bit**

Convert colors to 8 bit RGB.

**Type** bool

**comment**

Optional comment (if supported by selected format).

**Type** string

**crs**

Output coordinate system.

**Type** *CoordinateSystem*

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(*json*)**

Initialize task parameters from a JSON string.

**embed\_texture**

Embeds texture inside the model file (if supported by format).

**Type** bool

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

Export format.

**Type** *ModelFormat*

**name**

Task name.

**Type** string

**path**

Path to output model.

**Type** string

**precision**

Number of digits after the decimal point (for text formats).

**Type** int

**raster\_transform**

Raster band transformation.

**Type** *RasterTransformType*

**save\_alpha**

Enables/disables alpha channel export.

**Type** bool

**save\_cameras**

Enables/disables camera export.

**Type** bool

**save\_colors**

Enables/disables export of vertex colors.

**Type** bool

**save\_comment**

Enables/disables comment export.

**Type** bool

**save\_confidence**

Enables/disables export of vertex confidence.

**Type** bool

**save\_markers**

Enables/disables marker export.

**Type** bool

**save\_normals**

Enables/disables export of vertex normals.

**Type** bool

**save\_texture**

Enables/disables texture export.

**Type** bool

**save\_udim**

Enables/disables UDIM texture layout.

**Type** bool

**save\_uv**

Enables/disables uv coordinates export.

**Type** bool

**shift**

Optional shift to be applied to vertex coordinates.

**Type** *Vector*

**strip\_extensions**

Strips camera label extensions during export.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**texture\_format**

Texture format.

**Type** *ImageFormat*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**viewpoint**

Default view.

**Type** *Viewpoint*

**workitem\_count**

Work item count.

**Type** int

**class ExportOrthophotos**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode(dict)**

Initialize task parameters with a dictionary.



**decodeJSON(*json*)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**image\_compression**  
Image compression parameters.  
**Type** *ImageCompression*

**name**  
Task name.  
**Type** string

**north\_up**  
Use north-up orientation for export.  
**Type** bool

**path**  
Path to output orthophoto.  
**Type** string

**projection**  
Output projection.  
**Type** *OrthoProjection*

**raster\_transform**  
Raster band transformation.  
**Type** *RasterTransformType*

**region**  
Region to be exported.  
**Type** *BBox*

**resolution**  
Output resolution in meters.  
**Type** float

**resolution\_x**  
Pixel size in the X dimension in projected units.  
**Type** float

**resolution\_y**  
Pixel size in the Y dimension in projected units.  
**Type** float

**save\_alpha**  
Enable alpha channel generation.  
**Type** bool

**save\_kml**  
Enable kml file generation.  
**Type** bool

**save\_world**  
Enable world file generation.  
**Type** bool

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects]*)  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**white\_background**  
Enable white background.  
**Type** bool

**workitem\_count**  
Work item count.  
**Type** int

**class ExportPoints**  
Task class containing processing parameters.

**apply**(*object*[, *workitem*][, *progress*])  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**binary**  
Enables/disables binary encoding for selected format (if applicable).  
**Type** bool

**block\_height**  
Block height in meters.  
**Type** float

**block\_width**  
Block width in meters.  
**Type** float

**classes**  
List of dense point classes to be exported.  
**Type** list of int

**clip\_to\_boundary**  
Clip point cloud to boundary shapes.  
**Type** bool

**colors\_rgb\_8bit**  
Convert colors to 8 bit RGB.  
**Type** bool

**comment**  
Optional comment (if supported by selected format).  
**Type** string

**compression**  
Enable compression (Cesium format only).  
**Type** bool

---

**crs**  
Output coordinate system.  
**Type** *CoordinateSystem*

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**format**  
Export format.  
**Type** *PointsFormat*

**image\_format**  
Image data format.  
**Type** *ImageFormat*

**name**  
Task name.  
**Type** string

**path**  
Path to output file.  
**Type** string

**raster\_transform**  
Raster band transformation.  
**Type** *RasterTransformType*

**region**  
Region to be exported.  
**Type** *BBox*

**save\_classes**  
Enables/disables export of point classes.  
**Type** bool

**save\_colors**  
Enables/disables export of point colors.  
**Type** bool

**save\_comment**  
Enable comment export.  
**Type** bool

**save\_confidence**  
Enables/disables export of point confidence.  
**Type** bool

**save\_images**  
Enable image export.  
**Type** bool

**save\_normals**  
Enables/disables export of point normals.

**Type** bool

**screen\_space\_error**

Target screen space error (Cesium format only).

**Type** float

**shift**

Optional shift to be applied to point coordinates.

**Type** *Vector*

**source\_data**

Selects between dense point cloud and tie points. If not specified, uses dense cloud if available.

**Type** *DataSource*

**split\_in\_blocks**

Enable tiled export.

**Type** bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*objects* )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**viewpoint**

Default view.

**Type** *Viewpoint*

**workitem\_count**

Work item count.

**Type** int

**class ExportRaster**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**block\_height**

Raster block height in pixels.

**Type** int

**block\_width**

Raster block width in pixels.

**Type** int

**clip\_to\_boundary**

Clip raster to boundary shapes.

**Type** bool

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**description**

Export description.

**Type** string

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

Export format.

**Type** *RasterFormat*

**height**

Raster height.

**Type** int

**image\_compression**

Image compression parameters.

**Type** *ImageCompression*

**image\_description**

Optional description to be added to image files.

**Type** string

**image\_format**

Tile format.

**Type** *ImageFormat*

**max\_zoom\_level**

Maximum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).

**Type** int

**min\_zoom\_level**

Minimum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).

**Type** int

**name**

Task name.

**Type** string

**network\_links**

Enable network links generation for KMZ format.

**Type** bool

**nodata\_value**

No-data value (DEM export only).

**Type** float

**north\_up**

Use north-up orientation for export.

**Type** bool

**path**  
Path to output orthomosaic.  
**Type** string

**projection**  
Output projection.  
**Type** *OrthoProjection*

**raster\_transform**  
Raster band transformation.  
**Type** *RasterTransformType*

**region**  
Region to be exported.  
**Type** *BBox*

**resolution**  
Output resolution in meters.  
**Type** float

**resolution\_x**  
Pixel size in the X dimension in projected units.  
**Type** float

**resolution\_y**  
Pixel size in the Y dimension in projected units.  
**Type** float

**save\_alpha**  
Enable alpha channel generation.  
**Type** bool

**save\_kml**  
Enable kml file generation.  
**Type** bool

**save\_scheme**  
Enable tile scheme files generation.  
**Type** bool

**save\_world**  
Enable world file generation.  
**Type** bool

**source\_data**  
Selects between DEM and orthomosaic.  
**Type** *DataSource*

**split\_in\_blocks**  
Split raster in blocks.  
**Type** bool

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**tile\_height**

Tile height in pixels.

**Type** int

**tile\_width**

Tile width in pixels.

**Type** int

**title**

Export title.

**Type** string

**toNetworkTask**(*objects* )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**white\_background**

Enable white background.

**Type** bool

**width**

Raster width.

**Type** int

**workitem\_count**

Work item count.

**Type** int

**world\_transform**

2x3 raster-to-world transformation matrix.

**Type** *Matrix*

**class ExportReference**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**columns**

Column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, u/v/w - estimated coordinates, U/V/W - coordinate errors, d/e/f - estimated orientation angles, D/E/F - orientation errors, p/q/r - estimated coordinates variance, i/j/k - estimated orientation angles variance, [] - group of multiple values, | - column separator within group).

**Type** string

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**delimiter**

Column delimiter in csv format.

**Type** string

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**format**  
Export format.  
**Type** *ReferenceFormat*

**items**  
Items to export in CSV format.  
**Type** *ReferenceItems*

**name**  
Task name.  
**Type** string

**path**  
Path to the output file.  
**Type** string

**precision**  
Number of digits after the decimal point (for CSV format).  
**Type** int

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask([objects])**  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**class ExportReport**  
Task class containing processing parameters.

**apply(object[, workitem][, progress])**  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**description**  
Report description.  
**Type** string



---

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**font\_size**  
Font size (pt).  
**Type** int

**include\_system\_info**  
Include system information.  
**Type** bool

**name**  
Task name.  
**Type** string

**page\_numbers**  
Enable page numbers.  
**Type** bool

**path**  
Path to output report.  
**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**title**  
Report title.  
**Type** string

**toNetworkTask([objects])**  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**user\_settings**  
A list of user defined settings to include on the Processing Parameters page.  
**Type** list of (string, string) tuples

**workitem\_count**  
Work item count.  
**Type** int

**class ExportShapes**  
Task class containing processing parameters.

**apply(object[, workitem][, progress])**  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**crs**  
Output coordinate system.  
**Type** *CoordinateSystem*

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**format**  
Export format.  
**Type** *ShapesFormat*

**groups**  
A list of shape groups to export.  
**Type** list of int

**name**  
Task name.  
**Type** string

**path**  
Path to shape file.  
**Type** string

**polygons\_as\_polylines**  
Save polygons as polylines.  
**Type** bool

**save\_attributes**  
Export attributes.  
**Type** bool

**save\_labels**  
Export labels.  
**Type** bool

**save\_points**  
Export points.  
**Type** bool

**save\_polygons**  
Export polygons.  
**Type** bool

**save\_polylines**  
Export polylines.  
**Type** bool

**shift**  
Optional shift to be applied to vertex coordinates.  
**Type** *Vector*

**supports\_gpu**  
GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects ]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ExportTexture**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**path**

Path to output file.

**Type** string

**save\_alpha**

Enable alpha channel export.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**texture\_type**

Texture type.

**Type** *Model.TextureType*

**toNetworkTask**(*[objects ]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ExportTiledModel**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**clip\_to\_boundary**

Clip tiled model to boundary shapes.

**Type** bool

**crs**

Output coordinate system.

**Type** *CoordinateSystem*

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**format**

Export format.

**Type** *TiledModelFormat*

**image\_compression**

Image compression parameters.

**Type** *ImageCompression*

**model\_compression**

Enable mesh compression (Cesium format only).

**Type** bool

**model\_format**

Model format for zip export.

**Type** *ModelFormat*

**name**

Task name.

**Type** string

**path**

Path to output model.

**Type** string

**raster\_transform**

Raster band transformation.

**Type** *RasterTransformType*

**screen\_space\_error**

Target screen space error (Cesium format only).

**Type** float

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**texture\_format**

Texture format.

**Type** *ImageFormat*

**toNetworkTask**(*[objects]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class FilterDenseCloud**

Task class containing processing parameters.

**apply**(*object*[, *workitem*][, *progress*])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**asset**

Dense cloud key to filter.

**Type** int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**point\_spacing**

Desired point spacing (m).

**Type** float

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects]*)

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class GenerateMasks**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**blur\_threshold**

Allowed blur radius on a photo in pix (only if *mask\_defocus*=True).

**Type** float

**cameras**

Optional list of cameras to be processed.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**depth\_threshold**

Maximum depth of masked areas in meters (only if *mask\_defocus*=False).

**Type** float

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**fix\_coverage**

Extend masks to cover whole mesh (only if *mask\_defocus*=True).

**Type** bool

**mask\_defocus**

Mask defocus areas.

**Type** bool

**mask\_operation**

Mask operation.

**Type** *MaskOperation*

**masking\_mode**

Mask generation mode.

Type *MaskingMode*

**name**

Task name.

Type string

**path**

Mask file name template.

Type string

**supports\_gpu**

GPU support flag.

Type bool

**target**

Task target.

Type *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

Parameters **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**tolerance**

Background masking tolerance.

Type int

**workitem\_count**

Work item count.

Type int

**class GeneratePrescriptionMap**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**boundary\_shape\_group**

Boundary shape group.

Type int

**breakpoints**

Classification breakpoints.

Type list of float

**cell\_size**

Step of prescription grid, meters.

Type float

**class\_count**

Number of classes.

Type int

**classification\_method**

Index values classification method.

Type *ClassificationMethod*

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**rates**

Fertilizer rate for each class.

**Type** list of float

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ImportCameras**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**crs**

Ground coordinate system.

**Type** *CoordinateSystem*

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.



