

---

# Metashape Python Reference

*Release 2.3.1*

**Agisoft LLC**

**Apr 18, 2026**



# CONTENTS

<b>1 Overview</b>	<b>3</b>
<b>2 Application Modules</b>	<b>5</b>
<b>3 Python API Change Log</b>	<b>323</b>
<b>Python Module Index</b>	<b>367</b>



Copyright (c) 2026 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.buildModel(source_data=Metashape.DepthMapsData, 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.

#### **bias**

GNSS bias.

#### **Type**

*Metashape.Vector*

#### **bias\_acc**

GNSS bias accuracy.

#### **Type**

*Metashape.Vector*

#### **bias\_covariance**

GNSS bias covariance.

#### **Type**

*Metashape.Matrix*

**bias\_fixed**

Fix GNSS bias flag.

**Type**  
bool

**bias\_ref**

GNSS bias reference.

**Type**  
*Metashape.Vector*

**copy()**

Return a copy of the object.

**Returns**  
A copy of the object.

**Return type**  
*Metashape.Antenna*

**fixed**

Fix antenna location and rotation flags.

**Type**  
bool

**location**

Antenna coordinates.

**Type**  
*Metashape.Vector*

**location\_acc**

Antenna location accuracy.

**Type**  
*Metashape.Vector*

**location\_covariance**

Antenna location covariance.

**Type**  
*Metashape.Matrix*

**location\_fixed**

Fix antenna location flag.

**Type**  
bool

**location\_ref**

Antenna location reference.

**Type**  
*Metashape.Vector*

**rotation**

Antenna rotation angles.

**Type**  
*Metashape.Vector*

**rotation\_acc**

Antenna rotation accuracy.

**Type**

*Metashape.Vector*

**rotation\_covariance**

Antenna rotation covariance.

**Type**

*Metashape.Matrix*

**rotation\_fixed**

Fix antenna rotation flag.

**Type**

bool

**rotation\_ref**

Antenna rotation reference.

**Type**

*Metashape.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**

str

**class ModelView**

ModelView class provides access to the model view

**class ModelViewMode**

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

**class PointCloudViewMode**

Point cloud view mode in [PointCloudViewSolid, PointCloudViewColor, PointCloudViewClassification, PointCloudViewIntensity, PointCloudViewElevation, PointCloudViewHeightAGL, PointCloudViewConfidence, PointCloudViewReturnNumber, PointCloudViewScanAngle, PointCloudViewSourceId, PointCloudViewTime, PointCloudViewDifference]

**class TiePointsViewMode**

Tie points view mode in [TiePointsViewColor, TiePointsViewElevation, TiePointsViewImageCount, TiePointsViewVariance]

**class TiledModelViewMode**

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

**captureVideo**(*path*, *width*, *height* [, *frame\_rate*] [, *transparent*] [, *compressed*] [, *hide\_items* ])

Capture video using camera track. Transparent capture can't be compressed. Method requires gui and inaccessible from python module. If script is passed as a program argument, `-gui` flag should be specified.

**Parameters**

- **path** (*str:arg width: Video width.*) – Output path.
- **height** (*int*) – Video height.
- **frame\_rate** (*int*) – Video frame rate.
- **transparent** (*bool*) – Sets transparent background.
- **compressed** (*bool*) – Enables video compression.
- **hide\_items** (*bool*) – Hides all items.

**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**

*Metashape.Image*

**model\_view\_mode**

Model view mode.

**Type**

*Metashape.Application.ModelView.ModelViewMode*

**point\_cloud\_view\_mode**

Point cloud view mode.

**Type**

*Metashape.Application.ModelView.PointCloudViewMode*

**show\_basemap**

Show or hide basemap.

**Type**

bool

**show\_cameras**

Show or hide cameras.

**Type**

bool

**show\_elevation**

Display digital elevation model.

**Type**

bool

**show\_laser\_scans**

Show or hide laser scans.

**Type**

bool

**show\_markers**

Show or hide markers.

**Type**

bool

**show\_orthomosaic**

Display orthomosaic.

**Type**

bool

**show\_shapes**

Show or hide shapes.

**Type**

bool

**show\_trajectory**

Show or hide trajectory.

**Type**

bool

**texture\_view\_mode**

Texture view mode.

**Type**

*Metashape.Model.TextureType*

**tie\_points\_view\_mode**

Tie points view mode.

**Type**

*Metashape.Application.ModelView.TiePointsViewMode*

**tiled\_model\_view\_mode**

Tiled model view mode.

**Type**

*Metashape.Application.ModelView.TiledModelViewMode*

**view\_mode**

View mode.

**Type**

*Metashape.DataSource*

**viewpoint**

Viewpoint in the model view.

**Type**

*Metashape.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**

*Metashape.Image*

**center**

Ortho view center coordinates.

**Type**

*Metashape.Vector*

**height**

Ortho view window height.

**Type**

int

**projection**

Ortho view projection.

**Type**

*Metashape.OrthoProjection*

**scale**

Ortho view scale in px/m.

**Type**

float

**view\_mode**

View mode.

**Type**

*Metashape.DataSource*

**width**

Ortho view window width.

**Type**

int

**class PhotoView**

PhotoView class provides access to the photo view

**camera**

Get currently opened camera.

**Type**

*Metashape.Camera*

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

Capture image from photo 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**

*Metashape.Image*

**center**

Image coordinates at photo view center.

**Type**

*Metashape.Vector*

**close()**

Close active photo view.

**height**

Photo view window height.

**Type**

int

**open**(*camera*, *new\_tab=False*)

Open camera in photo view.

**Parameters**

- **camera** (*Metashape.Camera*) – Camera to open.
- **new\_tab** (*bool*) – Open camera in new tab.

**scale**

Photo view scale in view px / image px

**Type**

float

**show\_markers**

Show or hide markers.

**Type**

bool

**show\_shapes**

Show or hide shapes.

**Type**

bool

**width**

Photo view window width.

**Type**  
int

**class PhotosPane**

PhotosPane class provides access to the photos pane

**resetFilter()**

Reset photos pane filter.

**setFilter(*items*)**

Set photos pane filter.

**Parameters**

**items** (*list* [*Metashape.Camera* / *Metashape.Marker*]) – filter to apply.

**class Settings**

PySettings()

Application settings

**language**

User interface language.

**Type**  
str

**load()**

Load settings from disk.

**log\_enable**

Enable writing log to file.

**Type**  
bool

**log\_path**

Log file path.

**Type**  
str

**network\_enable**

Network processing enabled flag.

**Type**  
bool

**network\_host**

Network server host name.

**Type**  
str

**network\_path**

Network data root path.

**Type**  
str

**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.

**Parameters**

- **key** (*str*) – Key.
- **value** (*object*) – Value.

**value(key)**

Return settings value.

**Parameters**

**key** (*str*) – Key.

**Returns**

Settings value.

**Return type**

object

**activated**

Metashape activation status.

**Type**

bool

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

Create a new menu entry.

**Parameters**

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

**addMenuSeparator(label)**

Add menu separator.

**Parameters**

**label** (*str*) – Menu label.

**console\_pane**

Console pane.

**Type**

*Metashape.Application.ConsolePane*

**cpu\_enable**

Use CPU when GPU is active.

**Type**

bool

**document**

Main application document object.

**Type**

*Metashape.Document*

**enumGPUDevices()**

Enumerate installed GPU devices.

**Returns**

A list of devices.

**Return type**

list

**getBool(label="")**

Prompt user for the boolean value.

**Parameters**

**label** (*str*) – 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** (*str*) – Optional text label for the dialog.
- **value** (*Metashape.CoordinateSystem*) – Default value.

**Returns**

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

**Return type**

*Metashape.CoordinateSystem*

**getExistingDirectory([hint ][, dir ])**

Prompt user for the existing folder.

**Parameters**

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

**Returns**

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

**Return type**

str

**getFloat**(*label=""*, *value=0*)

Prompt user for the floating point value.

**Parameters**

- **label** (*str*) – 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** (*str*) – 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** (*str*) – Optional text label for the dialog.
- **dir** (*str*) – Optional default folder.
- **filter** (*str*) – 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**

str

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

Prompt user for one or more existing files.

**Parameters**

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

str

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

Prompt user for the string value.

**Parameters**

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

**Returns**

String entered by the user.

**Return type**

str

**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** (*str*) – Text message to be displayed.

**model\_view**

Model view.

**Type**

*Metashape.Application.ModelView*

**ortho\_view**

Ortho view.

**Type**

*Metashape.Application.OrthoView*

**photo\_view**

Photo view.

**Type**

*Metashape.Application.PhotoView*

**photos\_pane**

Photos pane.

**Type**

*Metashape.Application.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** (*str*) – Menu item label.

**settings**

Application settings.

**Type**

*Metashape.Application.Settings*

**title**

Application name.

**Type**

*str*

**update()**

Update user interface during long operations.

**version**

Metashape version.

**Type**

*str*

**class Metashape.AttachedGeometry**

Attached geometry data.

**GeometryCollection(*geometries*)**

Create a GeometryCollection geometry.

**Parameters**

**geometries** (*list*[*Metashape.AttachedGeometry*]) – Child geometries.

**Returns**

A GeometryCollection geometry.

**Return type**

*Metashape.AttachedGeometry*

**LineString(*coordinates*)**

Create a LineString geometry.

**Parameters**

**coordinates** (*list*[*int*]) – List of vertex coordinates.

**Returns**

A LineString geometry.

**Return type**

*Metashape.AttachedGeometry*

**MultiLineString**(*geometries*)

Create a MultiLineString geometry.

**Parameters**

**geometries** (*list* [*Metashape.AttachedGeometry*]) – Child line strings.

**Returns**

A point geometry.

**Return type**

*Metashape.AttachedGeometry*

**MultiPoint**(*geometries*)

Create a MultiPoint geometry.

**Parameters**

**geometries** (*list* [*Metashape.AttachedGeometry*]) – Child points.

**Returns**

A point geometry.

**Return type**

*Metashape.AttachedGeometry*

**MultiPolygon**(*geometries*)

Create a MultiPolygon geometry.

**Parameters**

**geometries** (*list* [*Metashape.AttachedGeometry*]) – Child polygons.

**Returns**

A point geometry.

**Return type**

*Metashape.AttachedGeometry*

**Point**(*key*)

Create a Point geometry.

**Parameters**

**key** (*int*) – Point marker key.

**Returns**

A point geometry.

**Return type**

*Metashape.AttachedGeometry*

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

Create a Polygon geometry.

**Parameters**

- **exterior\_ring** (*list* [*int*]) – Point coordinates.
- **interior\_rings** (*list* [*int*]) – Point coordinates.

**Returns**

A Polygon geometry.

**Return type**

*Metashape.AttachedGeometry*

**coordinates**

List of vertex keys.

**Type**

list[int]

**geometries**

List of child geometries.

**Type**

list[*Metashape.AttachedGeometry*]

**type**

Geometry type.

**Type**

*Metashape.Geometry.Type*

**class Metashape.BBox**

Axis aligned bounding box

**copy()**

Return a copy of the object.

**Returns**

A copy of the object.

**Return type**

*Metashape.BBox*

**max**

Maximum bounding box extent.

**Type**

*Metashape.Vector*

**min**

Minimum bounding box extent.

**Type**

*Metashape.Vector*

**size**

Bounding box dimension.

**Type**

int

**class Metashape.BlendingMode**

Blending mode in [AverageBlending, MosaicBlending, NaturalBlending, 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**  
*Metashape.Calibration*

**covariance\_matrix**

Covariance matrix.

**Type**  
*Metashape.Matrix*

**covariance\_params**

Covariance matrix parameters.

**Type**  
list[str]

**cx**

Principal point X coordinate.

**Type**  
float

**cy**

Principal point Y coordinate.

**Type**  
float

**error(point, proj)**

Return projection error.

**Parameters**

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

**Returns**  
2D projection error.

**Return type**  
*Metashape.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** (*str*) – path to calibration file
- **format** (*Metashape.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** (*Metashape.Vector*) – Coordinates of the point to be projected.

**Returns**

2D projected point coordinates.

**Return type**

*Metashape.Vector*

**rpc**

RPC model.

**Type**

*Metashape.RPCModel*

**save**(*path*, *format=CalibrationFormatXML* [, *label*] [, *pixel\_size*] [, *film\_size*] [, *focal\_length* ], *cx* = 0, *cy* = 0)

Saves calibration to file.

**Parameters**

- **path** (*str*) – path to calibration file
- **format** (*Metashape.CalibrationFormat*) – Calibration format.
- **label** (*str*) – Calibration label.
- **pixel\_size** (*Metashape.Vector*) – Pixel size in mm for digital camera.
- **film\_size** (*Metashape.Vector*) – Film size in mm for film camera with fiducial marks (Inpho and USGS formats only).
- **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).

**todict**(*format=CalibrationFormatXML* [, *pixel\_size*] [, *film\_size*] [, *params* ])

Return calibration coefficients as a dict.

**Parameters**

- **format** (*Metashape.CalibrationFormat*) – Calibration format.
- **pixel\_size** (*Metashape.Vector*) – Pixel size in mm for digital camera.
- **film\_size** (*Metashape.Vector*) – Film size in mm for film camera with fiducial marks (Inpho and USGS formats only).
- **params** (*list[str]*) – List of calibration coefficients to fit (USGS format only).

**Returns**

Calibration coefficients.

**Return type**

dict

**type**

Camera model.

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

Return direction corresponding to the image point.

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

3D vector in the camera coordinate system.

**Return type***Metashape.Vector***width**

Image width.

**Type**

int

**class Metashape.CalibrationFormat**

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

**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: Similar approach can be used to create multiplane camera layout as well.

```
>>> 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]]
>>> groups = [2, 2, 2]
>>> chunk.addPhotos(filenamees=images, filegroups=groups, layout=Metashape.
↳ MultiplaneLayout)
```

**class Reference**

Camera reference data.

**accuracy**

Camera location accuracy.

**Type**

*Metashape.Vector*

**enabled**

Location enabled flag.

**Type**

bool

**location**

Camera coordinates.

**Type**

*Metashape.Vector*

**location\_accuracy**

Camera location accuracy.