**format**  
File format.  
**Type** *CamerasFormat*

**image\_list**  
Path to image list file (Bundler format only).  
**Type** string

**image\_orientation**  
Image coordinate system (0 - X right, 1 - X up, 2 - X left, 3 - X down).  
**Type** int

**load\_image\_list**  
Enable Bundler image list import.  
**Type** bool

**name**  
Task name.  
**Type** string

**path**  
Path to the file.  
**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask([objects])**  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**class ImportLaserScans**  
Task class containing processing parameters.

**apply(object[, workitem][, progress])**  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**filenames**

List of files to import.

**Type** list of string

**format**

Point cloud format.

**Type** *PointsFormat*

**image\_path**

Path template to output files.

**Type** string

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ImportMarkers**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**path**

Path to the file.

**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask**(*objects* )  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

#### class ImportModel

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])  
Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**crs**  
Model coordinate system.  
**Type** *CoordinateSystem*

**decode**(*dict*)  
Initialize task parameters with a dictionary.

**decodeJSON**(*json*)  
Initialize task parameters from a JSON string.

**decode\_udim**  
Load UDIM texture layout.  
**Type** bool

**encode**()  
Create a dictionary with task parameters.

**encodeJSON**()  
Create a JSON string with task parameters.

**format**  
Model format.  
**Type** *ModelFormat*

**name**  
Task name.  
**Type** string

**path**  
Path to model.  
**Type** string

**shift**  
Optional shift to be applied to vertex coordinates.  
**Type** *Vector*

**supports\_gpu**

GPU support flag.

**Type** bool**target**

Task target.

**Type** *Tasks.TargetType***toNetworkTask([objects ])**Convert task to *NetworkTask* to be applied to specified objects.**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.**workitem\_count**

Work item count.

**Type** int**class ImportPoints**

Task class containing processing parameters.

**apply(object[, workitem ][, progress ])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**calculate\_normals**

Calculate point normals.

**Type** bool**crs**

Point cloud coordinate system.

**Type** *CoordinateSystem***decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

Point cloud format.

**Type** *PointsFormat***name**

Task name.

**Type** string**path**

Path to point cloud.

**Type** string**point\_neighbors**

Number of point neighbors to use for normal estimation.

**Type** int

**shift**

Optional shift to be applied to point coordinates.

**Type** *Vector*

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ImportRaster**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**crs**

Default coordinate system if not specified in GeoTIFF file.

**Type** *CoordinateSystem*

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**has\_nodata\_value**

No-data value valid flag.

**Type** bool

**name**

Task name.

**Type** string

**nodata\_value**

No-data value.

**Type** float

**path**

Path to elevation model in GeoTIFF format.

**Type** string

**raster\_type**

Type of raster layer to import.

**Type** *DataSource*

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ImportReference**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**columns**

Column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, [] - group of multiple values, | - column separator within group).

**Type** string

**create\_markers**

Create markers for missing entries (csv format only).

**Type** bool

**crs**

Reference data coordinate system (csv format only).

**Type** *CoordinateSystem*

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**delimiter**

Column delimiter in csv format.

**Type** string

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

File format.

**Type** *ReferenceFormat***group\_delimiters**

Combine consecutive delimiters in csv format.

**Type** bool**ignore\_labels**

Matches reference data based on coordinates alone (csv format only).

**Type** bool**items**

List of items to load reference for (csv format only).

**Type** *ReferenceItems***name**

Task name.

**Type** string**path**

Path to the file with reference data.

**Type** string**shutter\_lag**

Shutter lag in seconds (APM format only).

**Type** float**skip\_rows**

Number of rows to skip in (csv format only).

**Type** int**supports\_gpu**

GPU support flag.

**Type** bool**target**

Task target.

**Type** *Tasks.TargetType***threshold**

Error threshold in meters used when ignore\_labels is set (csv format only).

**Type** float**toNetworkTask([objects])**Convert task to *NetworkTask* to be applied to specified objects.**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.**workitem\_count**

Work item count.

**Type** int**class ImportShapes**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**boundary\_type**

Boundary type to be applied to imported shapes.

**Type** *Shape.BoundaryType*

**columns**

Column order in csv format (n - label, x/y/z - coordinates, d - description, [] - group of multiple values, | - column separator within group).

**Type** string

**crs**

Reference data coordinate system (csv format only).

**Type** *CoordinateSystem*

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**delimiter**

Column delimiter in csv format.

**Type** string

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

Shapes format.

**Type** *ShapesFormat*

**group\_delimiters**

Combine consecutive delimiters in csv format.

**Type** bool

**name**

Task name.

**Type** string

**path**

Path to shape file.

**Type** string

**replace**

Replace current shapes with new data.

**Type** bool

**skip\_rows**

Number of rows to skip in (csv format only).

**Type** int

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*



**toNetworkTask**(*objects* )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ImportTiledModel**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**path**

Path to tiled model.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**(*objects* )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class InvertMasks**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.

- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class LoadProject**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string**path**

Path to project file.

**Type** string**read\_only**

Open project in read only mode.

**Type** bool**supports\_gpu**

GPU support flag.

**Type** bool**target**

Task target.

**Type** *Tasks.TargetType***toNetworkTask([objects])**Convert task to *NetworkTask* to be applied to specified objects.**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.**workitem\_count**

Work item count.

**Type** int**class MatchPhotos**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of cameras to match.

**Type** list of int**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**downscale**

Image alignment accuracy.

**Type** int**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**filter\_mask**

Filter points by mask.

**Type** bool

**filter\_stationary\_points**  
Exclude tie points which are stationary across images.  
**Type** bool

**generic\_preselection**  
Enable generic preselection.  
**Type** bool

**guided\_matching**  
Enable guided image matching.  
**Type** bool

**keep\_keypoints**  
Store keypoints in the project.  
**Type** bool

**keypoint\_limit**  
Key point limit.  
**Type** int

**keypoint\_limit\_per\_mpx**  
Key point limit per megapixel.  
**Type** int

**mask\_tiepoints**  
Apply mask filter to tie points.  
**Type** bool

**max\_workgroup\_size**  
Maximum workgroup size.  
**Type** int

**name**  
Task name.  
**Type** string

**pairs**  
User defined list of camera pairs to match.  
**Type** list of (int, int) tuples

**reference\_preselection**  
Enable reference preselection.  
**Type** bool

**reference\_preselection\_mode**  
Reference preselection mode.  
**Type** *ReferencePreselectionMode*

**reset\_matches**  
Reset current matches.  
**Type** bool

**subdivide\_task**  
Enable fine-level task subdivision.  
**Type** bool

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**tiepoint\_limit**  
Tie point limit.  
**Type** int

**toNetworkTask([objects])**  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**workitem\_size\_cameras**  
Number of cameras in a workitem.  
**Type** int

**workitem\_size\_pairs**  
Number of image pairs in a workitem.  
**Type** int

**class MergeAssets**  
Task class containing processing parameters.

**apply(object[, workitem][, progress])**  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**assets**  
List of assets to process.  
**Type** list of int

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**name**  
Task name.  
**Type** string

**source\_data**  
Asset type.  
**Type** *DataSource*

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType***toNetworkTask**(*[objects ]*)Convert task to *NetworkTask* to be applied to specified objects.**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.**workitem\_count**

Work item count.

**Type** *int***class MergeChunks**

Task class containing processing parameters.

**apply**(*object*[, *workitem* ][, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**chunks**

List of chunks to process.

**Type** *list of int***decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**merge\_dense\_clouds**

Merge dense clouds.

**Type** *bool***merge\_depth\_maps**

Merge depth maps.

**Type** *bool***merge\_elevations**

Merge DEMs.

**Type** *bool***merge\_markers**

Merge markers.

**Type** *bool***merge\_models**

Merge models.

**Type** *bool***merge\_orthomosaics**

Merge orthomosaics.

**Type** *bool*

**merge\_tiepoints**

Merge tie points.

**Type** bool**name**

Task name.

**Type** string**supports\_gpu**

GPU support flag.

**Type** bool**target**

Task target.

**Type** *Tasks.TargetType***toNetworkTask([objects])**Convert task to *NetworkTask* to be applied to specified objects.**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.**workitem\_count**

Work item count.

**Type** int**class OptimizeCameras**

Task class containing processing parameters.

**adaptive\_fitting**

Enable adaptive fitting of distortion coefficients.

**Type** bool**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**fit\_b1**

Enable optimization of aspect ratio.

**Type** bool**fit\_b2**

Enable optimization of skew coefficient.

**Type** bool**fit\_corrections**

Enable optimization of additional corrections.

**Type** bool

**fit\_cx**  
Enable optimization of X principal point coordinates.  
**Type** bool

**fit\_cy**  
Enable optimization of Y principal point coordinates.  
**Type** bool

**fit\_f**  
Enable optimization of focal length coefficient.  
**Type** bool

**fit\_k1**  
Enable optimization of k1 radial distortion coefficient.  
**Type** bool

**fit\_k2**  
Enable optimization of k2 radial distortion coefficient.  
**Type** bool

**fit\_k3**  
Enable optimization of k3 radial distortion coefficient.  
**Type** bool

**fit\_k4**  
Enable optimization of k3 radial distortion coefficient.  
**Type** bool

**fit\_p1**  
Enable optimization of p1 tangential distortion coefficient.  
**Type** bool

**fit\_p2**  
Enable optimization of p2 tangential distortion coefficient.  
**Type** bool

**name**  
Task name.  
**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**tiepoint\_covariance**  
Estimate tie point covariance matrices.  
**Type** bool

**toNetworkTask([objects])**  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int



**class PlanMission**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**attach\_viewpoints**

Generate additional viewpoints to increase coverage.

**Type** bool

**capture\_distance**

Image capture distance (m).

**Type** float

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**group\_attached\_viewpoints**

Ignore minimum waypoint spacing for additional viewpoints.

**Type** bool

**home\_point**

Home point shape key.

**Type** int

**horizontal\_zigzags**

Cover surface with horizontal zigzags instead of vertical.

**Type** bool

**interesting\_zone**

Interesting zone shape layer key.

**Type** int

**max\_pitch**

Maximum camera pitch angle.

**Type** int

**min\_altitude**

Minimum altitude (m).

**Type** float

**min\_pitch**

Minimum camera pitch angle.

**Type** int

**min\_waypoint\_spacing**

Minimum waypoint spacing (m).

**Type** float

**name**

Task name.

**Type** string**overlap**

Overlap percent.

**Type** int**powerlines**

Powerlines shape layer key.

**Type** int**restricted\_zone**

Restricted zone shape layer key.

**Type** int**safety\_distance**

Safety distance (m).

**Type** float**safety\_zone**

Safety zone shape layer key.

**Type** int**sensor**

Sensor key.

**Type** int**supports\_gpu**

GPU support flag.

**Type** bool**target**

Task target.

**Type** *Tasks.TargetType***toNetworkTask**(*objects* )Convert task to *NetworkTask* to be applied to specified objects.**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.**use\_selection**

Focus on model selection.

**Type** bool**workitem\_count**

Work item count.

**Type** int**class ReduceOverlap**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**overlap**

Target number of cameras observing each point of the surface.

**Type** int

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**use\_selection**

Focus on model selection.

**Type** bool

**workitem\_count**

Work item count.

**Type** int

**class RefineMesh**

Task class containing processing parameters.

**apply**(*object* [, *workitem* ] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**downscale**

Refinement quality.

**Type** int

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**iterations**

Number of refinement iterations.

**Type** int

**name**

Task name.

**Type** string

**smoothness**

Smoothing strength. Should be in range [0, 1].

**Type** float

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class RemoveLighting**

Task class containing processing parameters.

**ambient\_occlusion\_multiplier**

Ambient occlusion multiplier. Should be in range [0.25, 4].

**Type** float

**ambient\_occlusion\_path**

Path to ambient occlusion texture atlas. Can be empty.

**Type** string

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**color\_mode**

Enable multi-color processing mode.

**Type** bool

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**internal\_blur**

Internal blur. Should be in range [0, 4].

**Type** float

**mesh\_noise\_suppression**

Mesh normals noise suppression strength. Should be in range [0, 4].

**Type** float

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class ResetMasks**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of cameras to process.

**Type** list of int

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects]*)  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**class RunScript**  
Task class containing processing parameters.

**apply**(*object*[, *workitem* ][, *progress* ])  
Apply task to specified object.  
**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**args**  
Script arguments.  
**Type** string

**code**  
Script code.  
**Type** string

**decode**(*dict*)  
Initialize task parameters with a dictionary.

**decodeJSON**(*json*)  
Initialize task parameters from a JSON string.

**encode**()  
Create a dictionary with task parameters.

**encodeJSON**()  
Create a JSON string with task parameters.

**name**  
Task name.  
**Type** string

**path**  
Script path.  
**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class SaveProject**

Task class containing processing parameters.

**apply**(*object*[, *workitem* ][, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**chunks**

List of chunks to be saved.

**Type** list of int

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

**Type** string

**path**

Path to project.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** *objects* (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**version**

Project version to save.

**Type** string

**workitem\_count**

Work item count.

**Type** int

**class SmoothModel**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**apply\_to\_selection**

Apply to selected faces.

**Type** bool

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**fix\_borders**

Fix borders.

**Type** bool

**name**

Task name.

**Type** string

**preserve\_edges**

Preserve edges.

**Type** bool

**strength**

Smoothing strength.

**Type** float

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**([*objects* ])

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class TargetType**

Task target type in [DocumentTarget, ChunkTarget, FrameTarget]



**class TrackMarkers**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**first\_frame**

Starting frame index.

**Type** int

**last\_frame**

Ending frame index.

**Type** int

**name**

Task name.

**Type** string

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask**( [*objects* ] )

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type** int

**class TransformRaster**

Task class containing processing parameters.

**apply**(*object* [, *workitem*] [, *progress* ])

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**asset**  
Asset key to transform.  
**Type** int

**data\_source**  
Selects between DEM and orthomosaic.  
**Type** *DataSource*

**decode(dict)**  
Initialize task parameters with a dictionary.

**decodeJSON(json)**  
Initialize task parameters from a JSON string.

**encode()**  
Create a dictionary with task parameters.

**encodeJSON()**  
Create a JSON string with task parameters.

**height**  
Raster height.  
**Type** int

**name**  
Task name.  
**Type** string

**nodata\_value**  
No-data value (DEM export only).  
**Type** float

**north\_up**  
Use north-up orientation for export.  
**Type** bool

**operand\_asset**  
Operand asset key.  
**Type** int

**operand\_chunk**  
Operand chunk key.  
**Type** int

**operand\_frame**  
Operand frame key.  
**Type** int

**projection**  
Output projection.  
**Type** *OrthoProjection*

**region**  
Region to be processed.  
**Type** *BBox*

**resolution**  
Output resolution in meters.  
**Type** float

**resolution\_x**  
Pixel size in the X dimension in projected units.

**Type** float

**resolution\_y**

Pixel size in the Y dimension in projected units.

**Type** float

**subtract**

Subtraction flag.

**Type** bool

**supports\_gpu**

GPU support flag.

**Type** bool

**target**

Task target.

**Type** *Tasks.TargetType*

**toNetworkTask([objects])**

Convert task to *NetworkTask* to be applied to specified objects.

**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**width**

Raster width.

**Type** int

**workitem\_count**

Work item count.

**Type** int

**world\_transform**

2x3 raster-to-world transformation matrix.

**Type** *Matrix*

**class TriangulatePoints**

Task class containing processing parameters.

**apply(object[, workitem][, progress])**

Apply task to specified object.

**Parameters**

- **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**max\_error**

Reprojection error threshold.

**Type** float

**min\_image**

Minimum number of point projections.

**Type** int

**name**  
Task name.  
**Type** string

**supports\_gpu**  
GPU support flag.  
**Type** bool

**target**  
Task target.  
**Type** *Tasks.TargetType*

**toNetworkTask**(*[objects]*)  
Convert task to *NetworkTask* to be applied to specified objects.  
**Parameters** **objects** (*Document*, *Chunk* or list of *Chunk*) – Objects to be processed.

**workitem\_count**  
Work item count.  
**Type** int

**createTask**(*name*)  
Create task object by its name.  
**Parameters** **name** (*string*) – Task name.  
**Returns** Task object.  
**Return type** object

**class** Metashape.**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.

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

**class** Metashape.**Thumbnails**  
A set of thumbnails generated for a chunk frame.

**items()**

List of items.

**keys()**

List of item keys.

**meta**

Thumbnails meta data.

**Type** *MetaData*

**modified**

Modified flag.

**Type** bool

**values()**

List of item values.

**class** `Metashape.TiledModel`

Tiled model data.

**class** `FaceCount`

Tiled model face count in [LowFaceCount, MediumFaceCount, HighFaceCount]

**bands**

List of color bands.

**Type** list of string

**clear()**

Clears tiled model data.

**copy()**

Create a copy of the tiled model.

**Returns** Copy of the tiled model.

**Return type** *TiledModel*

**crs**

Reference coordinate system.

**Type** *CoordinateSystem* or None

**data\_type**

Data type used to store color values.

**Type** *DataType*

**key**

Tiled model identifier.

**Type** int

**label**

Tiled model label.

**Type** string

**meta**

Tiled model meta data.

**Type** *MetaData*

**modified**

Modified flag.

Type bool

**pickPoint**(*origin, target, endpoints=1*)

Returns ray intersection with the tiled model.

**Parameters**

- **origin** (*Vector*) – Ray origin.
- **target** (*Vector*) – Point on the ray.
- **endpoints** (*int*) – Number of endpoints to check for (0 - line, 1 - ray, 2 - segment).

**Returns** Coordinates of the intersection point.

**Return type** *Vector*

**renderDepth**(*transform, calibration, resolution=1, cull\_faces=True, add\_alpha=True*)

Render tiled model depth image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** *Image*

**renderImage**(*transform, calibration, resolution=1, cull\_faces=True, add\_alpha=True, raster\_transform=RasterTransformNone*)

Render tiled model image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.
- **raster\_transform** (*RasterTransformType*) – Raster band transformation.

**Returns** Rendered image.

**Return type** *Image*

**renderMask**(*transform, calibration, resolution=1, cull\_faces=True*)

Render tiled model mask image for specified viewpoint.

**Parameters**

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

- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_faces** (*bool*) – Enable back-face culling.

**Returns** Rendered image.

**Return type** *Image*

**renderNormalMap**(*transform, calibration, resolution=1, cull\_faces=True, add\_alpha=True*)  
Render image with tiled model normals for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** *Image*

**renderPreview**(*width = 2048, height = 2048*[, *transform* ][, *progress* ])  
Generate tiled model preview image.

**Parameters**

- **width** (*int*) – Preview image width.
- **height** (*int*) – Preview image height.
- **transform** (*Matrix*) – 4x4 viewpoint transformation matrix.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**Returns** Preview image.

**Return type** *Image*

**transform**

4x4 tiled model transformation matrix.

**Type** *Matrix*

**class Metashape.TiledModelFormat**

Tiled model format in [TiledModelFormatNone, TiledModelFormatTLS, TiledModelFormatLOD, TiledModelFormatZIP, TiledModelFormatCesium, TiledModelFormatSLPK, TiledModelFormatOSGB, TiledModelFormatOSGT]

**class Metashape.Utils**

Utility functions.

**createChessboardImage**(*calib, cell\_size=150, max\_tilt=30*)  
Synthesizes photo of a chessboard.

**Parameters**

- **calib** (*Calibration*) – Camera calibration.
- **cell\_size** (*float*) – Chessboard cell size.
- **max\_tilt** (*float*) – Maximum camera tilt in degrees.

**Returns** Resulting image.

**Return type** *Image*

**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** (*bool*) – Enables white balance correction.

**Returns** Resulting mask.

**Return type** *Image*

**createMarkers**(*chunk, projections*)

Creates markers from a list of non coded projections.

**Parameters**

- **chunk** (*Chunk*) – Chunk to create markers in.
- **projections** (list of (*Camera, Target*) tuples) – List of marker projections.

**detectTargets**(*image, type=TargetCircular12bit, tolerance=50, inverted=False, noparity=False* [, *minimum\_size*] [, *minimum\_dist* ])

Detect targets on the image.

**Parameters**

- **image** (*Image*) – Image to process.
- **type** (*TargetType*) – Type of targets.
- **tolerance** (*int*) – Detector tolerance (0 - 100).
- **inverted** (*bool*) – Detect markers on black background.
- **noparity** (*bool*) – Disable parity checking.
- **minimum\_size** (*int*) – Minimum target radius in pixels to be detected (CrossTarget type only).
- **minimum\_dist** (*int*) – Minimum distance between targets in pixels (CrossTarget type only).

**Returns** List of detected targets.

**Return type** list of *Target*

**dmat2euler**(*R, dR, euler\_angles=EulerAnglesYPR*)

Calculate tangent euler rotation vector from tangent rotation matrix.

**Parameters**

- **R** (*Matrix*) – Rotation matrix.
- **dR** (*Matrix*) – Tangent rotation matrix.
- **euler\_angles** (*EulerAngles*) – Euler angles to use.

**Returns** Tangent rotation angles in degrees.

**Return type** *Vector*



**estimateImageQuality**(*image* [, *mask* ])

Estimate image sharpness.

**Parameters**

- **image** (*Image*) – Image to be analyzed.
- **mask** (*Image*) – Mask of the analyzed image region.

**Returns** Quality metric.

**Return type** float

**euler2mat**(*rotation*, *euler\_angles=EulerAnglesYPR*)

Calculate camera to world rotation matrix from euler rotation angles.

**Parameters**

- **rotation** (*Vector*) – Rotation vector.
- **euler\_angles** (*EulerAngles*) – Euler angles to use.

**Returns** Rotation matrix.

**Return type** *Matrix*

**mat2euler**(*R*, *euler\_angles=EulerAnglesYPR*)

Calculate euler rotation angles from camera to world rotation matrix.

**Parameters**

- **R** (*Matrix*) – Rotation matrix.
- **euler\_angles** (*EulerAngles*) – Euler angles to use.

**Returns** Rotation angles in degrees.

**Return type** *Vector*

**mat2opk**(*R*)

Calculate omega, phi, kappa from camera to world 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 camera to world 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** `Metashape.Vector`

n-component vector

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

**copy()**

Return a copy of the vector.

**Returns** A copy of the vector.

**Return type** *Vector*

**cross**(*a, b*)

Cross product of 2 vectors.

**Parameters**

- **a** (*Vector*) – First vector.
- **b** (*Vector*) – Second vector.

**Returns** Cross product.

**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** `Metashape.Version`  
Version object contains application version numbers.