**Type**

*Metashape.Vector*

**location\_enabled**

Location enabled flag.

**Type**

bool

**rotation**

Camera rotation angles.

**Type**

*Metashape.Vector*

**rotation\_accuracy**

Camera rotation accuracy.

**Type**

*Metashape.Vector*

**rotation\_enabled**

Rotation enabled flag.

**Type**

bool

**class Type**

Camera type in [Regular, Keyframe]

**calibration**

Adjusted camera calibration including photo-invariant parameters.

**Type**

*Metashape.Calibration*

**center**

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

**Type**

*Metashape.Vector*

**chunk**

Chunk the camera belongs to.

**Type***Metashape.Chunk***component**

Camera component.

**Type***Metashape.Component***enabled**

Enables/disables the photo.

**Type**

bool

**error**(*point*, *proj*)

Returns projection error.

**Parameters**

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

**Returns**

2D projection error.

**Return type***Metashape.Vector***film\_transform**

3x3 transformation matrix from film to image coordinates.

**Type***Metashape.Matrix***frames**

Camera frames.

**Type**list[*Metashape.Camera*]**group**

Camera group.

**Type***Metashape.CameraGroup***height**

Image height.

**Type**

int

**image**()

Returns image data.

**Returns**

Image data.

**Return type***Metashape.Image*

**key**

Camera identifier.

**Type**  
int

**label**

Camera label.

**Type**  
str

**layer\_index**

Camera layer index.

**Type**  
int

**location\_covariance**

Camera location covariance.

**Type**  
*Metashape.Matrix*

**mask**

Camera mask.

**Type**  
*Metashape.Mask*

**master**

Master camera.

**Type**  
*Metashape.Camera*

**meta**

Camera meta data.

**Type**  
*Metashape.MetaData*

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

Loads specified image file.

**Parameters**

- **path** (*str*) – 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**  
*Metashape.Photo*

**planes**

Camera planes.

**Type**

list[*Metashape.Camera*]

**point\_cloud**

Laser scan for attached cameras.

**Type**

*Metashape.PointCloud*

**project** (*point*)

Returns coordinates of the point projection on the photo.

**Parameters**

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

**Returns**

2D point coordinates.

**Return type**

*Metashape.Vector*

**reference**

Camera reference data.

**Type**

*Metashape.Camera.Reference*

**rotation\_covariance**

Camera rotation covariance.

**Type**

*Metashape.Matrix*

**selected**

Selects/deselects the photo.

**Type**

bool

**sensor**

Camera sensor.

**Type**

*Metashape.Sensor*

**shutter**

Camera shutter.

**Type**

*Metashape.Shutter*

**thumbnail**

Camera thumbnail.

**Type**

*Metashape.Thumbnail*

**transform**

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

**Type**

*Metashape.Matrix*

**type**

Camera type.

**Type**

*Metashape.Camera.Type*

**unproject**(*point*)

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

**Parameters**

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

**Returns**

3D point coordinates.

**Return type**

*Metashape.Vector*

**vignetting**

Vignetting for each band.

**Type**

list[*Metashape.Vignetting*]

**width**

Image width.

**Type**

int

**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]

**key**

Camera group identifier.

**Type**

int

**label**

Camera group label.

**Type**

str

**selected**

Current selection state.

**Type**  
bool

**type**  
Camera group type.

**Type**  
*Metashape.CameraGroup.Type*

**class Metashape.CameraTrack**

Camera track.

**chunk**  
Chunk the camera track belongs to.

**Type**  
*Metashape.Chunk*

**duration**  
Animation duration.

**Type**  
float

**field\_of\_view**  
Vertical field of view in degrees.

**Type**  
float

**interpolate(*time*)**  
Get animation camera transform matrix.

**Parameters**  
**time** (*float*) – Animation time point.

**Returns**  
Interpolated camera transformation matrix in chunk coordinate system.

**Return type**  
*Metashape.Matrix*

**key**  
Camera track identifier.

**Type**  
int

**keyframes**  
Camera track keyframes.

**Type**  
list[*Metashape.Camera*]

**label**  
Animation label.

**Type**  
str

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

Load camera track from file.

**Parameters**

- **path** (*str*) – Path to camera track file
- **projection** ([Metashape.CoordinateSystem](#)) – Camera track coordinate system.

**loop**

Loop track.

**Type**

bool

**meta**

Camera track meta data.

**Type**

[Metashape.MetaData](#)

**save**(*path*[, *file\_format* ][, *drone\_name* ][, *payload\_name* ][, *payload\_position* ][, *max\_waypoints* ][, *projection* ])

Save camera track to file.

**Parameters**

- **path** (*str*) – Path to camera track file
- **file\_format** (*str*) – File format. “deduce”: - Deduce from extension, “path”: Path, “earth”: Google Earth KML, “pilot”: DJI Pilot KML, “wpml”: DJI WPML KMZ, “trinity”: Asctec Trinity CSV, “autopilot”: Asctec Autopilot CSV, “litchi”: Litchi CSV
- **drone\_name** (*str*) – Drone model. “M300 RTK”: - DJI Matrice 300 RTK, “M30”: - DJI Matrice 30, “M30T”: - DJI Matrice 30T, “M3E”: - DJI Mavic 3E, “M3T”: - DJI Mavic 3T
- **payload\_name** (*str*) – Payload model. “P1 24mm”: - DJI Zenmuse P1 (24 mm lens), “P1 35mm”: - DJI Zenmuse P1 (35 mm lens), “P1 50mm”: - DJI Zenmuse P1 (50 mm lens), “H20”: - DJI Zenmuse H20, “H20T”: - DJI Zenmuse H20T, “H20N”: - DJI Zenmuse H20N, “L1”: - DJI Zenmuse L1, “M30”: - DJI M30, “M30T”: - DJI M30T, “M3E”: - DJI Mavic 3E Camera, “M3T”: - DJI Mavic 3T Camera
- **payload\_position** (*str*) – Payload position. For M300 RTK drone: “Front left”, “Front right”, “Top”. For other drones: “Main gimbal”
- **max\_waypoints** (*int*) – Max waypoints per flight
- **projection** ([Metashape.CoordinateSystem](#)) – Camera track coordinate system.

**selected**

Current selection state.

**Type**

bool

**smooth**

Smooth path.

**Type**

bool

**class Metashape.CamerasFormat**

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

**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 (tie points, model, etc)
- implements processing methods (matchPhotos, alignCameras, buildPointCloud, 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** (*Metashape.Sensor*) – Sensor to be assigned to this camera.

**Returns**

Created camera.

**Return type**

*Metashape.Camera*

**addCameraGroup()**

Add new camera group to the chunk.

**Returns**

Created camera group.

**Return type**

*Metashape.CameraGroup*

**addCameraTrack()**

Add new camera track to the chunk.

**Returns**

Created camera track.

**Return type**

*Metashape.CameraTrack*

**addDepthMaps()**

Add new depth maps set to the chunk.

**Returns**

Created depth maps set.

**Return type**

*Metashape.DepthMaps*

**addElevation()**

Add new elevation model to the chunk.

**Returns**

Created elevation model.

**Return type**

*Metashape.Elevation*

**addFrame()**

Add new frame to the chunk.

**Returns**

Created frame.

**Return type**

*Metashape.Chunk*

**addFrames**(*[chunk]*, *[frames]*, *copy\_depth\_maps=True*, *copy\_point\_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[int]*) – List of frame keys to copy.
- **copy\_depth\_maps** (*bool*) – Copy depth maps.
- **copy\_point\_cloud** (*bool*) – Copy point 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** (*Metashape.Vector*) – Point to initialize marker projections.
- **visibility** (*bool*) – Enables visibility check during projection assignment.

**Returns**

Created marker.

**Return type***Metashape.Marker***addMarkerGroup()**

Add new marker group to the chunk.

**Returns**

Created marker group.

**Return type***Metashape.MarkerGroup***addModel()**

Add new model to the chunk.

**Returns**

Created model.

**Return type***Metashape.Model***addModelGroup()**

Add new model group to the chunk.

**Returns**

Created model group.

**Return type***Metashape.ModelGroup***addOrthomosaic()**

Add new orthomosaic to the chunk.

**Returns**

Created orthomosaic.

**Return type***Metashape.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[str]*) – List of files to add.
- **filegroups** (*list[int]*) – List of file groups. Can be used to create Multiframe / Multiplane layout. In that case list values are number of sequential photos for each camera.
- **layout** (*Metashape.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 GNSS/INS offset from XMP meta data.
- **load\_rpc\_txt** (*bool*) – Load satellite RPC data from auxiliary TXT files.
- **progress** (*Callable[[float], None]*) – Progress callback.

#### **addPointCloud()**

Add new point cloud to the chunk.

##### **Returns**

Created point cloud.

##### **Return type**

*Metashape.PointCloud*

#### **addPointCloudGroup()**

Add new point cloud group to the chunk.

##### **Returns**

Created point cloud group.

##### **Return type**

*Metashape.PointCloudGroup*

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

Add new scale bar to the chunk.

##### **Parameters**

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

##### **Returns**

Created scale bar.

##### **Return type**

*Metashape.Scalebar*

#### **addScalebarGroup()**

Add new scale bar group to the chunk.

##### **Returns**

Created scale bar group.

##### **Return type**

*Metashape.ScalebarGroup*

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

Add new sensor to the chunk.

##### **Parameters**

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

##### **Returns**

Created sensor.

##### **Return type**

*Metashape.Sensor*

#### **addTiledModel()**

Add new tiled model to the chunk.

**Returns**

Created tiled model.

**Return type**

*Metashape.TiledModel*

**addTrajectory()**

Add new trajectory to the chunk.

**Returns**

Created trajectory.

**Return type**

*Metashape.Trajectory*

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

Perform photo alignment for the chunk.

**Parameters**

- **cameras** (*list[int]*) – List of cameras to align.
- **point\_clouds** (*list[int]*) – List of point clouds 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.
- **align\_laser\_scans** (*bool*) – Align laser scans using geometric features.
- **progress** (*Callable[[float], None]*) – Progress callback.

**alignCamerasByReference**(*[cameras]*)

Use photo reference metadata as alignment.

**Parameters**

**cameras** (*list[Metashape.Camera]*) – Optional list of cameras to process.

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

Estimate image quality. Estimated value is stored in camera metadata with Image/Quality key. Cameras with quality less than 0.5 are considered blurred and we recommend to disable them.

**Parameters**

- **cameras** (*list[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** (*Metashape.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=PointCloudData, interpolation=EnabledInterpolation*[, *projection*][, *region*][, *classes*], *resolution=0, subdivide\_task=True, workitem\_size\_tiles=10, max\_workgroup\_size=100, replace\_asset=False*[, *frames*][, *progress*])

Build elevation model for the chunk.

#### Parameters

- **source\_data** (*Metashape.DataSource*) – Selects between point cloud and tie points.
- **interpolation** (*Metashape.Interpolation*) – Interpolation mode.
- **projection** (*Metashape.OrthoProjection*) – Output projection.
- **region** (*Metashape.BBox*) – Region to be processed.
- **classes** (*list[int]*) – List of point classes to be used for surface extraction.
- **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.
- **replace\_asset** (*bool*) – Replace default asset with generated DEM.
- **frames** (*list[int]*) – List of frames to process.
- **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 (1 - Ultra high, 2 - High, 4 - Medium, 8 - Low, 16 - Lowest).
- **filter\_mode** (*Metashape.FilterMode*) – Depth map filtering mode.
- **cameras** (*list[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=DepthMapsData[, classes ], vertex_colors=True,
vertex_confidence=True, volumetric_masks=False, keep_depth=True, replace_asset=False,
split_in_blocks=False[, blocks_crs ], blocks_size=250[, blocks_origin ],
clip_to_boundary=False, export_blocks=False, build_texture=True, output_folder="",
trimming_radius=10[, cameras ][, frames ], subdivide_task=True, workitem_size_cameras=20,
max_workgroup_size=100[, progress ])
```

Generate model for the chunk frame.

### Parameters

- **surface\_type** (`Metashape.SurfaceType`) – Type of object to be reconstructed.
- **interpolation** (`Metashape.Interpolation`) – Interpolation mode.
- **face\_count** (`Metashape.FaceCount`) – Target face count.
- **face\_count\_custom** (*int*) – Custom face count.
- **source\_data** (`Metashape.DataSource`) – Selects between point cloud, tie points, depth maps and laser scans.
- **classes** (*list[int]*) – List of point classes to be used for surface extraction.
- **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.
- **replace\_asset** (*bool*) – Replace default asset with generated model.
- **split\_in\_blocks** (*bool*) – Split model in blocks.
- **blocks\_crs** (`Metashape.CoordinateSystem`) – Blocks grid coordinate system.
- **blocks\_size** (*float*) – Blocks size in coordinate system units.
- **blocks\_origin** (`Metashape.Vector`) – Blocks grid origin.
- **clip\_to\_boundary** (*bool*) – Clip to boundary shapes.
- **export\_blocks** (*bool*) – Export completed blocks.
- **build\_texture** (*bool*) – Generate preview textures.
- **output\_folder** (*str*) – Path to output folder.
- **trimming\_radius** (*int*) – Trimming radius (no trimming if zero).
- **cameras** (*list[int]*) – List of cameras to process.
- **frames** (*list[int]*) – List of frames to process.
- **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, color_enhancement=False, cull_faces=False,
                 refine_seamlines=False[, projection ][, region ], resolution=0, resolution_x=0,
                 resolution_y=0, transfer_texture=False, use_occlusion_texture=False,
                 use_point_cloud_intensity=False, subdivide_task=True, workitem_size_cameras=20,
                 workitem_size_tiles=10, max_workgroup_size=100, replace_asset=False[, frames ][,
                 progress ])
```

Build orthomosaic for the chunk.