**build**  
Build number.

**Type** int

**copy()**  
Return a copy of the object.

**Returns** A copy of the object.

**Return type** *Version*

**major**  
Major version number.

**Type** int

**micro**  
Micro version number.

**Type** int

**minor**  
Minor version number.

**Type** int

**class** `Metashape.Viewpoint`(*app*)  
Represents viewpoint in the model view

**center**  
Camera center.

**Type** *Vector*

**coo**  
Center of orbit.

**Type** *Vector*

**copy()**  
Return a copy of the object.

**Returns** A copy of the object.

**Return type** *Viewpoint*

**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

**class Metashape.Vignetting**

Vignetting polynomial

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *Vignetting*

## PYTHON API CHANGE LOG

### 3.1 Metashape version 1.8.0

- Added BuildPanorama and CalculatePointNormals classes
- Added ImageFormatJXL to ImageFormat enum
- Added Cylindrical to Sensor.Type enum
- Added Chunk.buildPanorama(), Chunk.calculatePointNormals() and Chunk.filterDenseCloud() methods
- Added findCamera(), findCameraGroup(), findCameraTrack(), findDenseCloud(), findDepthMaps(), findElevation(), findMarker(), findMarkerGroup(), findModel(), findOrthomosaic(), findScalebar(), findScalebarGroup(), findSensor() and findTiledModel() methods to Chunk class
- Added NetworkClient.serverStatus() method
- Added NetworkClient.setBatchPaused() and NetworkClient.setNodePaused() methods
- Added Settings.project\_absolute\_paths and Settings.project\_compression attributes
- Added CloseHoles.apply\_to\_selection attribute
- Added ConvertImages.merge\_planes attribute
- Added ExportPoints.screen\_space\_error and ExportTiledModel.screen\_space\_error attributes
- Added ExportReport.font\_size attribute
- Added ImportPoints.point\_neighbors attribute
- Added home\_point, interesting\_zone, powerlines, restricted\_zone and safety\_zone attributes to PlanMission class
- Added apply\_to\_selection argument to Model.closeHoles() method
- Added file\_format and max\_waypoints arguments to CameraTrack.save() method
- Added screen\_space\_error argument to Chunk.exportPoints() and Chunk.exportTiledModel() methods
- Added font\_size argument to Chunk.exportReport() method
- Added point\_neighbors argument to Chunk.importPoints() method
- Removed Shape.Type enum
- Removed ExportPanorama class
- Removed has\_z, type, vertex\_ids and vertices attributes from Shape class
- Removed pauseBatch(), resumeBatch(), pauseNode() and resumeNode() methods from NetworkClient class
- Removed PlanMission.max\_waypoints attribute

- Removed `SaveProject.absolute_paths` and `SaveProject.compression` attributes
- Removed `compression` and `absolute_paths` arguments from `Document.save()` method
- Changed default value of `BuildTiledModel.face_count` attribute to 20000
- Changed default value of `face_count` argument in `Chunk.buildTiledModel()` method to 20000

## 3.2 Metashape version 1.7.6

- Added `Cylindrical` to `Sensor.Type` enum

## 3.3 Metashape version 1.7.5

- Added `ClassifyGroundPoints.erosion_radius` attribute
- Added `erosion_radius` argument to `DenseCloud.classifyGroundPoints()` method

## 3.4 Metashape version 1.7.4

- Added `ServiceCesium` to `ServiceType` enum
- Added `ImportLaserScans` class
- Added `Chunk.colorizeDenseCloud()` and `Chunk.colorizeModel()` methods
- Added `Chunk.exportTexture()` and `Chunk.importLaserScans()` methods
- Added `breakpoints` and `rates` attributed to `GeneratePrescriptionMap` class
- Added `SmoothModel.preserve_edges` attribute
- Added `breakpoints` and `rates` arguments to `Chunk.generatePrescriptionMap()` method
- Added `preserve_edges` argument to `Chunk.smoothModel` method
- Renamed `ClusteringMethod` enum to `ClassificationMethod`
- Renamed `cluster_count`, `clustering_method` and `boundary` attributes in `GeneratePrescriptionMap` class
- Renamed `cluster_count`, `clustering_method` and `boundary` arguments in `Chunk.generatePrescriptionMap()` method
- Removed `ServiceSputnik` from `ServiceType` enum
- Removed `min_value`, `max_value` and `grid_azimuth` attributes from `GeneratePrescriptionMap` class
- Removed `min_value`, `max_value` and `grid_azimuth` arguments from `Chunk.generatePrescriptionMap()` method

### 3.5 Metashape version 1.7.3

- Added ModelFormatOSGT and ModelFormatLandXML to ModelFormat enum
- Added TiledModelFormatOSGT to TiledModelFormat enum
- Added CoordinateSystem.datumTransform() method
- Added DenseCloud.selectPointsByShapes() method
- Added Sensor.makeMaster() method
- Added Utils.dmat2euler() method
- Added Settings.lanuage attribute
- Added ShapeGroup.meta attribute
- Added Shapes.group attribute
- Added ExportPoints.compression attribute
- Added ExportTiledModel.model\_compression attribute
- Added ImportModel.decode\_udim attribute
- Added MatchPhotos.keypoint\_limit\_per\_mpx attribute
- Added compression argument to Chunk.exportPoints() method
- Added model\_compression argument to Chunk.exportTiledModel() method
- Added decode\_udim argument to Chunk.importModel() method
- Added keypoint\_limit\_per\_mpx argument to Chunk.matchPhotos() method
- Added uniform\_sampling argument to Chunk.samplePoints() method

### 3.6 Metashape version 1.7.2

- Added ClusteringMethod enum
- Added PointsFormatSLPK to PointsFormat enum
- Added DuplicateAsset and GeneratePrescriptionMap classes
- Added Chunk.generatePrescriptionMap() method
- Added merge, operand\_chunk, operand\_frame and operand\_asset attributes to BuildTiledModel class
- Added ExportReport.include\_system\_info attribute
- Added GenerateMasks.depth\_threshold attribute
- Added merge, operand\_chunk, operand\_frame and operand\_asset arguments to Chunk.buildTiledModel() method
- Added include\_system\_info argument to Chunk.exportReport() method
- Added depth\_threshold argument to Chunk.generateMasks() method

## 3.7 Metashape version 1.7.1

- Removed LegacyMapping from MappingMode enum
- Removed ReduceOverlap.sensor attribute
- Removed sensor argument from Chunk.reduceOverlap() method

## 3.8 Metashape version 1.7.0

- Added Geometry and AttachedGeometry classes
- Added FrameStep enum
- Added ServiceType enum
- Added Chunk.importVideo(), Chunk.publishData() and Chunk.samplePoints() methods
- Added Shape.geometry and Shape.is\_attached attributes
- Added alpha component to ShapeGroup.color attribute value
- Added ImportRaster.nodata\_value and ImportRaster.has\_nodata\_value attributes
- Added MatchPhotos.filter\_stationary\_points attribute
- Added BuildOrthomosaic.ghosting\_filter attribute
- Added attach\_viewpoints, group\_attached\_viewpoints and horizontal\_zigzags attributes to PlanMission class
- Added ReduceOverlap.sensor attribute
- Added dir argument to Application.getExistingDirectory(), getOpenFileName(), getOpenFileNames() and getSaveFileName() methods
- Added nodata\_value and has\_nodata\_value arguments to Chunk.importRaster() method
- Added filter\_stationary\_points argument to Chunk.matchPhotos() method
- Added ghosting\_filter argument to Chunk.buildOrthomosaic() method
- Added sensor argument to Chunk.reduceOverlap() method
- Renamed ImportMasks class to GenerateMasks
- Renamed MaskSource enum to MaskingMode
- Renamed Chunk.importMasks() method to Chunk.generateMasks()
- Removed ReduceOverlap.max\_cameras attribute
- Removed max\_cameras argument from Chunk.reduceOverlap() method



### 3.9 Metashape version 1.6.6

- Added Tasks.TransformRaster class
- Added ExportReference.precision attribute
- Added toNetworkTask() method to task classes
- Added Chunk.transformRaster() method
- Added precision argument to Chunk.exportReference() method

### 3.10 Metashape version 1.6.5

- Added Sensor.meta attribute

### 3.11 Metashape version 1.6.4

- Added Model.Vertex.confidence attribute
- Added ConvertImages.use\_initial\_calibration attribute
- Added image\_orientation, save\_invalid\_matches and use\_initial\_calibration attributes to ExportCameras class
- Added ExportModel.save\_confidence attribute
- Added crs and image\_orientation attributes to ImportCameras class
- Added CalibrationFormatPhotomod to CalibrationFormat enum
- Added save\_invalid\_matches, use\_initial\_calibration and image\_orientation arguments to Chunk.exportCameras() method
- Added save\_confidence argument to Chunk.exportModel() method
- Added crs and image\_orientation arguments to Chunk.importCameras() method
- Removed BuildUV.adaptive\_resolution attribute
- Removed adaptive\_resolution argument from Chunk.buildUV() method

### 3.12 Metashape version 1.6.3

- Added renderPreview() methods to DenseCloud, Model, Orthomosaic, PointCloud and TiledModel classes
- Added BuildUV.texture\_size attribute
- Added DecimateModel.apply\_to\_selection attribute
- Added DetectFiducials.cameras, DetectFiducials.frames and DetectFiducials.generate\_masks attributes
- Added ExportModel.embed\_texture attribute
- Added clip\_to\_boundary attribute to ExportPoints, ExportModel, ExportTiledModel and ExportRaster classes
- Added RasterFormatGeoPackage to RasterFormat enum
- Added ShapesFormatGeoPackage to ShapesFormat enum

- Added source argument to `Chunk.addSensor()` method
- Added `texture_size` argument to `Chunk.buildUV()` method
- Added `apply_to_selection` argument to `Chunk.decimateModel()` method
- Added `generate_masks`, `cameras` and `frames` arguments to `Chunk.detectFiducials()` method
- Added `embed_texture` argument to `Chunk.exportModel()` method
- Added `width`, `height`, `point_size` and `progress` arguments to `Chunk.renderPreview()` method
- Added `clip_to_boundary` argument to `Chunk.exportPoints()`, `Chunk.exportModel()`, `Chunk.exportTiledModel()` and `Chunk.exportRaster()` methods
- Added `meta` argument to `NetworkClient.createBatch()` method
- Removed `CalibrateLens.fit_p3` and `CalibrateLens.fit_p4` attributes

### 3.13 Metashape version 1.6.2

- Added `Application.ModelView` and `Application.OrthoView` classes
- Added `Application.removeMenuItem()` method
- Added `Model.transform()` method
- Added `PointCloud.cleanup()` method
- Added `Application.model_view` and `Application.ortho_view` attributes
- Added `BuildTexture.transfer_texture` attribute
- Added `PlanMission.min_pitch` and `PlanMission.max_pitch` attributes
- Added `columns`, `crs`, `delimiter`, `group_delimiters` and `skip_rows` attributes to `ImportShapes` class
- Added `CamerasFormatNVM` to `CamerasFormat` enum
- Added `PointsFormatPTX` to `PointsFormat` enum
- Added `ShapesFormatCSV` to `ShapesFormat` enum
- Added `transfer_texture` argument to `Chunk.buildTexture()` method
- Added `columns`, `crs`, `delimiter`, `group_delimiters` and `skip_rows` arguments to `Chunk.importShapes()` method
- Moved `ModelViewMode` enum to `ModelView` class
- Renamed `Application.console` attribute to `console_pane`
- Renamed `Application.captureModelView()` method to `ModelView.captureView()`
- Renamed `Application.captureOrthoView()` method to `OrthoView.captureView()`
- Renamed `Application.viewpoint` attribute to `ModelView.viewpoint`
- Removed `ReduceOverlap.capture_distance` attribute
- Removed `capture_distance` argument from `Chunk.reduceOverlap()` method
- Changed default values of `AlignCameras.reset_alignment` and `MatchPhotos.reset_matches` attributes to `False`
- Changed default value of `reset_alignment` argument in `Chunk.alignCameras()` method to `False`
- Changed default value of `reset_matches` argument in `Chunk.matchPhotos()` method to `False`

### 3.14 Metashape version 1.6.1

- Added `Application.releaseFreeMemory()` method
- Added `CoordinateSystem.towgs84` attribute
- Added `Marker.enabled` attribute
- Added `BuildModel.subdivide_task` attribute
- Added `subdivide_task` argument to `Chunk.buildModel()` method
- Changed default value of `keep_depth` argument in `Chunk.buildModel()` and `Chunk.buildTiledModel()` to `True`

### 3.15 Metashape version 1.6.0

- Added `BBox`, `ImageCompression`, `RPCModel` and `Model.Texture` classes
- Added `Tasks.ImportTiledModel` and `Task.ColorizeModel` classes
- Added `CalibrationFormat` and `ReferencePreselectionMode` enums
- Added `Model.addTexture()` and `Model.remove()` methods
- Added `Model.getActiveTexture()` and `Model.setActiveTexture()` methods
- Added `NetworkClient.setMasterServer()` method
- Added `setClassesFilter()`, `setConfidenceFilter()`, `setSelectionFilter()` and `resetFilters()` methods to `DenseCloud` class
- Added `renderDepth()`, `renderImage()`, `renderMask()` and `renderNormalMap()` methods to `PointCloud`, `DenseCloud` and `TiledModel` classes
- Added `Chunk.renderPreview()` method
- Added `Utils.euler2mat()` and `Utils.mat2euler()` methods
- Added `Calibration.rpc` attribute
- Added `Marker.position_covariance` attribute
- Added `Model.textures` attribute
- Added `TiledModel.crs` and `TiledModel.transform` attributes
- Added `EulerAnglesPOK` and `EulerAnglesANK` values to `EulerAngles` enum
- Added `PointsFormatPCD` to `PointsFormat` enum
- Added `ShapesFormatGeoJSON` to `ShapesFormat` enum
- Added `RPC` to `Sensor.Type` enum
- Added `image_compression` attribute to `ExportOrthophotos`, `ExportRaster`, `ExportTiledModel` and `UndistortPhotos` classes
- Added `AddPhotos.load_rpc_txt` attribute
- Added `AlignCameras.min_image` attribute
- Added `BuildDenseCloud.point_confidence` attribute
- Added `BuildModel.vertex_confidence`, `BuildModel.max_workgroup_size` and `BuildModel.workitem_size_cameras` attributes

- Added `BuildTexture.source_model` and `BuildTexture.texture_type` attributes
- Added `BuildUV.adaptive_resolution` attribute
- Added `DecimateModel.asset` attribute
- Added `ExportPanorama.image_compression` attribute
- Added `ExportPoints.save_classes` and `ExportPoints.save_confidence` attributes
- Added `ExportTexture.texture_type` attribute
- Added `ExportTiledModel.crs` attribute
- Added `ImportCameras.image_list` and `ImportCameras.load_image_list` attributes
- Added `ImportPoints.calculate_normals` attribute
- Added `MatchPhotos.guided_matching` and `MatchPhotos.reference_preselection_mode` attributes
- Added `MergeChunks.merge_depth_maps`, `MergeChunks.merge_elevations` and `MergeChunks.merge_orthomosaics` attributes
- Added `OptimizeCameras.fit_corrections` attribute
- Added `TriangulatePoints.max_error` and `TriangulatePoints.min_image` attributes
- Added `endpoints` argument to `PointCloud.pickPoint()`, `DenseCloud.pickPoint()`, `Model.pickPoint()` and `TiledModel.pickPoint()` methods
- Added `compression` argument to `Image.save()` method
- Added `cull_faces` and `add_alpha` arguments to `Model.renderDepth()` method
- Added `cull_faces`, `add_alpha` and `raster_transform` arguments to `Model.renderImage()` method
- Added `cull_faces` argument to `Model.renderMask()` method
- Added `cull_faces` and `add_alpha` arguments to `Model.renderNormalMap()` method
- Moved `TiffCompression` enum to `ImageCompression` class
- Renamed `Tasks.UndistortPhotos` class to `Tasks.ConvertImages`
- Renamed `Chunk.estimateImageQuality()` method to `Chunk.analyzePhotos()`
- Renamed `Chunk.buildPoints()` method to `Chunk.triangulatePoints()`
- Renamed `Chunk.loadReference()` method to `Chunk.importReference()`
- Renamed `Chunk.saveReference()` method to `Chunk.exportReference()`
- Renamed `Chunk.refineModel()` method to `Chunk.refineMesh()`
- Renamed `network_distribute` tasks attribute to `subdivide_task`
- Renamed `AlignChunks.align_method` attribute to `method`
- Renamed `AlignChunks.match_downscale` attribute to `downscale`
- Renamed `AlignChunks.match_filter_mask` attribute to `filter_mask`
- Renamed `AlignChunks.match_mask_tiepoints` attribute to `mask_tiepoints`
- Renamed `AlignChunks.match_point_limit` attribute to `keypoint_limit`
- Renamed `AlignChunks.match_select_pairs` attribute to `generic_preselection`
- Renamed `BuildDenseCloud.store_depth` attribute to `keep_depth`
- Renamed `BuildModel.store_depth` attribute to `keep_depth`

- Renamed BuildOrthomosaic.ortho\_surface attribute to surface\_data
- Renamed BuildTiledModel.store\_depth attribute to keep\_depth
- Renamed BuildUV.texture\_count attribute to page\_count
- Renamed CalibrateColors.data\_source attribute to source\_data
- Renamed CalibrateColors.calibrate\_color\_balance attribute to white\_balance
- Renamed ClassifyGroundPoints.cls\_from attribute to source\_class
- Renamed ClassifyPoints.cls\_from attribute to source\_class
- Renamed ClassifyPoints.cls\_to attribute to target\_classes
- Renamed DecimateModel.target\_face\_count attribute to face\_count
- Renamed DuplicateChunk.copy\_dense\_cloud attribute to copy\_dense\_clouds
- Renamed ClassifyPoints.copy\_elevation attribute to copy\_elevations
- Renamed ClassifyPoints.copy\_model attribute to copy\_models
- Renamed ClassifyPoints.copy\_orthomosaic attribute to copy\_orthomosaics
- Renamed ClassifyPoints.copy\_tiled\_model attribute to copy\_tiled\_models
- Renamed ExportCameras.bingo\_export\_geoin attribute to bingo\_save\_geoin
- Renamed ExportCameras.bingo\_export\_gps attribute to bingo\_save\_gps
- Renamed ExportCameras.bingo\_export\_image attribute to bingo\_save\_image
- Renamed ExportCameras.bingo\_export\_itera attribute to bingo\_save\_itera
- Renamed ExportCameras.bundler\_export\_list attribute to bundler\_save\_list
- Renamed ExportCameras.chan\_order\_rotate attribute to chan\_rotation\_order
- Renamed ExportCameras.coordinates attribute to crs
- Renamed ExportCameras.export\_markers attribute to save\_markers
- Renamed ExportCameras.export\_points attribute to save\_points
- Renamed ExportMarkers.coordinates attribute to crs
- Renamed ExportModel.coordinates attribute to crs
- Renamed ExportModel.export\_alpha attribute to save\_alpha
- Renamed ExportModel.export\_cameras attribute to save\_cameras
- Renamed ExportModel.export\_colors attribute to save\_colors
- Renamed ExportModel.export\_comment attribute to save\_comment
- Renamed ExportModel.export\_markers attribute to save\_markers
- Renamed ExportModel.export\_normals attribute to save\_normals
- Renamed ExportModel.export\_texture attribute to save\_texture
- Renamed ExportModel.export\_udim attribute to save\_udim
- Renamed ExportModel.export\_uv attribute to save\_uv
- Renamed ExportOrthophotos.write\_alpha attribute to save\_alpha
- Renamed ExportOrthophotos.write\_kml attribute to save\_kml

- Renamed `ExportOrthophotos.write_world` attribute to `save_world`
- Renamed `ExportPoints.coordinates` attribute to `crs`
- Renamed `ExportPoints.data_source` attribute to `source_data`
- Renamed `ExportPoints.export_colors` attribute to `save_colors`
- Renamed `ExportPoints.export_comment` attribute to `save_comment`
- Renamed `ExportPoints.export_images` attribute to `save_images`
- Renamed `ExportPoints.export_normals` attribute to `save_normals`
- Renamed `ExportPoints.tile_height` attribute to `block_height`
- Renamed `ExportPoints.tile_width` attribute to `block_width`
- Renamed `ExportPoints.write_tiles` attribute to `split_in_blocks`
- Renamed `ExportRaster.data_source` attribute to `source_data`
- Renamed `ExportRaster.kmz_section_enable` attribute to `network_links`
- Renamed `ExportRaster.tile_width` attribute to `block_width`
- Renamed `ExportRaster.tile_height` attribute to `block_height`
- Renamed `ExportRaster.write_alpha` attribute to `save_alpha`
- Renamed `ExportRaster.write_kml` attribute to `save_kml`
- Renamed `ExportRaster.write_scheme` attribute to `save_scheme`
- Renamed `ExportRaster.write_tiles` attribute to `split_in_blocks`
- Renamed `ExportRaster.write_world` attribute to `save_world`
- Renamed `ExportRaster.xyz_level_min` attribute to `min_zoom_level`
- Renamed `ExportRaster.xyz_level_max` attribute to `max_zoom_level`
- Renamed `ExportShapes.coordinates` attribute to `crs`
- Renamed `ExportShapes.export_attributes` attribute to `save_attributes`
- Renamed `ExportShapes.export_labels` attribute to `save_labels`
- Renamed `ExportShapes.export_points` attribute to `save_points`
- Renamed `ExportShapes.export_polygons` attribute to `save_polygons`
- Renamed `ExportShapes.export_polylines` attribute to `save_polylines`
- Renamed `ExportTexture.write_alpha` attribute to `save_alpha`
- Renamed `ExportTiledModel.mesh_format` attribute to `model_format`
- Renamed `ImportMasks.method` attribute to `source`
- Renamed `ImportModel.coordinates` attribute to `crs`
- Renamed `ImportPoints.coordinates` attribute to `crs`
- Renamed `ImportReference.coordinates` attribute to `crs`
- Renamed `MatchPhotos.preselection_generic` attribute to `generic_preselection`
- Renamed `MatchPhotos.preselection_reference` attribute to `reference_preselection`
- Renamed `MatchPhotos.store_keypoints` attribute to `keep_keypoints`

- Renamed RefineMesh.iterations attribute to iterations
- Renamed SmoothModel.apply\_to\_selected attribute to apply\_to\_selection
- Renamed TrackMarkers.frame\_start attribute to first\_frame
- Renamed TrackMarkers.frame\_end attribute to last\_frame
- Renamed processing methods arguments to match task parameters names (e.g. dx/dy -> resolution\_x/resolution\_y, write\_xxx -> save\_xxx, export\_xxx -> save\_xxx, import\_xxx -> load\_xxx, preselection\_generic -> generic\_preselection, preselection\_reference -> reference\_preselection, source\_data -> data\_source, etc.)
- Replaced Chunk.importDem() method with Chunk.importRaster() method
- Replaced Chunk.exportDem() and Chunk.exportOrthomosaic() methods with Chunk.exportRaster() method
- Removed Accuracy and Quality enums
- Removed Model.texture() and Model.setTexture() methods
- Removed ExportPoints.precision attribute
- Removed OptimizeCameras.fit\_p3 and OptimizeCameras.fit\_p4 attributes
- Removed PlanMission.max\_cameras and PlanMission.use\_cameras attributes
- Removed tiff\_big, tiff\_tiled and tiff\_overviews attributes from ExportOrthophotos and ExportRaster classes
- Removed tiff\_compression attribute from ExportOrthophotos, ExportRaster and UndistortPhotos classes
- Removed jpeg\_quality attribute from ExportOrthophotos, ExportRaster, ExportTiledModel and UndistortPhotos classes

## 3.16 Metashape version 1.5.5

No Python API changes

## 3.17 Metashape version 1.5.4

- Added Tasks.FilterDenseCloud class
- Added TiledModel.FaceCount enum
- Added copy() method to Antenna, Calibration, ChunkTransform, CirTransform, CoordinateSystem, Document, MetaData, OrthoProjection, RasterTransform, Region, Shutter, Target, Version, Viewpoint and Vignetting classes
- Added CameraTrack.save() and CameraTrack.load() methods
- Added Chunk.reduceOverlap() method
- Added location\_enabled and rotation\_enabled attributes to Sensor.Reference class
- Added CameraTrack.chunk and CameraTrack.meta attributes
- Added BuildTiledModel.ghosting\_filter and BuildTiledModel.transfer\_texture attributes
- Added ExportPoints.network\_distribute and ExportPoints.region attributes
- Added ExportTiledModel.jpeg\_quality and ExportTiledModel.texture\_format attributes
- Added prevent\_intersections argument to Chunk.buildContours() method

- Added `transfer_texture` argument to `Chunk.buildTiledModel()` method
- Added `region` argument to `Chunk.exportPoints()` method
- Added `texture_format` and `jpeg_quality` arguments to `Chunk.exportTiledModel()` method
- Added `progress` argument to `Chunk.importMarkers()` method
- Added `ImageFormatWebP` to `ImageFormat` enum

### 3.18 Metashape version 1.5.3

- Added `DepthMap.getCalibration()` and `DepthMap.setCalibration()` methods
- Added `NetworkClient.dumpBatches()`, `NetworkClient.loadBatches()` and `NetworkClient.setBatchNodeLimit()` methods
- Added `location_enabled` and `rotation_enabled` attributes to `Camera.Reference` class
- Added `keep_depth` argument to `Chunk.buildTiledModel()` method
- Added `uv` argument to `Chunk.exportModel()` method
- Added `level` argument to `DepthMap.image()` and `DepthMap.setImage()` methods
- Changed default value of `keep_depth` argument in `Chunk.buildDenseCloud()` and `Chunk.buildModel()` methods to `True`
- Changed default value of `max_neighbors` argument in `Chunk.buildDenseCloud()` method to 100