#### Parameters

- **surface\_data** ([Metashape.DataSource](#)) – Orthorectification surface.
- **blending\_mode** ([Metashape.BlendingMode](#)) – Orthophoto blending mode.
- **fill\_holes** (*bool*) – Enable hole filling.
- **ghosting\_filter** (*bool*) – Enable ghosting filter.
- **color\_enhancement** (*bool*) – Enable automatic color enhancement.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **refine\_seamlines** (*bool*) – Refine seamlines based on image content.
- **projection** ([Metashape.OrthoProjection](#)) – Output projection.
- **region** ([Metashape.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.
- **transfer\_texture** (*bool*) – Transfer model texture to orthomosaic.
- **use\_occlusion\_texture** (*bool*) – Use model occlusion texture.
- **use\_point\_cloud\_intensity** (*bool*) – Use point cloud intensity as color source.
- **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.
- **replace\_asset** (*bool*) – Replace default asset with generated orthomosaic.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
buildPanorama(blending_mode=MosaicBlending, ghosting_filter=False, color_enhancement=False[,
               rotation ][, region ], width=0, height=0[, camera_groups ][, frames ][, progress ])
```

Generate spherical panoramas from camera stations.

#### Parameters

- **blending\_mode** ([Metashape.BlendingMode](#)) – Panorama blending mode.
- **ghosting\_filter** (*bool*) – Enable ghosting filter.
- **color\_enhancement** (*bool*) – Enable automatic color enhancement.
- **rotation** ([Metashape.Matrix](#)) – Panorama 3x3 orientation matrix.

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

**buildPointCloud**(`source_data=DepthMapsData, point_colors=True, point_confidence=False, keep_depth=True, max_neighbors=100, uniform_sampling=True, points_spacing=0.1[, asset]`, `subdivide_task=True, workitem_size_cameras=20, max_workgroup_size=100, replace_asset=False[, frames]`[, `progress` ])

Generate point cloud for the chunk.

#### Parameters

- **source\_data** (`Metashape.DataSource`) – Source data to extract points from.
- **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.
- **uniform\_sampling** (`bool`) – Enable uniform point sampling.
- **points\_spacing** (`float`) – Desired point spacing (m).
- **asset** (`int`) – Asset to process.
- **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.
- **replace\_asset** (`bool`) – Replace default asset with generated point cloud.
- **frames** (`list[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=NaturalBlending, texture_size=8192, downscale=2, sharpening=1, use_assigned_images=False, fill_holes=True, ghosting_filter=True, out_of_focus_filter=False, color_enhancement=False[, cameras]`, `texture_type=DiffuseMap, source_data=ImagesData[, source_asset]`, `transfer_texture=True, workitem_size_cameras=20, max_workgroup_size=100, anti_aliasing=1[, progress]`)

Generate texture for the chunk.

#### Parameters

- **blending\_mode** (`Metashape.BlendingMode`) – Texture blending mode.
- **texture\_size** (`int`) – Texture page size.
- **downscale** (`int`) – Images downscale (natural blending only).
- **sharpening** (`float`) – Sharpening strength (natural blending only).
- **use\_assigned\_images** (`bool`) – Use assigned images when blending.
- **fill\_holes** (`bool`) – Enable hole filling.
- **ghosting\_filter** (`bool`) – Enable ghosting filter.
- **out\_of\_focus\_filter** (`bool`) – Enable out-of-focus filter (natural blending only).
- **color\_enhancement** (`bool`) – Enable automatic color enhancement.
- **cameras** (`list[int]`) – A list of cameras to be used for texturing.
- **texture\_type** (`Metashape.Model.TextureType`) – Texture type.
- **source\_data** (`Metashape.DataSource`) – Source data to create texture from.
- **source\_asset** (`int`) – Source asset.
- **transfer\_texture** (`bool`) – Transfer texture.
- **workitem\_size\_cameras** (`int`) – Number of cameras in a workitem (block model only).
- **max\_workgroup\_size** (`int`) – Maximum workgroup size (block model only).
- **anti\_aliasing** (`int`) – Anti-aliasing coefficient for baking
- **progress** (`Callable[[float], None]`) – Progress callback.

**buildTiledModel**(`pixel_size=0, tile_size=256, source_data=DepthMapsData, face_count=20000, ghosting_filter=False, color_enhancement=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, replace_asset=False[, frames ][, 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** (`Metashape.DataSource`) – Selects between point cloud and mesh.
- **face\_count** (`int`) – Number of faces per megapixel of texture resolution.
- **ghosting\_filter** (`bool`) – Enable ghosting filter.
- **color\_enhancement** (`bool`) – Enable automatic color enhancement.
- **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[int]`) – List of 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.
- **replace\_asset** (*bool*) – Replace default asset with generated tiled model.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**buildUV**(*mapping\_mode=GenericMapping, page\_count=1, texture\_size=8192, pixel\_size=0*[, *camera*][, *progress*])

Generate uv mapping for the model.

#### Parameters

- **mapping\_mode** (*Metashape.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.
- **pixel\_size** (*float*) – Texture resolution in meters.
- **camera** (*int*) – Camera to be used for texturing in CameraMapping mode.
- **progress** (*Callable[[float], None]*) – Progress callback.

**calculatePointNormals**(*point\_neighbors=28*[, *point\_cloud*][, *point\_clouds*], *replace\_asset=True*[, *frames*][, *progress*])

Calculate point cloud normals.

#### Parameters

- **point\_neighbors** (*int*) – Number of point neighbors to use for normal estimation.
- **point\_cloud** (*int*) – Point cloud key to process.
- **point\_clouds** (*list[int]*) – List of point clouds to process.
- **replace\_asset** (*bool*) – Replace source point cloud with processed one.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

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

Perform radiometric calibration.

#### Parameters

- **source\_data** (*Metashape.DataSource*) – Source data for calibration.
- **white\_balance** (*bool*) – Calibrate white balance.
- **cameras** (*list[int]*) – List of cameras to calibrate.
- **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**

*Metashape.CoordinateSystem*

**camera\_groups**

List of camera groups in the chunk.

**Type**

*list[Metashape.CameraGroup]*

**camera\_location\_accuracy**

Expected accuracy of camera coordinates in meters.

**Type**

*Metashape.Vector*

**camera\_rotation\_accuracy**

Expected accuracy of camera orientation angles in degrees.

**Type**

*Metashape.Vector*

**camera\_track**

Camera track.

**Type**

*Metashape.CameraTrack*

**camera\_tracks**

List of camera tracks in the chunk.

**Type**

*list[Metashape.CameraTrack]*

**cameras**

List of Regular and Keyframe cameras in the chunk.

**Type**

*list[Metashape.Camera]*

**cir\_transform**

CIR calibration matrix.

**Type**

*Metashape.CirTransform*

**cleanModel** (*criterion=ComponentSize, level=0[, model][, progress]*)

Remove model faces based on specified criterion.

**Parameters**

- **criterion** (*Metashape.Model.Criterion*) – Model filtering criterion.
- **level** (*int*) – Filtering threshold in percents.
- **model** (*int*) – Model to filter.

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

**cleanPointCloud**(*point\_cloud*[[*point\_clouds*]], *criterion=Confidence*, *threshold=0*[[*frames*]][, *progress*])

Remove points based on specified criterion.

#### Parameters

- **point\_cloud** (*int*) – Point cloud key to filter.
- **point\_clouds** (*list* [*int*]) – List of point clouds to filter.
- **criterion** (*Metashape.PointCloud.Criterion*) – Point cloud filtering criterion.
- **threshold** (*float*) – Filtering threshold.
- **frames** (*list* [*int*]) – List of frames to process.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**cleanTiePoints**(*criterion=ReprojectionError*, *threshold=0*[[*progress*]])

Remove tie points based on specified criterion.

#### Parameters

- **criterion** (*Metashape.TiePoints.Criterion*) – Tie points filtering criterion.
- **threshold** (*float*) – Filtering threshold.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**colorizeModel**(*source\_data=ImagesData*[[*model*]], *color\_enhancement=False*[[*progress*]])

Calculate vertex colors for the model.

#### Parameters

- **source\_data** (*Metashape.DataSource*) – Source data to extract colors from.
- **model** (*int*) – Key of model to colorize.
- **color\_enhancement** (*bool*) – Enable automatic color enhancement.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**colorizePointCloud**(*source\_data=ImagesData*[[*point\_cloud*]][, *point\_clouds*], *replace\_asset=False*, *color\_enhancement=False*[[*frames*]], *workitem\_size\_cameras=20*, *max\_workgroup\_size=100*, *subdivide\_task=True*[[*progress*]])

Calculate point colors for the point cloud.

#### Parameters

- **source\_data** (*Metashape.DataSource*) – Source data to extract colors from.
- **point\_cloud** (*int*) – Point cloud key to colorize.
- **point\_clouds** (*list* [*int*]) – List of point clouds to colorize.
- **replace\_asset** (*bool*) – Replace source point cloud with colorized one.
- **color\_enhancement** (*bool*) – Enable automatic color enhancement.
- **frames** (*list* [*int*]) – List of frames to process.
- **workitem\_size\_cameras** (*int*) – Number of cameras in a workitem.
- **max\_workgroup\_size** (*int*) – Maximum workgroup size.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.

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

**component**

Component.

**Type**

*Metashape.Component*

**components**

List of components in the chunk.

**Type**

list[*Metashape.Component*]

**convertImages**(*path*='{filename}.{fileext}', *use\_initial\_calibration*=*False*, *color\_correction*=*False*, *color\_enhancement*=*False*, *merge\_planes*=*False*, *update\_gps\_tags*=*False* [, *cameras*] [, *image\_compression*] [, *progress*])

Convert images.

**Parameters**

- **path** (*str*) – Path to output file.
- **use\_initial\_calibration** (*bool*) – Transform to initial calibration.
- **color\_correction** (*bool*) – Apply color correction.
- **color\_enhancement** (*bool*) – Enable automatic color enhancement.
- **merge\_planes** (*bool*) – Merge multispectral images.
- **update\_gps\_tags** (*bool*) – Update GPS tags.
- **cameras** (*list*[*int*]) – List of cameras to process.
- **image\_compression** (*Metashape.ImageCompression*) – Image compression parameters.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**copy**([*frames*] [, *items*] [, *keypoints*=*True*] [, *cameras*] [, *laser\_scans*] [, *progress*])

Make a copy of the chunk.

**Parameters**

- **frames** (*list*[*Metashape.Chunk*]) – Optional list of frames to be copied.
- **items** (*list*[*Metashape.DataSource*]) – A list of items to copy.
- **keypoints** (*bool*) – Copy key points data.
- **cameras** (*list*[*Metashape.Camera*]) – Optional list of cameras to be copied.
- **laser\_scans** (*list*[*Metashape.PointCloud*]) – Optional list of laser scans to be copied.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**Returns**

Copy of the chunk.

**Return type**

*Metashape.Chunk*

**crs**

Coordinate system used for reference data.

**Type**

*Metashape.CoordinateSystem*

**decimateModel**(*face\_count=200000*[, *model* ], *apply\_to\_selection=False*, *replace\_asset=False*[, *frames* ][, *progress* ])

Decimate the model to the specified face count.

**Parameters**

- **face\_count** (*int*) – Target face count.
- **model** (*int*) – Model to process.
- **apply\_to\_selection** (*bool*) – Apply to selection.
- **replace\_asset** (*bool*) – Replace source model with decimated model.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**depth\_maps**

Default depth maps set for the current frame.

**Type**

*Metashape.DepthMaps*

**depth\_maps\_sets**

List of depth maps sets for the current frame.

**Type**

*list[Metashape.DepthMaps]*

**detectFiducials**(*generate\_masks=False*, *mask\_dark\_pixels=True*, *generic\_detector=True*, *right\_angle\_detector=False*, *v\_shape\_detector=False*, *frame\_detector=False*, *fiducials\_position\_corners=True*, *fiducials\_position\_sides=True*[, *cameras* ][, *frames* ][, *progress* ])

Detect fiducial marks on film cameras.

**Parameters**

- **generate\_masks** (*bool*) – Generate background masks.
- **mask\_dark\_pixels** (*bool*) – Mask out dark pixels near frame edge.
- **generic\_detector** (*bool*) – Use generic detector.
- **right\_angle\_detector** (*bool*) – Use right angle detector.
- **v\_shape\_detector** (*bool*) – Detect V-shape fiducials.
- **frame\_detector** (*bool*) – Detect frame.
- **fiducials\_position\_corners** (*bool*) – Search corners for fiducials.
- **fiducials\_position\_sides** (*bool*) – Search sides for fiducials.
- **cameras** (*list[int]*) – List of cameras to process.
- **frames** (*list[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, merge\_markers=False*[, *cameras* ][, *frames* ][, *progress* ])

Create markers from coded targets.

**Parameters**

- **target\_type** (*Metashape.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).
- **merge\_markers** (*bool*) – Merge detected targets with existing markers.
- **cameras** (*list[int]*) – List of cameras to process.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**detectPowerlines**(*min\_altitude=1, n\_points\_per\_line=100, max\_quantization\_error=0.01, use\_model=True*[, *progress* ])

Detect powerlines for the chunk.

**Parameters**

- **min\_altitude** (*float*) – Minimum altitude for reconstructed powerlines.
- **n\_points\_per\_line** (*int*) – Maximum number of vertices per detected line.
- **max\_quantization\_error** (*float*) – Maximum allowed distance between polyline and smooth continuous curve.
- **use\_model** (*bool*) – Use model for visibility checks.
- **progress** (*Callable[[float], None]*) – Progress callback.

**elevation**

Default elevation model for the current frame.

**Type**

*Metashape.Elevation*

**elevations**

List of elevation models for the current frame.

**Type**

*list[Metashape.Elevation]*

**enabled**

Enables/disables the chunk.

**Type**

bool

**euler\_angles**

Euler angles triplet used for rotation reference.

**Type**

*Metashape.EulerAngles*

**exportCameras**(*path=""*, *format=CamerasFormatXML*[, *crs* ], *save\_points=True*, *save\_markers=False*, *save\_invalid\_matches=False*, *save\_absolute\_paths=False*, *save\_images=True*, *save\_masks=False*, *use\_labels=False*, *use\_initial\_calibration=False*, *convert\_to\_pinhole=False*, *image\_path='{filename}\_{filenum}.jpg'*, *image\_orientation=0*[, *image\_compression* ], *binary=False*[, *cameras* ], *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'*, *chan\_rotation\_order=RotationOrderXYZ*[, *progress* ])

Export point cloud and/or camera positions.

**Parameters**

- **path** (*str*) – Path to output file.
- **format** (*Metashape.CamerasFormat*) – Export format.
- **crs** (*Metashape.CoordinateSystem*) – Output coordinate system (except AgisoftXML format).
- **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.
- **save\_absolute\_paths** (*bool*) – Save absolute image paths (BlocksExchange and RZML formats only).
- **save\_images** (*bool*) – Enables/disables images export (Colmap format only).
- **save\_masks** (*bool*) – Enables/disables image masks export (Colmap format only).
- **use\_labels** (*bool*) – Enables/disables label based item identifiers.
- **use\_initial\_calibration** (*bool*) – Transform image coordinates to initial calibration.
- **convert\_to\_pinhole** (*bool*) – Transform images to pinhole model without distortions.
- **image\_path** (*str*) – Image name template
- **image\_orientation** (*int*) – Image coordinate system (0 - X right, 1 - X up, 2 - X left, 3 - X down).
- **image\_compression** (*Metashape.ImageCompression*) – Image compression parameters.
- **binary** (*bool*) – Enables/disables binary encoding for selected format (Colmap and FBX formats only).
- **cameras** (*list[int]*) – List of cameras to export (except AgisoftXML format).
- **bundler\_save\_list** (*bool*) – Enables/disables export of Bundler image list file.

- **bundler\_path\_list** (*str*) – 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** (*str*) – Path to BINGO ITERA file.
- **bingo\_path\_image** (*str*) – Path to BINGO IMAGE COORDINATE file.
- **bingo\_path\_geoin** (*str*) – Path to BINGO GEO INPUT file.
- **bingo\_path\_gps** (*str*) – Path to BINGO GPS/IMU file.
- **chan\_rotation\_order** ([Metashape.RotationOrder](#)) – Rotation order (CHAN format only).
- **progress** (*Callable[[float], None]*) – Progress callback.

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

Export markers.

#### Parameters

- **path** (*str*) – Path to output file.
- **crs** ([Metashape.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, *gltf\_y\_up*=True, *comment*="", *save\_comment*=True, *format*=ModelFormatNone[, *crs* ][, *shift* ], *clip\_to\_boundary*=True, *clip\_to\_region*=False, *clip\_to\_block*=False, *block\_margin*=0.5, *save\_metadata\_xml*=False[, *model* ][, *viewpoint* ][, *progress* ])

Export generated model for the chunk.

#### Parameters

- **path** (*str*) – 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** ([Metashape.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** (`Metashape.RasterTransformType`) – Raster band transformation.
- **colors\_rgb\_8bit** (*bool*) – Convert colors to 8 bit RGB.
- **gltf\_y\_up** (*bool*) – Enables/disables y-up axes notation used in glTF.
- **comment** (*str*) – Optional comment (if supported by selected format).
- **save\_comment** (*bool*) – Enables/disables comment export.
- **format** (`Metashape.ModelFormat`) – Export format.
- **crs** (`Metashape.CoordinateSystem`) – Output coordinate system.
- **shift** (`Metashape.Vector`) – Optional shift to be applied to vertex coordinates.
- **clip\_to\_boundary** (*bool*) – Clip model to boundary shapes.
- **clip\_to\_region** (*bool*) – Clip model to chunk region.
- **clip\_to\_block** (*bool*) – Clip model to block region.
- **block\_margin** (*float*) – Block margin (m).
- **save\_metadata\_xml** (*bool*) – Save metadata.xml file.
- **model** (*int*) – Model key to export.
- **viewpoint** (`Metashape.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** (*str*) – Path to output orthophoto.
- **cameras** (`list[int]`) – List of cameras to process.
- **raster\_transform** (`Metashape.RasterTransformType`) – Raster band transformation.
- **projection** (`Metashape.OrthoProjection`) – Output projection.
- **region** (`Metashape.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** ([Metashape.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.