### 3.19 Metashape version 1.5.2

- Added `CameraTrack` class
- Added `Tasks.PlanMission` and `Tasks.ReduceOverlap` classes
- Added `Camera.Type` enum
- Added `Chunk.addCameraTrack()` method
- Added `Application.title` attribute
- Added `Camera.type` attribute
- Added `Chunk.camera_track` and `Chunk.camera_tracks` attributes
- Added `BuildModel.trimming_radius` attribute
- Added `DetectMarkers.filter_mask` attribute
- Added `ImportReference.shutter_lag` attribute
- Added `Bundler` and `BINGO` specific attributes to `ExportCameras` class
- Added `supports_gpu` attribute to task classes
- Added `x`, `y`, `w`, `h` arguments to `Image.open()` method
- Added `filter_mask` argument to `Chunk.detectMarkers()` method
- Added `image_list` argument to `Chunk.importCameras()` method
- Added `shutter_lag` argument to `Chunk.loadReference()` method



- Added ImageFormatBIL, ImageFormatXYZ, ImageFormatDDS to ImageFormat enum
- Removed Tasks.PlanMotion class
- Removed Animation class
- Removed Chunk.animation attribute
- Removed smoothness attribute from Tasks.BuildModel and Tasks.BuildTiledModel classes
- Removed quality and reuse\_depth arguments from Chunk.buildModel() method
- Removed downscale, filter\_mode, max\_neighbors, max\_workgroup\_size, network\_distribute, reuse\_depth, workitem\_size\_cameras from Tasks.BuildModel class

## 3.20 Metashape version 1.5.1

- Added License class
- Added Tasks.MergeAssets class
- Added Metashape.license attribute
- Renamed Tasks.OptimizeCoverage class to Tasks.PlanMotion

## 3.21 Metashape version 1.5.0

- Added Sensor.Reference class
- Added Tasks.ClassifyPoints and Tasks.OptimizeCoverage classes
- Added DataType enum
- Added Model.TextureType enum
- Added Tasks.TargetType enum
- Added Animation.Track.resize() method
- Added Chunk.findFrame() method
- Added DenseCloud.classifyPoints() method
- Added Document.findChunk() method
- Added Model.Faces.resize(), Model.Vertices.resize() and Model.TexVertices.resize() methods
- Added Tasks.createTask() method
- Added decode(), decodeJSON(), encodeJSON() methods to task classes
- Added Antenna.location\_covariance and Antenna.rotation\_covariance attributes
- Added Camera.calibration, Camera.location\_covariance and Camera.rotation\_covariance attributes
- Added Chunk.image\_contrast attribute
- Added DenseCloud.bands and DenseCloud.data\_type attributes
- Added Model.bands and Model.data\_type attributes
- Added Elevation.palette attribute
- Added Model.Face.tex\_index attribute

- Added Orthomosaic.bands and Orthomosaic.data\_type attributes
- Added PointCloud.Point.cov attribute
- Added PointCloud.bands and PointCloud.data\_type attributes
- Added Sensor.data\_type, Sensor.film\_camera, Sensor.location\_covariance, Sensor.reference and Sensor.rotation\_covariance attributes
- Added Sensor.fixed\_params and Sensor.photo\_params attributes
- Added TiledModel.bands and TiledModel.data\_type attributes
- Added AlignChunks.markers and AlignChunks.match\_mask\_tiepoints attributes
- Added BuildOrthomosaic.refine\_seamlines attribute
- Added DetectMarkers.cameras and DetectMarkers.maximum\_residual attributes
- Added ExportModel.colors\_rgb\_8bit and ExportPoints.colors\_rgb\_8bit attributes
- Added ExportOrthophotos.tiff\_tiled and ExportRaster.tiff\_tiled attributes
- Added OptimizeCameras.tiepoint\_covariance attribute
- Added BuildModel.smoothness and BuildTiledModel.smoothness attributes
- Added target and workitem\_count attributes to task classes
- Added max\_workgroup\_size and workitem\_size\_tiles attributes to Tasks.BuildDem class
- Added max\_workgroup\_size and workitem\_size\_cameras attributes to Tasks.BuildDenseCloud class
- Added max\_workgroup\_size and workitem\_size\_cameras attributes to Tasks.BuildDepthMaps class
- Added max\_workgroup\_size and workitem\_size\_cameras attributes to Tasks.BuildModel class
- Added max\_workgroup\_size, workitem\_size\_cameras and workitem\_size\_tiles attributes to Tasks.BuildOrthomosaic class
- Added max\_workgroup\_size, workitem\_size\_cameras and face\_count attributes to Tasks.BuildTiledModel class
- Added max\_workgroup\_size, workitem\_size\_cameras and workitem\_size\_pairs attributes to Tasks.MatchPhotos class
- Added refine\_seamlines argument to Chunk.buildOrthomosaic() method
- Added face\_count argument to Chunk.buildTiledModel() method
- Added keypoints argument to Chunk.copy() method
- Added maximum\_residual and cameras arguments to Chunk.detectMarkers() method
- Added tiff\_tiled argument to Chunk.exportDem(), Chunk.exportOrthomosaic() and Chunk.exportOrthophotos() methods
- Added colors\_rgb\_8bit argument to Chunk.exportModel() and Chunk.exportPoints() methods
- Added tiepoint\_covariance argument to Chunk.optimizeCameras() method
- Added confidence argument to DenseCloud.classifyPoints() method
- Added mask\_tiepoints and markers arguments to Document.alignChunks() method
- Added ignore\_lock argument to Document.open() method
- Added type argument to Model.setTexture() and Model.texture() methods
- Added workitem argument to Task.apply() method

- Added ModelFormatGLTF and ModelFormatX3D to ModelFormat enum
- Added Car and Manmade to PointClass enum
- Changed default value of filter argument in Chunk.buildDepthMaps() to MildFiltering
- Removed Tasks.BuildModel.visibility\_mesh attribute

## 3.22 PhotoScan version 1.4.4

- Added AddPhotos.strip\_extensions attribute
- Added ExportRaster.image\_description attribute
- Added ExportShapes.export\_attributes, ExportShapes.export\_labels and ExportShapes.polygons\_as\_polylines attributes
- Added image\_description argument to Chunk.exportDem() and Chunk.exportOrthomosaic() methods
- Added format, polygons\_as\_polylines, export\_labels and export\_attributes arguments to Chunk.exportShapes() method
- Added format argument to Chunk.importShapes() method
- Added RasterFormatTMS to RasterFormat enum

## 3.23 PhotoScan version 1.4.3

- Added Version class
- Added Tasks.DetectFiducials class
- Added Chunk.detectFiducials() method
- Added Sensor.calibrateFiducials() method
- Added CoordinateSystem.addGeoid() method
- Added PhotoScan.version attribute
- Added Sensor.normalize\_to\_float attribute
- Added minimum\_dist attribute to Tasks.DetectMarkers class
- Added minimum\_dist argument to Chunk.detectMarkers() and Utils.detectTargets() methods
- Added keypoints argument to PointCloud.copy() method
- Changed default value of adaptive\_fitting argument in Chunk.alignCameras() to False

## 3.24 PhotoScan version 1.4.2

- Added `Tasks.ColorizeDenseCloud` class
- Added `PointCloud.removeKeypoints()` method
- Added `CoordinateSystem.transformationMatrix()` method
- Added `Vector.cross()` method
- Added `Shapes.updateAltitudes()` method
- Added `log_enable`, `log_path`, `network_enable`, `network_host`, `network_path` and `network_port` attributes to `Application.Settings` class
- Added `covariance_matrix` and `covariance_params` attributes to `Calibration` class
- Added `flip_x`, `flip_y`, `flip_z` attributes to `Tasks.BuildDem` and `Tasks.BuildOrthomosaic` classes
- Added `max_neighbors` attribute to `Tasks.BuildDenseCloud`, `Tasks.BuildDepthMaps` and `Tasks.BuildModel` classes
- Added `jpeg_quality`, `tiff_compression` and `update_gps_tags` attributes to `Tasks.UndistortPhotos` class
- Added `copy_keypoints` attribute to `Tasks.DuplicateChunk` class
- Added `width`, `height` and `world_transform` attributes to `Tasks.ExportRaster` class
- Added `store_depth` attribute to `Tasks.BuildTiledModel` class
- Added `DenseCloud.crs` and `DenseCloud.transform` attributes
- Added `CoordinateSystem.wkt2` attribute
- Added `keep_keypoints` argument to `Chunk.matchPhotos()` method
- Added `flip_x`, `flip_y`, `flip_z` arguments to `Chunk.buildDem()` and `Chunk.buildOrthomosaic()` methods
- Added `max_neighbors` argument to `Chunk.buildDenseCloud()` and `Chunk.buildDepthMaps()` methods
- Added `cull_faces` argument to `Chunk.buildOrthomosaic()` method
- Added `reuse_depth` and `ghosting_filter` arguments to `Chunk.buildTiledModel()` method
- Added `use_reflectance_panels` and `use_sun_sensor` arguments to `Chunk.calibrateReflectance()` method
- Added `width`, `height` and `world_transform` arguments to `Chunk.exportDem()` and `Chunk.exportOrthomosaic()` methods
- Added `filter_mask` argument to `Chunk.estimateImageQuality()` method
- Added `revision` argument to `NetworkClient.nodeList()` method
- Added `ImagesData` to `DataSource` enum
- Added `ModelFormatOSGB` to `ModelFormat` enum
- Added `TiledModelFormatOSGB` to `TiledModelFormat` enum

## 3.25 PhotoScan version 1.4.1

- Added OrthoProjection.Type enum
- Added Camera.image() method
- Added Chunk.loadReflectancePanelCalibration() method
- Added PointCloud.Points.copy() and PointCloud.Points.resize() methods
- Added PointCloud.Projections.resize() method
- Added PointCloud.Tracks.copy() and PointCloud.Tracks.resize() methods
- Added OrthoProjection.matrix, OrthoProjection.radius and OrthoProjection.type attributes
- Added Tasks.AnalyzePhotos.filter\_mask attribute
- Added Tasks.CalibrateReflectance.use\_reflectance\_panels and Tasks.CalibrateReflectance.use\_sun\_sensor attributes
- Added Tasks.MatchPhotos.mask\_tiepoints attribute
- Added Tasks.OptimizeCameras.adaptive\_fitting attribute
- Added strip\_extensions argument to Chunk.addPhotos() method
- Added keep\_depth argument to Chunk.buildDenseCloud() method
- Added adaptive\_resolution argument to Chunk.buildUV() method
- Added alpha argument to Chunk.exportModel() method
- Added mask\_tiepoints argument to Chunk.matchPhotos() method
- Added adaptive\_fitting argument to Chunk.optimizeCameras() method
- Added mask argument to Utils.estimateImageQuality() method
- Added CamerasFormatABC and CamerasFormatFBX to CamerasFormat enum
- Added ImageFormatJP2 to ImageFormat enum
- Added LegacyMapping to MappingMode enum

## 3.26 PhotoScan version 1.4.0

- Added Tasks classes
- Added Animation, OrthoProjection, Target and Vignetting classes
- Added ShapesFormat enum
- Added Marker.Type enum
- Added Chunk.calibrateColors(), Chunk.calibrateReflectance() and Chunk.locateReflectancePanels() methods
- Added Chunk.buildDepthMaps(), Chunk.importPoints(), Chunk.refineModel() and Chunk.removeLighting() methods
- Added Chunk.addDenseCloud(), Chunk.addDepthMaps(), Chunk.addElevation(), Chunk.addModel(), Chunk.addOrthomosaic() and Chunk.addTiledModel() methods
- Added Chunk.sortCameras(), Chunk.sortMarkers() and Chunk.sortScalebars() methods
- Added DenseCloud.clear() method

- Added `DepthMaps.clear()` and `DepthMaps.copy()` methods
- Added `Elevation.clear()` and `Elevation.copy()` methods
- Added `Model.clear()` method
- Added `Orthomosaic.clear()` and `Orthomosaic.copy()` methods
- Added `TiledModel.clear()` and `TiledModel.copy()` methods
- Added `Image.gaussianBlur()` and `Image.uniformNoise()` methods
- Added `NetworkTask.encode()` method
- Added `Utils.createChessboardImage()` and `Utils.detectTargets()` methods
- Added `Camera.Reference.location_accuracy` and `Camera.Reference.rotation_accuracy` attributes
- Added `Camera.layer_index`, `Camera.master` and `Camera.vignetting` attributes
- Added `Chunk.dense_clouds`, `Chunk.depth_maps_sets`, `Chunk.elevations`, `Chunk.models`, `Chunk.orthomosaics` and `Chunk.tiled_models` attributes
- Added `Chunk.animation`, `Chunk.camera_crs`, `Chunk.marker_crs` and `Chunk.world_crs` attributes
- Added `CoordinateSystem.geoccs` and `CoordinateSystem.geoid_height` attributes
- Added `Marker.Projection.valid` attribute
- Added `Sensor.black_level`, `Sensor.fiducials`, `Sensor.fixed_calibration`, `Sensor.fixed_location`, `Sensor.fixed_rotation`, `Sensor.layer_index`, `Sensor.location`, `Sensor.master`, `Sensor.normalize_sensitivity`, `Sensor.rolling_shutter`, `Sensor.rotation`, `Sensor.sensitivity` and `Sensor.vignetting` attributes
- Added `Camera.chunk`, `Marker.chunk`, `Scalebar.chunk` and `Sensor.chunk` attributes
- Added `Marker.sensor` and `Marker.type` attributes
- Added `Elevation.projection`, `Orthomosaic.projection` and `Shapes.projection` attributes
- Added `DenseCloud.key` and `DenseCloud.label` attributes
- Added `DepthMaps.key` and `DepthMaps.label` attributes
- Added `Elevation.key` and `Elevation.label` attributes
- Added `Model.key` and `Model.label` attributes
- Added `Orthomosaic.key` and `Orthomosaic.label` attributes
- Added `TiledModel.key` and `TiledModel.label` attributes
- Added `point_colors` argument to `Chunk.buildDenseCloud()` method
- Added `ghosting_filter` argument to `Chunk.buildTexture()` method
- Added `minimum_size` argument to `Chunk.detectMarkers()` method
- Added `raster_transform` argument to `Chunk.exportModel()`, `Chunk.exportPoints()`, `Chunk.exportTiledModel()` methods
- Added `tiff_overviews` argument to `Chunk.exportDem()`, `Chunk.exportOrthomosaic()` and `Chunk.exportOrthophotos()` methods
- Added `min_zoom_level` and `max_zoom_level` arguments to `Chunk.exportDem()` and `Chunk.exportOrthomosaic()` methods
- Added `cameras` argument to `Chunk.exportOrthophotos()` method
- Added `image_format` argument to `Chunk.exportPoints()` method

- Added `page_numbers` argument to `Chunk.exportReport()` method
- Added `items`, `crs`, `ignore_labels`, `threshold` and `progress` arguments to `Chunk.loadReference()` method
- Added `create_markers` argument to `Chunk.loadReference()` method
- Added `progress` argument to `Chunk.saveReference()` method
- Added `quality`, `volumetric_masks`, `keep_depth` and `reuse_depth` arguments to `Chunk.buildModel()` method
- Added `selected_faces` and `fix_borders` arguments to `Chunk.smoothModel()` method
- Added `export_points`, `export_markers`, `use_labels` and `progress` arguments to `Chunk.exportCameras()` method
- Added `channels` and `datatype` arguments to `Photo.image()` method
- Added `CamerasFormatBlocksExchange` and `CamerasFormatORIMA` to `CamerasFormat` enum
- Added `ImageFormatNone` to `ImageFormat` enum
- Added `UndefinedLayout` to `ImageLayout` enum
- Added `ModelFormatNone` and `ModelFormatABC` to `ModelFormat` enum
- Added `PointsFormatNone` and `PointsFormatCesium` to `PointsFormat` enum
- Added `RasterFormatNone` to `RasterFormat` enum
- Added `ReferenceFormatNone` and `ReferenceFormatAPM` to `ReferenceFormat` enum
- Added `TiledModelFormatNone`, `TiledModelFormatCesium` and `TiledModelFormatSLPK` to `TiledModelFormat` enum
- Renamed `Chunk.master_channel` attribute to `Chunk.primary_channel`
- Removed `MatchesFormat` enum
- Removed `Chunk.exportMatches()` method
- Removed `Camera.Reference.accuracy_ypr` attribute
- Removed `quality`, `filter`, `cameras`, `keep_depth`, `reuse_depth` arguments from `Chunk.buildDenseCloud()` method
- Removed `color_correction` argument from `Chunk.buildOrthomosaic()` and `Chunk.buildTexture()` methods
- Removed `fit_shutter` argument from `Chunk.optimizeCameras()` method

## 3.27 PhotoScan version 1.3.5

No Python API changes

## 3.28 PhotoScan version 1.3.4

No Python API changes

## 3.29 PhotoScan version 1.3.3

- Added `network_links` argument to `Chunk.exportDem()` and `Chunk.exportOrthomosaic()` methods
- Added `read_only` argument to `Document.open()` method
- Added `NetworkClient.setNodeCPUEnable()` and `NetworkClient.setNodeGPUMask()` methods
- Added `Chunk.modified`, `DenseCloud.modified`, `DepthMaps.modified`, `Document.modified`, `Elevation.modified`, `Masks.modified`, `Model.modified`, `Orthomosaic.modified`, `PointCloud.modified`, `Shapes.modified`, `Thumbnails.modified`, `TiledModel.modified` attributes
- Added `Document.read_only` attribute
- Added `CamerasFormatSummit` to `CamerasFormat` enum

## 3.30 PhotoScan version 1.3.2

- Added `vertex_colors` argument to `Chunk.buildModel()` method
- Added `Shape.vertex_ids` attribute

## 3.31 PhotoScan version 1.3.1

- Added `Settings` and `TiledModel` classes
- Added `Application.getBool()` method
- Added `Camera.unproject()` method
- Added `Chunk.addFrames()`, `Chunk.addMarkerGroup()`, `Chunk.addScalebarGroup()` and `Chunk.buildSeamlines()` methods
- Added `DenseCloud.pickPoint()` and `DenseCloud.updateStatistics()` methods
- Added `Elevation.altitude()` method
- Added `Matrix.svd()` method
- Added `Model.pickPoint()` method
- Added `Orthomosaic.reset()` and `Orthomosaic.update()` methods
- Added `PointCloud.pickPoint()` method
- Added `filter` argument to `Application.getOpenFileName()`, `Application.getOpenFileNames()` and `Application.getSaveFileName()` methods
- Added `point` and `visibility` arguments to `Chunk.addMarker()` method
- Added `raster_transform` and `write_scheme` arguments to `Chunk.exportDem()` method
- Added `write_scheme` and `white_background` arguments to `Chunk.exportOrthomosaic()` method
- Added `white_background` argument to `Chunk.exportOrthophotos()` method
- Added `projection` argument to `Chunk.exportMarkers()` method
- Added `markers` argument to `Chunk.exportModel()` method
- Added `pairs` argument to `Chunk.matchPhotos()` method



- Added columns and delimiter arguments to `Chunk.saveReference()` method
- Added version argument to `Document.save()` method
- Renamed `npasses` argument in `Chunk.smoothModel()` method to `strength` and changed its type to `float`
- Renamed `from` and `to` arguments in `CoordinateSystem.transform()`, `DenseCloud.assignClass()`, `DenseCloud.assignClassToSelection()` and `DenseCloud.classifyGroundPoints()` methods to avoid collision with reserved words
- Added `Application.settings` attribute
- Added `Chunk.tiled_model` attribute
- Added `ShapeGroup.color` and `ShapeGroup.show_labels` attributes
- Added `ImageFormatTGA` to `ImageFormat` enum

### 3.32 PhotoScan version 1.3.0

- Added `MarkerGroup`, `Masks`, `ScalebarGroup`, `Shutter` and `Thumbnails` classes
- Added `Application.PhotosPane` class
- Added `Model.Statistics` class
- Added `Orthomosaic.Patch` and `Orthomosaic.Patches` classes
- Added `PointCloud.Filter` class
- Added `CamerasFormat`, `EulerAngles`, `ImageFormat`, `ImageLayout`, `MaskOperation`, `MaskSource`, `MatchesFormat`, `ModelFormat`, `ModelViewMode`, `PointClass`, `PointsFormat`, `RasterFormat`, `ReferenceFormat`, `ReferenceItems`, `RotationOrder`, `TiffCompression`, `TiledModelFormat` enums
- Added `Application.captureOrthoView()` method
- Added `Chunk.refineMarkers()` method
- Added `CoordinateSystem.listBuiltinCRS()` class method
- Added `Matrix.translation()` method
- Added `Model.statistics()` method
- Added `NetworkClient.serverInfo()`, `NetworkClient.nodeStatus()`, `NetworkClient.setNodeCapability()` and `NetworkClient.quitNode()` methods
- Added `Photo.imageMeta()` method
- Added `Shape.area()`, `Shape.perimeter2D()`, `Shape.perimeter3D()` and `Shape.volume()` methods
- Added `Utils.createMarkers()` method
- Added `source` argument to `Application.captureModelView()` method
- Added `image_format` argument to `Chunk.exportDem()` method
- Added `write_alpha` argument to `Chunk.exportOrthophotos()` method
- Added `image_format` and `write_alpha` arguments to `Chunk.exportOrthomosaic()` method
- Added `groups`, `projection`, `shift` and `progress` arguments to `Chunk.exportShapes()` method
- Added `items` and `progress` arguments to `Chunk.copy()` method
- Added `sensor` argument to `Chunk.addCamera()` method

- Added layout argument to `Chunk.addPhotos()` method
- Added `jpeg_quality` argument to `Chunk.exportOrthomosaic()` and `Chunk.exportOrthophotos()` methods
- Added `fill_holes` argument to `Chunk.buildOrthomosaic()` method
- Added `fit_shutter` argument to `Chunk.optimizeCameras()` method
- Added `settings` argument to `Chunk.exportReport()` method
- Added `progress` argument to various `DenseCloud` methods
- Added `from` argument to `DenseCloud.classifyGroundPoints()` method
- Added `chunks` and `progress` arguments to `Document.append()` method
- Added `progress` argument to `Document.alignChunks()` and `Document.mergeChunks()` methods
- Added `revision` argument to `NetworkClient.batchList()`, `NetworkClient.batchStatus()` methods
- Added `Application.photos_pane` attribute
- Added `Camera.shutter` attribute
- Added `Chunk.masks` and `Chunk.thumbnails` attributes
- Added `Chunk.marker_groups` and `Chunk.scalebar_groups` attributes
- Added `Chunk.euler_angles` and `Chunk.scalebar_accuracy` attributes
- Added `CoordinateSystem.name` attribute
- Added `Marker.group` and `Scalebar.group` attributes
- Added `Orthomosaic.patches` attribute
- Added `RasterTransform.false_color` attribute
- Added `Sensor.bands` attribute
- Added `Shape.attributes` attribute
- Added `DepthMapsData`, `TiledModelData` and `OrthomosaicData` to `DataSource` enum
- Added `CircularTarget14bit` to `TargetType` enum
- Renamed `CameraReference` class to `Camera.Reference`
- Renamed `ConsolePane` class to `Application.ConsolePane`
- Renamed `MarkerProjection` class to `Marker.Projection`
- Renamed `MarkerProjections` class to `Marker.Projections`
- Renamed `MarkerReference` class `Marker.Reference`
- Renamed `MeshFace` class to `Model.Face`
- Renamed `MeshFaces` class to `Model.Faces`
- Renamed `MeshTexVertex` class to `Model.TexVertex`
- Renamed `MeshTexVertices` class to `Model.TexVertices`
- Renamed `MeshVertex` class to `Model.Vertex`
- Renamed `MeshVertices` class to `Model.Vertices`
- Renamed `PointCloudCameras` class to `PointCloud.Cameras`
- Renamed `PointCloudPoint` class to `PointCloud.Point`