```
exportPointCloud(path="", source_data=PointCloudData[, point_clouds ], binary=True,
    save_point_color=True, save_point_normal=True, save_point_intensity=True,
    save_point_classification=True, save_point_confidence=True,
    save_point_return_number=True, save_point_scan_angle=True,
    save_point_source_id=True, save_point_timestamp=True, save_point_index=True,
    raster_transform=RasterTransformNone, colors_rgb_8bit=True, comment="",
    save_comment=True, format=PointCloudFormatNone,
    image_format=ImageFormatJPEG[, crs ][, shift ][, region ], clip_to_boundary=True,
    clip_to_region=False, block_width=1000, block_height=1000, split_in_blocks=False[,
    classes ], save_images=False, compression=True, tileset_version='1.0',
    screen_space_error=16, folder_depth=5[, viewpoint ], subdivide_task=True,
    no_double_precision=False[, progress ])
```

Export point cloud.

#### Parameters

- **path** (*str*) – Path to output file.
- **source\_data** ([Metashape.DataSource](#)) – Selects between point cloud and tie points. If not specified, uses point cloud if available.
- **point\_clouds** (*list* [*int*]) – Point cloud keys to export.
- **binary** (*bool*) – Enables/disables binary encoding for selected format (if applicable).
- **save\_point\_color** (*bool*) – Enables/disables export of point color.
- **save\_point\_normal** (*bool*) – Enables/disables export of point normal.
- **save\_point\_intensity** (*bool*) – Enables/disables export of point intensity.
- **save\_point\_classification** (*bool*) – Enables/disables export of point classification.
- **save\_point\_confidence** (*bool*) – Enables/disables export of point confidence.
- **save\_point\_return\_number** (*bool*) – Enables/disables export of point return number.
- **save\_point\_scan\_angle** (*bool*) – Enables/disables export of point scan angle.
- **save\_point\_source\_id** (*bool*) – Enables/disables export of point source ID.
- **save\_point\_timestamp** (*bool*) – Enables/disables export of point timestamp.
- **save\_point\_index** (*bool*) – Enables/disables export of point row and column indices.
- **raster\_transform** ([Metashape.RasterTransformType](#)) – Raster band transformation.
- **colors\_rgb\_8bit** (*bool*) – Convert colors to 8 bit RGB.
- **comment** (*str*) – Optional comment (if supported by selected format).

- **save\_comment** (*bool*) – Enable comment export.
- **format** (`Metashape.PointCloudFormat`) – Export format.
- **image\_format** (`Metashape.ImageFormat`) – Image data format.
- **crs** (`Metashape.CoordinateSystem`) – Output coordinate system.
- **shift** (`Metashape.Vector`) – Optional shift to be applied to point coordinates.
- **region** (`Metashape.BBox`) – Region to be exported.
- **clip\_to\_boundary** (*bool*) – Clip point cloud to boundary shapes.
- **clip\_to\_region** (*bool*) – Clip point cloud to chunk region.
- **block\_width** (*float*) – Block width in meters.
- **block\_height** (*float*) – Block height in meters.
- **split\_in\_blocks** (*bool*) – Enable tiled export.
- **classes** (*list[int]*) – List of point classes to be exported.
- **save\_images** (*bool*) – Enable image export.
- **compression** (*bool*) – Enable compression (Cesium format only).
- **tileset\_version** (*str*) – Cesium 3D Tiles format version to export (1.0 or 1.1).
- **screen\_space\_error** (*float*) – Target screen space error (Cesium format only).
- **folder\_depth** (*int*) – Tileset subdivision depth (Cesium format only).
- **viewpoint** (`Metashape.Viewpoint`) – Default view.
- **subdivide\_task** (*bool*) – Enable fine-level task subdivision.
- **no\_double\_precision** (*bool*) – Use single precision float coordinates for binary ply format.
- **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, global_profile=False,
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[, asset ], north_up=True, tile_width=0, tile_height=0[,
progress ])
```

Export DEM or orthomosaic to file.

#### Parameters

- **path** (*str*) – Path to output orthomosaic.
- **format** (`Metashape.RasterFormat`) – Export format.
- **image\_format** (`Metashape.ImageFormat`) – Tile format.
- **raster\_transform** (`Metashape.RasterTransformType`) – Raster band transformation.
- **projection** (`Metashape.OrthoProjection`) – Output projection.

- **region** (`Metashape.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** (`Metashape.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** (`str`) – Optional description to be added to image files.
- **image\_compression** (`Metashape.ImageCompression`) – Image compression parameters.
- **network\_links** (`bool`) – Enable network links generation for KMZ format.
- **global\_profile** (`bool`) – Use global profile (GeoPackage and TMS formats only).
- **min\_zoom\_level** (`int`) – Minimum zoom level (GeoPackage, Google Map Tiles, MBTiles and World Wind Tiles formats only).
- **max\_zoom\_level** (`int`) – Maximum zoom level (GeoPackage, 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** (`str`) – Export title.
- **description** (`str`) – Export description.
- **source\_data** (`Metashape.DataSource`) – Selects between DEM and orthomosaic.
- **asset** (`int`) – Asset key to export.
- **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`, `save_rotation=True`, `save_location_accuracy=False`, `save_rotation_accuracy=False`, `save_errors=True`, `save_estimated=True`, `save_variance=False`, `save_enabled=False`, `progress`])

Export reference data to the specified file.

#### Parameters

- **path** (*str*) – Path to the output file.
- **format** (`Metashape.ReferenceFormat`) – Export format.
- **items** (`Metashape.ReferenceItems`) – Items to export (CSV format only).
- **columns** (*str*) – 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** (*str*) – Column delimiter (CSV format only).
- **precision** (*int*) – Number of digits after the decimal point (CSV format only).
- **save\_rotation** (*bool*) – Save rotation angles (CSV format only).
- **save\_location\_accuracy** (*bool*) – Save location accuracy (CSV format only).
- **save\_rotation\_accuracy** (*bool*) – Save rotation accuracy (CSV format only).
- **save\_errors** (*bool*) – Save errors (CSV format only).
- **save\_estimated** (*bool*) – Save estimated values (CSV format only).
- **save\_variance** (*bool*) – Save variance (CSV format only).
- **save\_enabled** (*bool*) – Save enabled flag (CSV format only).
- **progress** (`Callable[[float], None]`) – Progress callback.

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

Export processing report in PDF format.

#### Parameters

- **path** (*str*) – Path to output report.
- **title** (*str*) – Report title.
- **description** (*str*) – Report description.
- **font\_size** (*int*) – Font size (pt).
- **page\_numbers** (*bool*) – Enable page numbers.
- **save\_system\_info** (*bool*) – Include system information.
- **save\_lidar\_separation** (*bool*) – Include lidar swath separation image.
- **user\_settings** (`list[tuple[str, str]]`) – A list of user defined settings to include on the Processing Parameters page.
- **logo\_path** (*str*) – Path to company logo file.
- **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, save_elevation=True[, progress ])
```

Export shapes layer to file.

#### Parameters

- **path** (*str*) – Path to shape file.
- **save\_points** (*bool*) – Export points.
- **save\_polylines** (*bool*) – Export polylines.
- **save\_polygons** (*bool*) – Export polygons.
- **groups** (*list[int]*) – A list of shape groups to export.
- **format** ([Metashape.ShapesFormat](#)) – Export format.
- **crs** ([Metashape.CoordinateSystem](#)) – Output coordinate system.
- **shift** ([Metashape.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.
- **save\_elevation** (*bool*) – Export elevation values for 3D shapes.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
exportTexture(path="", texture_type=DiffuseMap, raster_transform=RasterTransformNone,
              save_alpha=False[, progress ])
```

Export model texture to file.

#### Parameters

- **path** (*str*) – Path to output file.
- **texture\_type** ([Metashape.Model.TextureType](#)) – Texture type.
- **raster\_transform** ([Metashape.RasterTransformType](#)) – Raster band transformation.
- **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, clip_to_region=False[,
                  tiled_model ], model_compression=True, zip_compression=True, tileset_version='1.0',
                  use_tileset_transform=True, screen_space_error=16, folder_depth=5[, model_group ],
                  pixel_size=0, tile_size=256, face_count=20000[, progress ])
```

Export generated tiled model for the chunk.

#### Parameters

- **path** (*str*) – Path to output model.
- **format** ([Metashape.TiledModelFormat](#)) – Export format.
- **model\_format** ([Metashape.ModelFormat](#)) – Model format for zip export.

- **texture\_format** (`Metashape.ImageFormat`) – Texture format.
- **raster\_transform** (`Metashape.RasterTransformType`) – Raster band transformation.
- **image\_compression** (`Metashape.ImageCompression`) – Image compression parameters.
- **crs** (`Metashape.CoordinateSystem`) – Output coordinate system.
- **clip\_to\_boundary** (`bool`) – Clip tiled model to boundary shapes.
- **clip\_to\_region** (`bool`) – Clip tiled model to chunk region.
- **tiled\_model** (`int`) – Tiled model key to export.
- **model\_compression** (`bool`) – Enable mesh compression (Cesium format only).
- **zip\_compression** (`bool`) – Enable zip compression (Cesium format only).
- **tileset\_version** (`str`) – Cesium 3D Tiles format version to export (1.0 or 1.1).
- **use\_tileset\_transform** (`bool`) – Use tileset transform instead of individual tile transforms (Cesium format only).
- **screen\_space\_error** (`float`) – Target screen space error (Cesium format only).
- **folder\_depth** (`int`) – Tileset subdivision depth (Cesium format only).
- **model\_group** (`int`) – Block model key to export.
- **pixel\_size** (`float`) – Target model resolution in meters (block model export only).
- **tile\_size** (`int`) – Size of tiles in pixels (block model export only).
- **face\_count** (`int`) – Number of faces per megapixel of texture resolution (block model export only).
- **progress** (`Callable[[float], None]`) – Progress callback.

**filterPointCloud**(`point_spacing=0`[, `point_cloud`][, `point_clouds`], `clip_to_region=False`, `replace_asset=False`[, `frames`][, `progress`])

Reduce point cloud points number.

#### Parameters

- **point\_spacing** (`float`) – Desired point spacing (m).
- **point\_cloud** (`int`) – Point cloud key to filter.
- **point\_clouds** (`list[int]`) – List of point clouds to filter.
- **clip\_to\_region** (`bool`) – Clip point cloud to chunk region.
- **replace\_asset** (`bool`) – Replace default point cloud with filtered one.
- **frames** (`list[int]`) – List of frames to process.
- **progress** (`Callable[[float], None]`) – Progress callback.

**findCamera**(`key`)

Find camera by its key.

#### Returns

Found camera.

#### Return type

`Metashape.Camera`

**findCameraGroup**(*key*)

Find camera group by its key.

**Returns**

Found camera group.

**Return type**

*Metashape.CameraGroup*

**findCameraTrack**(*key*)

Find camera track by its key.

**Returns**

Found camera track.

**Return type**

*Metashape.CameraTrack*

**findDepthMaps**(*key*)

Find depth maps by its key.

**Returns**

Found depth maps.

**Return type**

*Metashape.DepthMaps*

**findElevation**(*key*)

Find elevation model by its key.

**Returns**

Found elevation model.

**Return type**

*Metashape.Elevation*

**findFrame**(*key*)

Find frame by its key.

**Returns**

Found frame.

**Return type**

*Metashape.Chunk*

**findMarker**(*key*)

Find marker by its key.

**Returns**

Found marker.

**Return type**

*Metashape.Marker*

**findMarkerGroup**(*key*)

Find marker group by its key.

**Returns**

Found marker group.

**Return type**

*Metashape.MarkerGroup*

**findModel**(*key*)

Find model by its key.

**Returns**

Found model.

**Return type**

*Metashape.Model*

**findModelGroup**(*key*)

Find model group by its key.

**Parameters**

**key** (*int*) – Model group key.

**Returns**

Found model group.

**Return type**

*Metashape.ModelGroup*

**findOrthomosaic**(*key*)

Find orthomosaic by its key.

**Returns**

Found orthomosaic.

**Return type**

*Metashape.Orthomosaic*

**findPointCloud**(*key*)

Find point cloud by its key.

**Returns**

Found point cloud.

**Return type**

*Metashape.PointCloud*

**findPointCloudGroup**(*key*)

Find point cloud group by its key.

**Parameters**

**key** (*int*) – Point cloud group key.

**Returns**

Found point cloud group.

**Return type**

*Metashape.PointCloudGroup*

**findScalebar**(*key*)

Find scalebar by its key.

**Returns**

Found scalebar.

**Return type**

*Metashape.Scalebar*

**findScalebarGroup**(*key*)

Find scalebar group by its key.

**Returns**

Found scalebar group.

**Return type**

*Metashape.ScalebarGroup*

**findSensor**(*key*)

Find sensor by its key.

**Returns**

Found sensor.

**Return type**

*Metashape.Sensor*

**findTiledModel**(*key*)

Find tiled model by its key.

**Returns**

Found tiled model.

**Return type**

*Metashape.TiledModel*

**findTrajectory**(*key*)

Find trajectory by its key.

**Returns**

Found trajectory.

**Return type**

*Metashape.Trajectory*

**frame**

Current frame index.

**Type**

int

**frames**

List of frames in the chunk.

**Type**

list[*Metashape.Chunk*]

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

Generate masks for multiple cameras.

**Parameters**

- **path** (*str*) – Mask file name template.
- **masking\_mode** (*Metashape.MaskingMode*) – Mask generation mode.
- **mask\_operation** (*Metashape.MaskOperation*) – Mask operation.
- **tolerance** (*int*) – Background masking tolerance.

- **cameras** (*list[int]*) – Optional list of cameras to be processed.
- **replace\_asset** (*bool*) – Update default set of masks with generated masks.
- **frames** (*list[int]*) – List of frames to process.
- **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** (*Metashape.ClassificationMethod*) – Index values classification method.
- **boundary\_shape\_group** (*int*) – Boundary shape group.
- **breakpoints** (*list[float]*) – Classification breakpoints.
- **rates** (*list[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** (*str*) – Path to the file.
- **format** (*Metashape.CamerasFormat*) – File format.
- **crs** (*Metashape.CoordinateSystem*) – Ground coordinate system.
- **image\_orientation** (*int*) – Image coordinate system (0 - X right, 1 - X up, 2 - X left, 3 - X down).

- **image\_list** (*str*) – Path to image list file (Bundler format only).
- **load\_image\_list** (*bool*) – Enable Bundler image list import.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
importDepthImages(format=PointCloudFormatNone[, filenames ][, color_filenames ], image_path=",  
                  multiplane=False[, progress ])
```

Import images with depth data.

#### Parameters

- **format** (*Metashape.PointCloudFormat*) – Point cloud format.
- **filenames** (*list[str]*) – List of files to import. Either smartphone depth photos or depth images with corresponding color images in *color\_filenames* parameter.
- **color\_filenames** (*list[str]*) – List of corresponding color files, if present.
- **image\_path** (*str*) – Path template to output pre-processed files.
- **multiplane** (*bool*) – Import as a multi-camera system
- **progress** (*Callable[[float], None]*) – Progress callback.

```
importMarkers(path="[, progress ])
```

Import markers.

#### Parameters

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

```
importModel(path=", format=ModelFormatNone[, crs ][, shift ], decode_udim=True,  
              replace_asset=False[, frame_paths ][, progress ])
```

Import model from file.

#### Parameters

- **path** (*str*) – Path to model.
- **format** (*Metashape.ModelFormat*) – Model format.
- **crs** (*Metashape.CoordinateSystem*) – Model coordinate system.
- **shift** (*Metashape.Vector*) – Optional shift to be applied to vertex coordinates.
- **decode\_udim** (*bool*) – Load UDIM texture layout.
- **replace\_asset** (*bool*) – Replace default asset with imported model.
- **frame\_paths** (*list[str]*) – List of model paths to import in each frame of a multiframe chunk.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
importPointCloud(path='',format=PointCloudFormatNone[, crs ][, shift ],precision=0,columns='xyz',
delimter=' ',group_delimiters=False,skip_rows=0,is_laser_scan=False,
replace_asset=False,load_point_color=True,load_point_normal=True,
load_point_intensity=True,load_point_classification=True,
load_point_confidence=True,load_point_return_number=True,
load_point_scan_angle=True,load_point_source_id=True,
load_point_timestamp=True,load_point_index=True,load_images=True,
generate_panorama=True,calculate_normals=True,point_neighbors=28,
scanner_at_origin=False,ignore_scanner_origin=False,ignore_trajectory=False[,
trajectory ][,frame_paths ][,progress ])
```

Import point cloud from file.

### Parameters

- **path** (*str*) – Path to point cloud.
- **format** (`Metashape.PointCloudFormat`) – Point cloud format.
- **crs** (`Metashape.CoordinateSystem`) – Point cloud coordinate system.
- **shift** (`Metashape.Vector`) – Optional shift to be applied to point coordinates.
- **precision** (*float*) – Coordinate precision (m). For default precision use 0.
- **columns** (*str*) – Column order (x/y/z - coordinates, X/Y/Z - normal, r/g/b - color, i - intensity, t - time, c - classification, space - skip column).
- **delimter** (*str*) – CSV delimiter.
- **group\_delimiters** (*bool*) – Combine consecutive delimiters in csv format.
- **skip\_rows** (*int*) – Number of rows to skip.
- **is\_laser\_scan** (*bool*) – Import point clouds as laser scans.
- **replace\_asset** (*bool*) – Replace default asset with imported point cloud.
- **load\_point\_color** (*bool*) – Import point color.
- **load\_point\_normal** (*bool*) – Import point normal.
- **load\_point\_intensity** (*bool*) – Import point intensity.
- **load\_point\_classification** (*bool*) – Import point classification.
- **load\_point\_confidence** (*bool*) – Import point confidence.
- **load\_point\_return\_number** (*bool*) – Import point return number.
- **load\_point\_scan\_angle** (*bool*) – Import point scan angle.
- **load\_point\_source\_id** (*bool*) – Import point source ID.
- **load\_point\_timestamp** (*bool*) – Import point timestamp.
- **load\_point\_index** (*bool*) – Import point row and column indices.
- **load\_images** (*bool*) – Import images embedded in laser scan.
- **generate\_panorama** (*bool*) – Generate panorama from point colors.
- **calculate\_normals** (*bool*) – Calculate point normals.
- **point\_neighbors** (*int*) – Number of point neighbors to use for normal estimation.
- **scanner\_at\_origin** (*bool*) – Use laser scan origin as scanner position for unstructured point clouds.

- **ignore\_scanner\_origin** (*bool*) – Do not use laser scan origin as scanner position for structured point clouds.
- **ignore\_trajectory** (*bool*) – Do not attach trajectory to imported point cloud.
- **trajectory** (*int*) – Trajectory key to attach.
- **frame\_paths** (*list[str]*) – List of point cloud paths to import in each frame of a multiframe chunk.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
importRaster(path="[ crs ], raster_type=ElevationData, nodata_value=-32767, has_nodata_value=False,
              replace_asset=False[, frames ][, progress ])
```

Import DEM or orthomosaic from file.

#### Parameters

- **path** (*str*) – Path to elevation model in GeoTIFF format.
- **crs** (*Metashape.CoordinateSystem*) – Default coordinate system if not specified in GeoTIFF file.
- **raster\_type** (*Metashape.DataSource*) – Type of raster layer to import.
- **nodata\_value** (*float*) – No-data value.
- **has\_nodata\_value** (*bool*) – No-data value valid flag.
- **replace\_asset** (*bool*) – Replace default raster with imported one.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
importReference(path=", format=ReferenceFormatCSV, columns=", delimiter=", group_delimiters=False,
                 skip_rows=0, items=ReferenceItemsAll[, crs ], rotation_angles=EulerAnglesUndefined,
                 create_markers=False, ignore_labels=False, threshold=0.1, load_rotation=True,
                 load_location_accuracy=False, load_rotation_accuracy=False, load_enabled=False,
                 column_enabled=10, column_label=1, column_x=2, column_y=3, column_z=4,
                 column_sx=8, column_sy=8, column_sz=8, column_a=5, column_b=6, column_c=7,
                 column_sa=9, column_sb=9, column_sc=9, shutter_lag=0[, progress ])
```

Import reference data from the specified file.

#### Parameters

- **path** (*str*) – Path to the file with reference data.
- **format** (*Metashape.ReferenceFormat*) – File format.
- **columns** (*str*) – 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** (*str*) – Column delimiter (CSV format only).
- **group\_delimiters** (*bool*) – Combine consecutive delimiters (CSV format only).
- **skip\_rows** (*int*) – Number of rows to skip (CSV format only).
- **items** (*Metashape.ReferenceItems*) – Items to load reference for (CSV format only).
- **crs** (*Metashape.CoordinateSystem*) – Reference data coordinate system (CSV format only).

- **rotation\_angles** (`Metashape.EulerAngles`) – Euler angles triplet used for rotation reference (CSV format only).
- **create\_markers** (`bool`) – Create markers for missing entries (CSV format only).
- **ignore\_labels** (`bool`) – Matches reference data based on coordinates alone (CSV format only).
- **threshold** (`float`) – Error threshold in meters used when `ignore_labels` is set (CSV format only).
- **load\_rotation** (`bool`) – Load rotation angles (CSV format only).
- **load\_location\_accuracy** (`bool`) – Load location accuracy (CSV format only).
- **load\_rotation\_accuracy** (`bool`) – Load rotation accuracy (CSV format only).
- **load\_enabled** (`bool`) – Load enabled flag (CSV format only).
- **column\_enabled** (`int`) – Enabled flag column (CSV format only).
- **column\_label** (`int`) – Label column (CSV format only).
- **column\_x** (`int`) – X coordinate column (CSV format only).
- **column\_y** (`int`) – Y coordinate column (CSV format only).
- **column\_z** (`int`) – Z coordinate column (CSV format only).
- **column\_sx** (`int`) – X coordinate accuracy column (CSV format only).
- **column\_sy** (`int`) – Y coordinate accuracy column (CSV format only).
- **column\_sz** (`int`) – Z coordinate accuracy column (CSV format only).
- **column\_a** (`int`) – 1st rotation angle column (CSV format only).
- **column\_b** (`int`) – 2nd rotation angle column (CSV format only).
- **column\_c** (`int`) – 3rd rotation angle column (CSV format only).
- **column\_sa** (`int`) – 1st rotation angle accuracy column (CSV format only).
- **column\_sb** (`int`) – 2nd rotation angle accuracy column (CSV format only).
- **column\_sc** (`int`) – 3rd rotation angle accuracy column (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** (`str`) – Path to shape file.
- **replace** (`bool`) – Replace current shapes with new data.
- **boundary\_type** (`Metashape.Shape.BoundaryType`) – Boundary type to be applied to imported shapes.
- **format** (`Metashape.ShapesFormat`) – Shapes format.
- **columns** (`str`) – Column order in csv format (n - label, x/y/z - coordinates, d - description, [] - group of multiple values, | - column separator within group).

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

```
importTiledModel(path="[, progress ])
```

Import tiled model from file.

#### Parameters

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

```
importTrajectory(path=", format=TrajectoryFormatNone, columns='xyz', delimiter=' ',  
                 group_delimiters=False, skip_rows=0[, crs ][, shift ], replace_asset=False[, progress  
                 ])
```

Import trajectory from file.

#### Parameters

- **path** (*str*) – Trajectory file path.
- **format** ([Metashape.TrajectoryFormat](#)) – Trajectory format.
- **columns** (*str*) – Column order in csv format (t - time, x/y/z - coordinates, a/b/c - rotation angles, space - skip column).
- **delimiter** (*str*) – 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** ([Metashape.CoordinateSystem](#)) – Point cloud coordinate system.
- **shift** ([Metashape.Vector](#)) – Optional shift to be applied to point coordinates.
- **replace\_asset** (*bool*) – Replace default asset with imported trajectory.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
importVideo(path=", image_path=", frame_step=CustomFrameStep, custom_frame_step=1, time_start=0,  
            time_end=-1[, progress ])
```

Import video frames from file.

#### Parameters

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

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

**key**

Chunk identifier.

**Type**

int

**label**

Chunk label.

**Type**

str

**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** (*str*) – Path to calibration file.
- **cameras** (*list[Metashape.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**

*Metashape.CoordinateSystem*

**marker\_groups**

List of marker groups in the chunk.