- Renamed PointCloudPoints class to PointCloud.Points
- Renamed PointCloudProjection class to PointCloud.Projection
- Renamed PointCloudProjections class to PointCloud.Projections
- Renamed PointCloudTrack class to PointCloud.Track
- Renamed PointCloudTracks class to PointCloud.Tracks
- Renamed ScalebarReference class to Scalebar.Reference
- Renamed ShapeVertices class to Shape.Vertices
- Renamed Application.enumOpenCLDevices() method to Application.enumGPUDevices()
- Renamed Shape.boundary attribute to Shape.boundary\_type
- Renamed Chunk.accuracy\_cameras to Chunk.camera\_location\_accuracy
- Renamed Chunk.accuracy\_cameras\_ypr to Chunk.camera\_rotation\_accuracy
- Renamed Chunk.accuracy\_markers to Chunk.marker\_location\_accuracy
- Renamed Chunk.accuracy\_projections to Chunk.marker\_projection\_accuracy
- Renamed Chunk.accuracy\_tiepoints to Chunk.tiepoint\_accuracy
- Renamed method argument in Chunk.importMasks() method to source and changed its type to MaskSource
- Replaced preselection argument with generic\_preselection and reference\_preselection arguments in Chunk.matchPhotos() method
- Replaced fit\_cxcy argument with fit\_cx and fit\_cy arguments in Chunk.optimizeCameras() method
- Replaced fit\_k1k2k3 argument with fit\_k1, fit\_k2 and fit\_k3 arguments in Chunk.optimizeCameras() method
- Replaced fit\_p1p2 argument with fit\_p1 and fit\_p2 arguments in Chunk.optimizeCameras() method
- Replaced Application.cpu\_cores\_inactive with Application.cpu\_enable attribute
- Changed type of source\_data argument in Chunk.buildContours() to DataSource
- Changed type of format argument in Chunk.importCameras() and Chunk.exportCameras() methods to Cameras-Format
- Changed type of rotation\_order argument in Chunk.exportCameras() to RotationOrder
- Changed type of format argument in Chunk.exportDem() and Chunk.exportOrthomosaic() methods to Raster-Format
- Changed type of format argument in Chunk.exportMatches() method to MatchesFormat
- Changed type of texture\_format argument in Chunk.exportModel() method to ImageFormat
- Changed type of format argument in Chunk.importModel() and Chunk.exportModel() methods to ModelFormat
- Changed type of format argument in Chunk.exportPoints() method to PointsFormat
- Changed type of tiff\_compression argument in Chunk.exportOrthomosaic() and Chunk.exportOrthophotos() methods to TiffCompression
- Changed type of items argument in Chunk.exportShapes() method to Shape.Type
- Changed type of format argument in Chunk.exportTiledModel() method to TiledModelFormat
- Changed type of mesh\_format argument in Chunk.exportTiledModel() method to ModelFormat
- Changed type of operation argument in Chunk.importMasks() method to MaskOperation

- Changed type of format argument in `Chunk.loadReference()` and `Chunk.saveReference()` methods to `ReferenceFormat`
- Changed type of items argument in `Chunk.saveReference()` method to `ReferenceItems`
- Removed return values from `Camera.open()`, `Chunk.addPhotos()`, `Chunk.alignCameras()`, `Chunk.buildContours()`, `Chunk.buildDem()`, `Chunk.buildDenseCloud()`, `Chunk.buildModel()`, `Chunk.buildOrthomosaic()`, `Chunk.buildPoints()`, `Chunk.buildTexture()`, `Chunk.buildTiledModel()`, `Chunk.buildUV()`, `Chunk.decimateModel()`, `Chunk.detectMarkers()`, `Chunk.estimateImageQuality()`, `Chunk.exportCameras()`, `Chunk.exportDem()`, `Chunk.exportMarkers()`, `Chunk.exportMatches()`, `Chunk.exportModel()`, `Chunk.exportOrthomosaic()`, `Chunk.exportOrthophotos()`, `Chunk.exportPoints()`, `Chunk.exportReport()`, `Chunk.exportShapes()`, `Chunk.exportTiledModel()`, `Chunk.importCameras()`, `Chunk.importDem()`, `Chunk.importMarkers()`, `Chunk.importMasks()`, `Chunk.importModel()`, `Chunk.importShapes()`, `Chunk.loadReference()`, `Chunk.loadReferenceExif()`, `Chunk.matchPhotos()`, `Chunk.optimizeCameras()`, `Chunk.remove()`, `Chunk.saveReference()`, `Chunk.smoothModel()`, `Chunk.thinPointCloud()`, `Chunk.trackMarkers()`, `CirTransform.calibrate()`, `CoordinateSystem.init()`, `DenseCloud.classifyGroundPoints()`, `DenseCloud.compactPoints()`, `DenseCloud.selectMaskedPoints()`, `DenseCloud.selectPointsByColor()`, `Document.alignChunks()`, `Document.append()`, `Document.clear()`, `Document.mergeChunks()`, `Document.open()`, `Document.remove()`, `Document.save()`, `Mask.load()`, `Model.closeHoles()`, `Model.fixTopology()`, `Model.loadTexture()`, `Model.removeComponents()`, `Model.saveTexture()`, `Model.setTexture()`, `NetworkClient.abortBatch()`, `NetworkClient.abortNode()`, `NetworkClient.connect()`, `NetworkClient.pauseBatch()`, `NetworkClient.pauseNode()`, `NetworkClient.resumeBatch()`, `NetworkClient.resumeNode()`, `NetworkClient.setBatchPriority()`, `NetworkClient.setNodePriority()`, `Photo.open()`, `PointCloud.export()`, `RasterTransform.calibrateRange()`, `Thumbnail.load()` methods in favor of exceptions
- Removed `Chunk.exportContours()` method
- Removed obsolete `Matrix.diag()` and `Matrix.translation()` class methods
- Removed unused `focal_length` argument from `Calibration.save()` method
- Modified `Utils.mat2opk()` and `Utils.opk2mat()` methods to work with camera to world rotation matrices

### 3.33 PhotoScan version 1.2.6

No Python API changes

### 3.34 PhotoScan version 1.2.5

- Added `ShapeGroup` and `ShapeVertices` classes
- Added `CoordinateSystem.proj4` and `CoordinateSystem.geogcs` attributes
- Added `Shapes.shapes` and `Shapes.groups` attributes
- Added `Shape.label`, `Shape.vertices`, `Shape.group`, `Shape.has_z`, `Shape.key` and `Shape.selected` attributes
- Added `Shapes.addGroup()`, `Shapes.addShape()` and `Shapes.remove()` methods
- Added `CoordinateSystem.transform()` method
- Added `Matrix.Diag()`, `Matrix.Rotation()`, `Matrix.Translation()` and `Matrix.Scale()` class methods
- Added `Matrix.rotation()` and `Matrix.scale()` methods
- Added `DenseCloud.restorePoints()` and `DenseCloud.selectPointsByColor()` methods

- Added `Application.captureModelView()` method
- Added `Mask.invert()` method
- Added `adaptive_fitting` parameter to `Chunk.alignCameras()` method
- Added `load_rotation` and `load_accuracy` parameters to `Chunk.loadReferenceExif()` method
- Added `source` parameter to `Chunk.buildTiledModel()` method
- Added `fill_holes` parameter to `Chunk.buildTexture()` method

### 3.35 PhotoScan version 1.2.4

- 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

### 3.36 PhotoScan version 1.2.3

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

### 3.37 PhotoScan version 1.2.2

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

### 3.38 PhotoScan version 1.2.1

- 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
- Renamed `RasterTransform` enum to `RasterTransformType`

### 3.39 PhotoScan version 1.2.0

- 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.exportMarkers() and Chunk.importMarkers() methods
- Added Chunk.importDem() method
- Added Chunk.buildDem(), Chunk.buildOrthomosaic() and Chunk.buildTiledModel() methods
- Added PointCloud.removeSelectedPoints() and PointCloud.cropSelectedPoints() methods
- 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 CameraReference.accuracy, MarkerReference.accuracy and ScalebarReference.accuracy attributes
- Added Application.activated attribute
- Added Chunk.image\_brightness attribute
- Added fit\_p3 and fit\_p4 parameters to Chunk.optimizeCameras() method
- 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
- Renamed Chunk.exportModelTiled() to Chunk.exportTiledModel()
- Renamed Chunk.exportOrthophoto() to Chunk.exportOrthomosaic()
- Removed OrthoSurface and PointsSource enums
- Removed PointCloud.groups attribute
- Removed Chunk.camera\_offset attribute

### 3.40 PhotoScan version 1.1.1

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

### 3.41 PhotoScan version 1.1.0

- Added `CameraOffset` and `ConsolePane` classes
- Added `CameraGroup`, `CameraReference`, `ChunkTransform`, `DepthMap`, `DepthMaps`, `MarkerReference`, `MarkerProjection`, `Mask`, `PointCloudGroups`, `PointCloudTrack`, `PointCloudTracks`, `ScalebarReference`, `Thumbnail` classes
- Added `Chunk.key`, `Sensor.key`, `Camera.key`, `Marker.key` and `Scalebar.key` attributes
- Added `Application.console` attribute
- Added `Application.addMenuSeparator()` method
- Added `Chunk.importMasks()` method
- Added `Chunk.addSensor()`, `Chunk.addCameraGroup()`, `Chunk.addCamera()`, `Chunk.addMarker()`, `Chunk.addScalebar()` methods
- Added `Chunk.addPhotos()`, `Chunk.addFrame()` methods
- Added `Chunk.master_channel` and `Chunk.camera_offset` attributes
- Added `Calibration.error()` method
- Added `Matrix.mulp()` and `Matrix.mulv()` methods
- 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 `PointCloud.tracks`, `PointCloud.groups` attributes
- Added `Image.tostring()` and `Image.fromstring()` methods
- Added `Image.channels` property
- Added U16 data type support in `Image` class
- 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
- Moved OpenCL settings into `Application` class
- Converted string constants to enum objects
- Removed `Cameras`, `Chunks`, `DenseClouds`, `Frame`, `Frames`, `GroundControl`, `GroundControlLocations`, `GroundControlLocation`, `Markers`, `MarkerPositions`, `Models`, `Scalebars`, `Sensors` classes

## 3.42 PhotoScan version 1.0.0

- Added DenseCloud and DenseClouds classes
- Added Chunk.exportModel() and Chunk.importModel() methods
- Added Chunk.estimateImageQuality() method
- Added Chunk.buildDenseCloud() and Chunk.smoothModel() methods
- Added Photo.thumbnail() method
- Added Image.resize() method
- Added Application.enumOpenCLDevices() method
- Added Utils.estimateImageQuality() 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
- Added shortcut parameter to Application.addItem() method
- Added absolute\_paths parameter to Document.save() method
- Added fit\_f, fit\_cxycy, fit\_k1k2k3 and fit\_k4 parameters to Chunk.optimizePhotos() method
- 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
- Removed Chunk.buildDepth() method
- Removed Camera.depth() and Camera.setDepth() methods
- Removed Frame.depth() and Frame.setDepth() methods
- Removed Frame.depth\_calib attribute

## 3.43 PhotoScan version 0.9.1

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



- Added `Model.closeHoles()` and `Model.fixTopology()` methods

### 3.44 PhotoScan version 0.9.0

- Added `Camera`, `Frame` and `CoordinateSystem` classes
- 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()` and `Chunk.resetDepth()` methods
- Added `Chunk.cameras` property
- Added `Utils.createDifferenceMask()` method
- Revised `Chunk.alignPhotos()` method
- Revised `Chunk.buildPoints()` method
- Revised `Chunk.buildModel()` method
- Removed `Photo` class (deprecated)
- Removed `GeoProjection` class (deprecated)
- Removed `Chunk.photos` property (deprecated)

### 3.45 PhotoScan version 0.8.5

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

### 3.46 PhotoScan version 0.8.4

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

### 3.47 PhotoScan version 0.8.3

Initial version of PhotoScan Python API

## PYTHON MODULE INDEX

m

Metashape, 5