**Type**

list[*Metashape.MarkerGroup*]

**marker\_location\_accuracy**

Expected accuracy of marker coordinates in meters.

**Type**

*Metashape.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[*Metashape.Marker*]

**mask\_sets**

List of mask sets for the current frame.

**Type**

list[*Metashape.Masks*]

**masks**

Image masks.

**Type**

*Metashape.Masks*

**matchPhotos**(*downscale=1, downscale\_3d=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\_3d=100000, keypoint\_limit\_depth\_maps=10000, 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, laser\_scans\_vertical\_axis=0, laser\_scans\_use\_initial\_orientation=False, match\_laser\_scans=False, match\_depth\_maps=False[, progress ]*)

Perform image matching for the chunk frame.

**Parameters**

- **downscale** (*int*) – Image alignment accuracy (0 - Highest, 1 - High, 2 - Medium, 4 - Low, 8 - Lowest).
- **downscale\_3d** (*int*) – Laser scan alignment accuracy (1 - Highest, 2 - High, 4 - Medium, 8 - Low, 16 - Lowest).
- **generic\_preselection** (*bool*) – Enable generic preselection.
- **reference\_preselection** (*bool*) – Enable reference preselection.
- **reference\_preselection\_mode** (*Metashape.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\_3d** (*int*) – Key point limit for laser scans.
- **keypoint\_limit\_depth\_maps** (*int*) – Key point limit for depth maps.
- **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[tuple[int, int]]*) – User defined list of camera pairs to match.

- **cameras** (*list[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.
- **laser\_scans\_vertical\_axis** (*int*) – Common laser scans axis.
- **laser\_scans\_use\_initial\_orientation** (*bool*) – Use initial laser scan orientation for keypoint matching.
- **match\_laser\_scans** (*bool*) – Match laser scans using geometric features.
- **match\_depth\_maps** (*bool*) – Match images with laser scans using geometric features.
- **progress** (*Callable[[float], None]*) – Progress callback.

**mergeComponents** (*components[, progress]*)

Merge components.

**Parameters**

- **components** (*list[Metashape.Component]*) – List of components to merge.
- **progress** (*Callable[[float], None]*) – Progress callback.

**meta**

Chunk meta data.

**Type**

*Metashape.MetaData*

**model**

Default model for the current frame.

**Type**

*Metashape.Model*

**model\_group**

Default model group for the current chunk.

**Type**

*Metashape.ModelGroup*

**model\_groups**

List of model groups in the chunk.

**Type**

*list[Metashape.ModelGroup]*

**models**

List of models for the current frame.

**Type**

*list[Metashape.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 tie points / 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 k4 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**

*Metashape.Orthomosaic*

**orthomosaics**

List of orthomosaics for the current frame.

**Type**

list[*Metashape.Orthomosaic*]

**pansharpenOrthomosaic**([*orthomosaic* ], *channels=0*[, *pan\_orthomosaic* ], *pan\_channels=0, clip\_to\_pan\_data=False, replace\_asset=False*[, *frames* ][, *progress* ])

Pansharpen orthomosaic.

**Parameters**

- **orthomosaic** (*int*) – Orthomosaic to pansharpen.
- **channels** (*int*) – Orthomosaic channel mask (boolean flags, e.g. 0b0010 means only 1st channel is used and the rest ignored).

- **pan\_orthomosaic** (*int*) – Detailed orthomosaic.
- **pan\_channels** (*int*) – Detailed orthomosaic channel mask (boolean flags, e.g. 0b0010 means only 1st channel is used and the rest ignored).
- **clip\_to\_pan\_data** (*bool*) – Clip result to high resolution orthomosaic.
- **replace\_asset** (*bool*) – Replace source orthomosaic with pansharpened result.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**point\_cloud**

Default point cloud for the current frame.

**Type**

*Metashape.PointCloud*

**point\_cloud\_groups**

List of point cloud groups in the chunk.

**Type**

*list[Metashape.PointCloudGroup]*

**point\_clouds**

List of point clouds for the current frame.

**Type**

*list[Metashape.PointCloud]*

**primary\_channel**

Primary channel index (-1 for default).

**Type**

*int*

**publishData**(*service=ServiceSketchfab, source\_data=TiePointsData, raster\_transform=RasterTransformNone, save\_point\_color=True, save\_camera\_track=True, title="", description="", tags="", owner="", token="", username="", password="", account="", hostname="", is\_draft=False, is\_private=False, is\_protected=False, tile\_size=256, min\_zoom\_level=-1, max\_zoom\_level=-1[, projection], resolution=0, project\_id="", upload\_images=False[, point\_classes][, image\_compression][, progress]*)

Publish generated data online.

**Parameters**

- **service** (*Metashape.ServiceType*) – Service to upload on.
- **source\_data** (*Metashape.DataSource*) – Asset type to upload.
- **raster\_transform** (*Metashape.RasterTransformType*) – Raster band transformation.
- **save\_point\_color** (*bool*) – Enables/disables export of point colors.
- **save\_camera\_track** (*bool*) – Enables/disables export of camera track.
- **title** (*str*) – Dataset title.
- **description** (*str*) – Dataset description.
- **tags** (*str*) – Dataset tags.
- **owner** (*str*) – Account owner (Cesium and Mapbox services).

- **token** (*str*) – Account token (Cesium, Mapbox, Nira (Key Secret), Picterra, Pointbox and Sketchfab services).
- **username** (*str*) – Account username (4DMapper, Agisoft Cloud and Pointscene services).
- **password** (*str*) – Account password (4DMapper, Agisoft Cloud, Pointscene and Sketchfab services).
- **account** (*str*) – Account name (Nira (Key ID) service).
- **hostname** (*str*) – Service hostname (4DMapper and Nira services).
- **is\_draft** (*bool*) – Mark dataset as draft (Sketchfab service).
- **is\_private** (*bool*) – Set dataset access to private (Pointbox and Sketchfab services).
- **is\_protected** (*bool*) – Set dataset access to protected (Pointbox service).
- **tile\_size** (*int*) – Tile size in pixels.
- **min\_zoom\_level** (*int*) – Minimum zoom level.
- **max\_zoom\_level** (*int*) – Maximum zoom level.
- **projection** ([Metashape.CoordinateSystem](#)) – Output projection.
- **resolution** (*float*) – Output resolution in meters.
- **project\_id** (*str*) – Id of a target project (from Agisoft Cloud project URL).
- **upload\_images** (*bool*) – Attach photos to Nira publication.
- **point\_classes** (*list[int]*) – List of point classes to be exported.
- **image\_compression** ([Metashape.ImageCompression](#)) – Image compression parameters.
- **progress** (*Callable[[float], None]*) – Progress callback.

#### **raster\_transform**

Raster transform.

##### **Type**

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

**refineModel**(*downscale=4, iterations=10, smoothness=0.5*[, *model*], *replace\_asset=False*[, *cameras* ][, *frames* ][, *progress* ])

Refine polygonal model.

#### Parameters

- **downscale** (*int*) – Refinement quality (1 - Ultra high, 2 - High, 4 - Medium, 8 - Low, 16 - Lowest).
- **iterations** (*int*) – Number of refinement iterations.
- **smoothness** (*float*) – Smoothing strength. Should be in range [0, 1].
- **model** (*int*) – Model to process.
- **replace\_asset** (*bool*) – Replace default asset with refined model.
- **cameras** (*list[int]*) – List of cameras to process.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

#### region

Reconstruction volume selection.

#### Type

*Metashape.Region*

#### remove(*items*)

Remove items from the chunk.

#### Parameters

**items** (*list[Metashape.Chunk | Metashape.Sensor | Metashape.Component | Metashape.CameraGroup | Metashape.MarkerGroup | Metashape.ScalebarGroup | Metashape.Camera | Metashape.Marker | Metashape.Scalebar | Metashape.CameraTrack | Metashape.DepthMaps | Metashape.PointCloud | Metashape.PointCloudGroup | Metashape.Model | Metashape.ModelGroup | Metashape.TiledModel | Metashape.Elevation | Metashape.Orthomosaic | Metashape.Trajectory]*) – 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** (*str*) – 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** (*Metashape.Matrix*) – 4x4 viewpoint transformation matrix.
- **point\_size** (*int*) – Point size.
- **progress** (*Callable[[float], None]*) – Progress callback.

**Returns**

Preview image.

**Return type**

*Metashape.Image*

**resetRegion()**

Reset reconstruction volume selector to default position.

**scalebar\_accuracy**

Expected scale bar accuracy in meters.

**Type**

float

**scalebar\_groups**

List of scale bar groups in the chunk.

**Type**

list[*Metashape.ScalebarGroup*]

**scalebars**

List of scale bars in the chunk.

**Type**

list[*Metashape.Scalebar*]

**selected**

Selects/deselects the chunk.

**Type**

bool

**sensors**

List of sensors in the chunk.

**Type**

list[*Metashape.Sensor*]

**shapes**

Shapes for the current frame.

**Type**

*Metashape.Shapes*

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

Smooth model 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.
- **model** (*int*) – Key of model to smooth.
- **replace\_asset** (*bool*) – Replace default asset with smoothed model.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**smoothPointCloud**(*smoothing\_radius=0*[, *point\_cloud* ][, *point\_clouds* ][, *classes* ],  
*apply\_to\_selection=False, clip\_to\_region=False, replace\_asset=False*[, *frames* ][,  
*progress* ])

Smooth point cloud.

#### Parameters

- **smoothing\_radius** (*float*) – Desired smoothing radius (m).
- **point\_cloud** (*int*) – Key of point cloud to smooth.
- **point\_clouds** (*list[int]*) – List of point clouds to smooth.
- **classes** (*list[int]*) – List of point classes to be smoothed.
- **apply\_to\_selection** (*bool*) – Smooth points within selection.
- **clip\_to\_region** (*bool*) – Clip point cloud to chunk region.
- **replace\_asset** (*bool*) – Replace default point cloud with smoothed one.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**sortCameras()**

Sorts cameras by their labels.

**sortMarkers()**

Sorts markers by their labels.

**sortScalebars()**

Sorts scalebars by their labels.

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

Split components.

#### Parameters

- **items** (*list[Metashape.Camera | Metashape.PointCloud]*) – List of items to split.
- **progress** (*Callable[[float], None]*) – Progress callback.

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

Remove excessive tracks from the tie point cloud.

**Parameters**

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

**thumbnails**

Image thumbnails.

**Type**

*Metashape.Thumbnails*

**tie\_points**

Generated tie point cloud.

**Type**

*Metashape.TiePoints*

**tiepoint\_accuracy**

Expected tie point accuracy in pixels.

**Type**

float

**tiled\_model**

Default tiled model for the current frame.

**Type**

*Metashape.TiledModel*

**tiled\_models**

List of tiled models for the current frame.

**Type**

list[*Metashape.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.

**trajectories**

List of trajectories for the current frame.

**Type**

list[*Metashape.Trajectory*]

**trajectory**

Default trajectory for the current frame.

**Type**

*Metashape.Trajectory*

**transform**

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

**Type***Metashape.ChunkTransform*

```
transformRaster(source_data=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 ], clip_to_boundary=True, copy_orthophotos=True,
replace_asset=False[, frames ][, progress ])
```

Transform DEM or orthomosaic.

**Parameters**

- **source\_data** ([Metashape.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** ([Metashape.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** ([Metashape.BBox](#)) – Region to be processed.
- **projection** ([Metashape.OrthoProjection](#)) – Output projection.
- **clip\_to\_boundary** (*bool*) – Clip raster to boundary shapes.
- **copy\_orthophotos** (*bool*) – Copy orthophotos (orthomosaic asset type only).
- **replace\_asset** (*bool*) – Replace default raster with transformed one.
- **frames** (*list[int]*) – List of frames to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

```
triangulateTiePoints(max_error=10, min_image=2[, progress ])
```

Rebuild tie 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**

*Metashape.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**

*Metashape.ChunkTransform*

**matrix**

Transformation matrix.

**Type**

*Metashape.Matrix*

**rotation**

Rotation component.

**Type**

*Metashape.Matrix*

**scale**

Scale component.

**Type**

float

**translation**

Translation component.

**Type**

*Metashape.Vector*

**class Metashape.CirTransform**

CIR calibration matrix.

**calibrate()**

Calibrate CIR matrix based on orthomosaic histogram.

**coeffs**

Color matrix.

**Type**

*Metashape.Matrix*

**copy()**

Return a copy of the object.

**Returns**

A copy of the object.

**Return type**

*Metashape.CirTransform*

**reset()**

Reset CIR calibration matrix.

**class Metashape.ClassificationMethod**

Index values classification method in [EqualIntervalsClassification, JenksNaturalBreaksClassification]

**class Metashape.CloudClient**

CloudClient class provides access to the Agisoft Cloud processing service and allows to create and manage cloud projects.

The following example connects to the service and lists available projects:

```
>>> import Metashape
>>> client = Metashape.CloudClient()
>>> client.username = 'user'
>>> client.password = 'password'
>>> client.projectList()
```

**abortProcessing(*document*)**

Cancel processing.

**Parameters**

**document** (*Metashape.Document*) – Project to cancel.

**client\_id**

Client software id (optional).

**Type**

str

**client\_secret**

Client software secret (optional).

**Type**

str

**downloadProject(*document* [, *progress* ])**

Download project from the cloud.

**Parameters**

- **document** (*Metashape.Document*) – Project to download.
- **progress** (*Callable[[float], None]*) – Progress callback.

**getProcessingStatus(*document*)**

Get processing status.

**Parameters**

**document** (*Metashape.Document*) – Project being processed.

**Returns**

Processing status.

**Return type**

dict

**getProjectList()**

Get list of projects in the cloud.

**Returns**

List of projects.

**Return type**

list

**password**

Cloud account password.

**Type**

str

**processProject**(*document*, *tasks*)

Start processing in the cloud.

**Parameters**

- **document** (*Metashape.Document*) – Project to process.
- **tasks** (*List [Metashape.NetworkTask]*) – List of processing tasks to execute.

**uploadProject**(*document*, *publish=False*[, *progress* ])

Upload project to the cloud.

**Parameters**

- **document** (*Metashape.Document*) – Project to upload.
- **publish** (*bool*) – Publish project for online visualization.
- **progress** (*Callable[[float], None]*) – Progress callback.

**username**

Cloud account username.

**Type**

str

**class Metashape.Component**

Component instance

**chunk**

Chunk the component belongs to.

**Type**

*Metashape.Chunk*

**key**

Component identifier.

**Type**

int

**label**

Component label.

**Type**

str

**partition**

Component partition.

**Type**  
list

**region**

Reconstruction volume selection.

**Type**  
*Metashape.Region*

**transform**

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

**Type**  
*Metashape.ChunkTransform*

**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** (*str*) – Path to geoid file.

**authority**

Authority identifier of the coordinate system.

**Type**  
str

**copy()**

Return a copy of the object.

**Returns**  
A copy of the object.

**Return type**  
*Metashape.CoordinateSystem*

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

Coordinate transformation from source to target coordinate system datum.

**Parameters**

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

**Returns**  
4x4 transformation matrix.

**Return type**  
*Metashape.Matrix*

**geoccs**

Base geocentric coordinate system.

**Type**

*Metashape.CoordinateSystem*

**geogcs**

Base geographic coordinate system.

**Type**

*Metashape.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** (*str*) – WKT definition of coordinate system or authority identifier.

**listBuiltinCRS()**

Returns a list of builtin coordinate systems.

**listGeoids()**

Returns a list of loaded geoids.

**localframe(*point*)**

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

**Parameters**

**point** (*Metashape.Vector*) – Coordinates of the origin in the geocentric coordinates.

**Returns**

Transformation from geocentric coordinates to local coordinates.

**Return type**

*Metashape.Matrix*

**name**

Name of the coordinate system.

**Type**

str

**proj4**

Coordinate system definition in PROJ.4 format.

**Type**

str

**project(*point*)**

Projects point from geocentric coordinates to projected geographic coordinate system.

**Parameters**

**point** (*Metashape.Vector*) – 3D point in geocentric coordinates.

**Returns**

3D point in projected coordinates.

**Return type***Metashape.Vector***setContext**(*document*)

Set geoid lookup context.

**Parameters****document** (*Metashape.Document*) – Document to use for geoid lookup.**towgs84**

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

**Type**

list[float]

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

Transform point coordinates between coordinate systems.

**Parameters**

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

**Returns**

Transformed point coordinates.

**Return type***Metashape.Vector***transformationMatrix**(*point, source, target*)

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

**Parameters**

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

**Returns**

4x4 transformation matrix.

**Return type***Metashape.Matrix***unproject**(*point*)

Unprojects point from projected coordinates to geocentric coordinates.

**Parameters****point** (*Metashape.Vector*) – 3D point in projected coordinate system.**Returns**

3D point in geocentric coordinates.

**Return type***Metashape.Vector*

**wkt**

Coordinate system definition in WKT format.

**Type**

str

**wkt2**

Coordinate system definition in WKT format, version 2.

**Type**

str

**class Metashape.DataSource**

Data source in [TiePointsData, PointCloudData, ModelData, TiledModelData, ElevationData, OrthomosaicData, DepthMapsData, ImagesData, TrajectoryData, LaserScansData, DepthMapsAndLaserScansData, MasksData, BlockModelData]

**class Metashape.DataType**

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

**class Metashape.DepthMap**

Depth map data.

**calibration**

Depth map calibration.

**Type**

*Metashape.Calibration*

**copy()**

Returns a copy of the depth map.

**Returns**

Copy of the depth map.

**Return type**

*Metashape.DepthMap*

**getCalibration(level=0)**

Returns calibration data.

**Parameters**

**level** (*int*) – Level index.

**Returns**

Calibration data.

**Return type**

*Metashape.Calibration*

**image([level])**

Returns image data.

**Parameters**

**level** (*int*) – Level index.

**Returns**

Image data.

**Return type**

*Metashape.Image*

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

**Parameters**

- **calibration** ([Metashape.Calibration](#)) – Calibration data.
- **level** (*int*) – Level index.

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

**Parameters**

- **image** ([Metashape.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**

*Metashape.DepthMaps*

**items**()

List of items.

**key**

Depth maps identifier.

**Type**

int

**keys**()

List of item keys.

**label**

Depth maps label.

**Type**

str

**meta**

Depth maps meta data.

**Type**

*Metashape.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**

*Metashape.Chunk*

**addGeoid(path)**

Add geoid to the document.

**Parameters**

**path** (*str*) – Path to the geoid file.

**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[int]*) – List of chunks to be aligned.
- **reference** (*int*) – Chunk to be used as a reference.
- **method** (*int*) – Alignment method (0 - tie point based, 1 - marker based, 2 - camera based).
- **fit\_scale** (*bool*) – Fit chunk scale during alignment.
- **downscale** (*int*) – Alignment accuracy (0 - Highest, 1 - High, 2 - Medium, 4 - Low, 8 - Lowest).
- **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[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** (*Metashape.Document*) – Document object to be appended.
- **chunks** (*list*[*Metashape.Chunk*]) – List of chunks to append.
- **progress** (*Callable*[[*float*], *None*]) – Progress callback.

**chunk**

Active chunk.

**Type**

*Metashape.Chunk*

**chunks**

List of chunks in the document.

**Type**

*list*[*Metashape.Chunk*]

**clear**()

Clear the contents of the Document object.

**copy**()

Return a copy of the document.

**Returns**

A copy of the document.

**Return type**

*Metashape.Document*

**findChunk**(*key*)

Find chunk by its key.

**Returns**

Found chunk.

**Return type**

*Metashape.Chunk*

**geoids**

List of geoids in the document.

**Type**

*list*[*Metashape.Geoid*]

**mergeChunks**(*copy\_laser\_scans*=*True*, *copy\_masks*=*True*, *copy\_depth\_maps*=*False*, *copy\_point\_clouds*=*False*, *copy\_models*=*False*, *copy\_tiled\_models*=*False*, *copy\_elevations*=*False*, *copy\_orthomosaics*=*False*, *merge\_markers*=*False*, *merge\_tiepoints*=*False*, *merge\_assets*=*False* [, *chunks*] [, *progress* ])

Merge specified set of chunks.

**Parameters**

- **copy\_laser\_scans** (*bool*) – Copy laser scans.
- **copy\_masks** (*bool*) – Copy masks.
- **copy\_depth\_maps** (*bool*) – Copy depth maps.

- **copy\_point\_clouds** (*bool*) – Copy point clouds.
- **copy\_models** (*bool*) – Copy models.
- **copy\_tiled\_models** (*bool*) – Copy tiled models.
- **copy\_elevations** (*bool*) – Copy DEMs.
- **copy\_orthomosaics** (*bool*) – Copy orthomosaics.
- **merge\_markers** (*bool*) – Merge markers.
- **merge\_tiepoints** (*bool*) – Merge tie points.
- **merge\_assets** (*bool*) – Merge default assets.
- **chunks** (*list[int]*) – List of chunks to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

**meta**

Document meta data.

**Type**

*Metashape.MetaData*

**modified**

Modified flag.

**Type**

*bool*

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

Load document from the specified file.

**Parameters**

- **path** (*str*) – Path to the file.
- **read\_only** (*bool*) – Open document in read-only mode.
- **ignore\_lock** (*bool*) – Ignore lock state for project modifications.
- **archive** (*bool*) – Override project format when using non-standard file extension.

**path**

Path to the document file.

**Type**

*str*

**read\_only**

Read only status.

**Type**

*bool*

**remove**(*items*)

Remove a set of items from the document.

**Parameters**

**items** (*list[Metashape.Chunk | Metashape.Geoid]*) – A list of items to be removed.

**save**(*[path]* [*, chunks*] [*, version*], *archive=True*)

Save document to the specified file.

**Parameters**

- **path** (*str*) – Optional path to the file.
- **chunks** (*list* [*Metashape.Chunk*]) – List of chunks to be saved.
- **version** (*str*) – Project version to save.
- **archive** (*bool*) – Override project format when using non-standard file extension.

**sortChunks**()

Sorts chunks by their labels.

**class Metashape.Elevation**

Digital elevation model.

**class Patch**

Elevation patch.

**class InterpolationType**

DEM fill patch interpolation method in [Constant, Plane, IDW, NaturalNeighbour]

**class Type**

DEM patch type in [Fill, Breakline]

**copy**()

Returns a copy of the patch.

**Returns**

Copy of the patch.

**Return type**

*Metashape.Elevation.Patch*

**exclude\_nested\_polygons**

Exclude nested polygons.

**Type**

bool

**fill\_elevation**

Elevation value for Constant interpolation method.

**Type**

float

**idw\_power**

Power parameter for IDW interpolation method.

**Type**

int

**interpolation\_type**

Interpolation method.

**Type**

*Metashape.Elevation.Patch.InterpolationType*

**sample\_edges**

Sample values from polygon edges (ignored for Constant interpolation method).

**Type**

bool

**type**

Patch type.

**Type**

Metashape.Elevation.Patch.PatchType

**class Patches**

A set of elevation patches.

**items()**

List of items.

**keys()**

List of item keys.

**remove(*items*)**

Remove patches from the elevation.

**Parameters**

**items** (*list*[Metashape.Shape] / Metashape.Shape) – A list of items to be removed.

**values()**

List of item values.

**altitude(*point*)**

Return elevation value at the specified point.

**Parameters**

**point** (Metashape.Vector) – Point coordinates in the DEM coordinate system.

**Returns**

Elevation value.

**Return type**

float

**altitudeSlopeAspect(*point*)**

Return elevation, slope and aspect values at the specified point.

**Parameters**

**point** (Metashape.Vector) – Point coordinates in the DEM coordinate system.

**Returns**

Elevation, slope and aspect values.

**Return type**

tuple[float, float, float]

**bottom**

Y coordinate of the bottom side.

**Type**

float

**clear()**

Clears digital elevation model data.

**copy()**

Create a copy of the digital elevation model.

**Returns**

Copy of the digital elevation model.

**Return type***Metashape.Elevation***coverageArea()**

Calculate coverage area of the DEM in m<sup>2</sup>. Only pixels with data are used.

**Returns**

Area covered by the DEM.

**Return type**

float

**crs**

Coordinate system of elevation model.

**Type***Metashape.CoordinateSystem***height**

Elevation model height.

**Type**

int

**key**

Elevation model identifier.

**Type**

int

**label**

Elevation model label.

**Type**

str

**left**

X coordinate of the left side.

**Type**

float

**legend\_range**

Custom elevation legend range.

**Type**

tuple[float, float]

**max**

Maximum elevation value.

**Type**

float

**meta**

Elevation model meta data.

**Type***Metashape.MetaData*

**min**

Minimum elevation value.

**Type**  
float

**modified**

Modified flag.

**Type**  
bool

**palette**

Color palette.

**Type**  
dict

**patches**

Elevation patches.

**Type**  
*Metashape.Elevation.Patches*

**pickPoint** (*origin, target*)

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

**Parameters**

- **origin** (*Metashape.Vector*) – Ray origin in the DEM coordinate system.
- **target** (*Metashape.Vector*) – Point on the ray in the DEM coordinate system.

**Returns**

Coordinates of the intersection point in the DEM coordinate system.

**Return type**  
*Metashape.Vector*

**projection**

Projection of elevation model.

**Type**  
*Metashape.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

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

Apply edits to elevation.

**Parameters**

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

**width**

Elevation model width.

**Type**

int

**class Metashape.EulerAngles**

Euler angles in [EulerAnglesUndefined, 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.Geoid**

Geoid attributes

**authority**

Geoid authority.

**Type**

str

**height**

Geoid height.

**Type**

int

**horz\_crs**

Horizontal coordinate system.

**Type**

*Metashape.CoordinateSystem*

**name**

Geoid name.

**Type**

str

**path**

Path to geoid file.

**Type**

str

**vert\_crs**

Vertical coordinate system.

**Type**

*Metashape.CoordinateSystem*

**width**

Geoid width.

**Type**

int

**class Metashape.Geometry**

Geometry data.

**GeometryCollection**(*geometries*)

Create a GeometryCollection geometry.

**Parameters**

**geometries** (*list*[*Metashape.Geometry*]) – Child geometries.

**Returns**

A GeometryCollection geometry.

**Return type**

*Metashape.Geometry*

**LineString**(*coordinates*)

Create a LineString geometry.

**Parameters**

**coordinates** (*list*[*Metashape.Vector*]) – List of vertex coordinates.

**Returns**

A LineString geometry.

**Return type**

*Metashape.Geometry*

**MultiLineString**(*geometries*)

Create a MultiLineString geometry.

**Parameters**

**geometries** (*list*[*Metashape.Geometry*]) – Child line strings.

**Returns**

A point geometry.

**Return type**

*Metashape.Geometry*

**MultiPoint**(*geometries*)

Create a MultiPoint geometry.

**Parameters**

**geometries** (*list*[*Metashape.Geometry*]) – Child points.

**Returns**

A point geometry.

**Return type**

*Metashape.Geometry*

**MultiPolygon**(*geometries*)

Create a MultiPolygon geometry.

**Parameters**

**geometries** (*list*[*Metashape.Geometry*]) – Child polygons.

**Returns**

A point geometry.

**Return type**

*Metashape.Geometry*

**Point**(*vector*)

Create a Point geometry.

**Parameters**

**vector** (*Metashape.Vector* | *list*[*float*]) – Point coordinates.

**Returns**

A point geometry.

**Return type**

*Metashape.Geometry*

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

Create a Polygon geometry.

**Parameters**

- **exterior\_ring** (*list*[*Metashape.Vector*]) – Point coordinates.
- **interior\_rings** (*list*[*Metashape.Vector*]) – Point coordinates.

**Returns**

A Polygon geometry.

**Return type**

*Metashape.Geometry*

**class Type**

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

**coordinates**

List of vertex coordinates.

**Type**

*list*[*Metashape.Vector*]

**geometries**

List of child geometries.

**Type**

*list*[*Metashape.Geometry*]

**is\_3d**

Is 3D flag.

**Type**

bool

**type**

Geometry type.

**Type**

*Metashape.Geometry.Type*

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

n-channel image

**Parameters**

- **width** (*int*) – image width
- **height** (*int*) – image height
- **channels** (*str*) – color channel layout, e.g. 'RGB', 'RGBA', etc.
- **datatype** (*str*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']

**channels**

Channel mapping for the image.

**Type**

str

**cn**

Number of color channels.

**Type**

int

**convert**(*channels* [, *datatype* ])

Convert image to specified data type and channel layout.

**Parameters**

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

**Returns**

Converted image.

**Return type**

*Metashape.Image*

**copy**()

Return a copy of the image.

**Returns**

copy of the image

**Return type**

*Metashape.Image*

**data\_type**

Data type used to store pixel values.

**Type**

str

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

Create image from byte array.

**Parameters**

- **data** (*str*) – raw image data
- **width** (*int*) – image width
- **height** (*int*) – image height
- **channels** (*str*) – color channel layout, e.g. 'RGB', 'RGBA', etc.
- **datatype** (*str*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']

**Returns**

Created image.

**Return type**

*Metashape.Image*

**gaussianBlur**(*radius*)

Smooth image with a gaussian filter.

**Parameters**

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

**Returns**

Smoothed image.

**Return type**

*Metashape.Image*

**height**

Image height.

**Type**

int

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

Load image from file.

**Parameters**

- **path** (*str*) – path to the image file
- **layer** (*int*) – image layer in case of multipage file
- **datatype** (*str*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']
- **channels** (*str*) – 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**

*Metashape.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**

*Metashape.Image*

**save**(*path*[, *compression* ])

Save image to the file.

**Parameters**

- **path** (*str*) – path to the image file
- **compression** (*Metashape.ImageCompression*) – compression options

**tostring**()

Convert image to byte array.

**Returns**

Raw image data.

**Return type**

str

**undistort**(*calib, center\_principal\_point=True, square\_pixels=True*)

Undistort image using provided calibration.

**Parameters**

- **calib** (*Metashape.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**

*Metashape.Image*

**uniformNoise**(*amplitude*)

Add uniform noise with specified amplitude.

**Parameters**

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

**Returns**

Image with added noise.

**Return type**

*Metashape.Image*

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

Warp image by rotating virtual viewpoint.

**Parameters**

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

**Returns**

warped image

**Return type**

*Metashape.Image*

**width**

Image width.

**Type**

int

**class** `Metashape.ImageCompression`

Image compression parameters.

The following example demonstrates how to export orthomosaic with custom compression parameters:

```
>>> import Metashape
>>> doc = Metashape.app.document
>>> chunk = doc.chunk
>>> compression = Metashape.ImageCompression()
>>> compression.tiff_tiled = True
>>> compression.tiff_overviews = True
>>> compression.tiff_compression = Metashape.ImageCompression.TiffCompressionJPEG
>>> compression.jpeg_quality = 95
>>> chunk.exportRaster('ortho.tif', source_data=Metashape.OrthomosaicData, image_
↳compression=compression)
```

**class** `TiffCompression`

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

**copy()**

Return a copy of the object.

**Returns**

A copy of the object.

**Return type**

*Metashape.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\_planar**

Export TIFF using separate planar configuration.

**Type**

bool

**tiff\_tiled**

Export tiled TIFF.

**Type**

bool

**class Metashape.ImageFormat**

Image format in [ImageFormatNone, ImageFormatJPEG, ImageFormatTIFF, ImageFormatPNG, ImageFormatBMP, ImageFormatEXR, ImageFormatPNM, ImageFormatSGI, ImageFormatCR2, ImageFormatBZ2, ImageFormatSEQ, ImageFormatBIL, ImageFormatASCII, ImageFormatXYZ, ImageFormatARA, ImageFormatTGA, ImageFormatDDS, ImageFormatJP2, ImageFormatWebP, ImageFormatJXL, ImageFormatKTX]

**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** (*str*) – Activation key.

**activateOffline**(*activation\_params*)

Create a request for offline activation.

**Parameters**

**activation\_params** (*str*) – The content of .actparam file.

**Returns**

The activation request which should be saved to .actreq file.

**Return type**

str

**borrowLicense(*seconds*)**

Borrow floating license for the specified number of seconds.

**Parameters****seconds** (*int*) – Borrow duration in seconds.**borrowed**

License borrowed flag.

**Type**

bool

**deactivate()**

Deactivate software online.

**deactivateOffline()**

Create a request for offline deactivation.

**Returns**

The deactivation request which should be saved to .actreq file.

**Return type**

str

**expiration**

License expiration as a Unix timestamp in seconds.

**Type**

int

**floating**

License floating flag.

**Type**

bool

**install(*activation\_response*)**

Install license from the activation response.

**Parameters****activation\_response** (*str*) – The content of .actresp file.**rehostable**

License rehostable flag.

**Type**

bool

**returnLicense()**

Return borrowed license to the license server.

**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 projection data.

**coord**

Point coordinates in pixels.

**Type**

*Metashape.Vector*

**pinned**

Pinned flag.

**Type**

bool

**size**

Projection size.

**Type**

float

**time**

Projection timestamp.

**Type**

float

**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**

*Metashape.Vector*

**enabled**

Enabled flag.

**Type**

bool

**location**

Marker coordinates.

**Type**

*Metashape.Vector*

**class Type**

Marker type in [Regular, Vertex, Fiducial]

**chunk**

Chunk the marker belongs to.

**Type**

*Metashape.Chunk*

**enabled**

Enables/disables the marker.

**Type**

bool

**frames**

Marker frames.

**Type**

list[*Metashape.Marker*]

**group**

Marker group.

**Type**

*Metashape.MarkerGroup*

**key**

Marker identifier.

**Type**

int

**label**

Marker label.

**Type**

str

**meta**

Marker meta data.

**Type**

*Metashape.MetaData*

**position**

Marker position in the current frame.

**Type**

*Metashape.Vector*

**position\_covariance**

Marker position covariance.

**Type**

*Metashape.Matrix*

**projections**

List of marker projections.

**Type**

*Metashape.Marker.Projections*

**reference**

Marker reference data.

**Type**

*Metashape.Marker.Reference*

**selected**

Selects/deselects the marker.

**Type**

bool

**sensor**

Fiducial mark sensor.

**Type**

*Metashape.Sensor*

**type**

Marker type.

**Type**

*Metashape.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.

**key**

Marker group identifier.

**Type**

int

**label**

Marker group label.

**Type**

str

**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**

*Metashape.Mask*

**image()**

Returns image data.

**Returns**

Image data.

**Return type**

*Metashape.Image*

**invert()**

Create inverted copy of the mask.

**Returns**

Inverted copy of the mask.

**Return type**

*Metashape.Mask*

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

Loads mask from file.

**Parameters**

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

**setImage(*image*)****Parameters**

**image** (*Metashape.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, MaskingModeAI]

**class Metashape.Masks**

A set of masks for a chunk frame.

**clear()**

Clears masks data.

**copy()**

Create a copy of the masks.

**Returns**

Copy of the masks.

**Return type***Metashape.Masks***items()**

List of items.

**key**

Masks identifier.

**Type**

int

**keys()**

List of item keys.

**label**

Masks label.

**Type**

str

**meta**

Masks meta data.

**Type***Metashape.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** (*Metashape.Vector* | *list[float]*) – The vector of diagonal entries.**Returns**

A diagonal matrix.

**Return type***Metashape.Matrix*

**Rotation**(*matrix*)

Create a rotation matrix.

**Parameters**

**matrix** (`Metashape.Matrix`) – The 3x3 rotation matrix.

**Returns**

4x4 matrix representing rotation.

**Return type**

*Metashape.Matrix*

**Scale**(*scale*)

Create a scale matrix.

**Parameters**

**scale** (`Metashape.Vector`) – The scale vector.

**Returns**

A matrix representing scale.

**Return type**

*Metashape.Matrix*

**Translation**(*vector*)

Create a translation matrix.

**Parameters**

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

**Returns**

A matrix representing translation.

**Return type**

*Metashape.Matrix*

**col**(*index*)

Returns column of the matrix.

**Returns**

matrix column.

**Return type**

*Metashape.Vector*

**copy**()

Returns a copy of this matrix.

**Returns**

an instance of itself

**Return type**

*Metashape.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**

*Metashape.Matrix*

**mulp(*point*)**

Transforms a point in homogeneous coordinates.

**Parameters**

**point** (*Metashape.Vector*) – The point to be transformed.

**Returns**

transformed point.

**Return type**

*Metashape.Vector*

**mulv(*vector*)**

Transforms vector in homogeneous coordinates.

**Parameters**

**vector** (*Metashape.Vector*) – The vector to be transformed.

**Returns**

transformed vector.

**Return type**

*Metashape.Vector*

**rotation()**

Returns rotation component of the 4x4 matrix.

**Returns**

rotation component

**Return type**

*Metashape.Matrix*

**row(*index*)**

Returns row of the matrix.

**Returns**

matrix row.

**Return type**

*Metashape.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**

tuple[*Metashape.Matrix*, *Metashape.Vector*, *Metashape.Matrix*]

**t()**

Return a new, transposed matrix.

**Returns**

a transposed matrix

**Return type**

*Metashape.Matrix*

**translation()**

Returns translation component of the 4x4 matrix.

**Returns**

translation component

**Return type**

*Metashape.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**

*Metashape.MetaData*

**items()**

List of items.

**keys()**

List of item keys.

**values()**

List of item values.

**class Metashape.Model**

Triangular mesh model instance

**class Criterion**

Model filter criterion in [ComponentSize, PolygonSize, VertexConfidence]

**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[int, int, int]

**vertices**

Vertex indices.

**Type**

tuple[int, int, int]

**class Faces**

Collection of model faces

**resize(count)**

Resize faces list.

**Parameters**

**count** (*int*) – new face count

**class Statistics**

Model 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

**invalid\_vertices**

Number of vertices with NaN coordinates.

**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**

2D vertex coordinates.

**Type**  
*Metashape.Vector*

**class TexVertices**

Collection of model texture vertices

**resize**(*count*)

Resize vertex list.

**Parameters**

**count** (*int*) – new vertex count

**class Texture**

Model texture.

**bands**

List of color bands.

**Type**

list[str]

**data\_type**

Data type used to store color values.

**Type**

*Metashape.DataType*

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

Return texture image.

**Parameters**

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

**Returns**

Texture image.

**Return type**

*Metashape.Image*

**label**

Animation label.

**Type**

str

**meta**

Texture meta data.

**Type**

*Metashape.MetaData*

**model**

Model the texture belongs to.

**Type**

*Metashape.Model*

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

Initialize texture from image data.

**Parameters**

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

**type**

Texture type.

**Type**

*Metashape.Model.TextureType*

**class TextureType**

Texture type in [DiffuseMap, NormalMap, OcclusionMap, DisplacementMap]

**class Vertex**

Vertex of the model

**color**

Vertex color.

**Type**

tuple of numbers

**confidence**

Vertex confidence.

**Type**

float

**coord**

Vertex coordinates.

**Type**

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

**Returns**

Created texture.

**Return type**

*Metashape.Model.Texture*

**area()**

Return area of the model surface.

**Returns**

Model area.

**Return type**

float

**bands**

List of color bands.

**Type**

list[str]

**block\_index**

Model block index.

**Type**  
tuple

**block\_region**

Model block region.

**Type**  
*Metashape.Region*

**cleanModel**(*criterion*, *level*[, *progress* ])

Remove model faces based on specified criterion.

**Parameters**

- **criterion** (*Metashape.Model.Criterion*) – Model filtering criterion.
- **level** (*int*) – Filtering threshold in percents.
- **progress** (*Callable[[float], None]*) – Progress callback.

**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**  
*Metashape.Model*

**cropSelection()**

Crop selected faces and free vertices from the mesh.

**crs**

Reference coordinate system.

**Type**  
*Metashape.CoordinateSystem* | None

**data\_type**

Data type used to store color values.

**Type**  
*Metashape.DataType*

**enabled**

Enables/disables the model.

**Type**  
bool

**faces**

Collection of model faces.

**Type**

*Metashape.Model.Faces*

**fixTopology()**

Remove polygons causing topological problems.

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

Return active texture.

**Parameters**

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

**Returns**

Texture image.

**Return type**

*Metashape.Image*

**group**

Model group.

**Type**

*Metashape.ModelGroup*

**invertSelection()**

Invert selection.

**key**

Model identifier.

**Type**

int

**label**

Model label.

**Type**

str

**loadTexture** (*path*)

Load texture from the specified file.

**Parameters**

**path** (*str*) – Path to the image file.

**meta**

Model meta data.

**Type**

*Metashape.MetaData*

**modified**

Modified flag.

**Type**

bool

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

Return ray intersection with mesh.

**Parameters**

- **origin** (*Metashape.Vector*) – Ray origin.
- **target** (*Metashape.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**

*Metashape.Vector*

**remove**(*items*)

Remove textures from the model.

**Parameters**

**items** (*list*[*Metashape.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.

**removeTextures**()

Remove textures.

**removeUV**()

Remove UV mapping.

**removeVertexColors**()

Remove vertex colors.

**removeVertexConfidence**()

Remove confidence.

**renderDepth**(*transform, calibration, cull\_faces=True, add\_alpha=True*)

Render model depth image for specified viewpoint.

**Parameters**

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

**Returns**

Rendered image.

**Return type**

*Metashape.Image*

**renderImage**(*transform, calibration, cull\_faces=True, add\_alpha=True, raster\_transform=RasterTransformNone, matcap\_image, smooth\_normals=True, color*)

Render model image for specified viewpoint.

#### Parameters

- **transform** (`Metashape.Matrix`) – Camera location.
- **calibration** (`Metashape.Calibration`) – Camera calibration.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.
- **raster\_transform** (`Metashape.RasterTransformType`) – Raster band transformation.
- **matcap\_image** (`Metashape.Image`) – Matcap image used to shade model.
- **smooth\_normals** – Enable normals smoothing.:type smooth\_normals: bool
- **color** (*list[int]:return: Rendered image.*) – Solid view color.

#### Return type

*Metashape.Image*

**renderMask**(*transform, calibration, cull\_faces=True*)

Render model mask image for specified viewpoint.

#### Parameters

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

#### Returns

Rendered image.

#### Return type

*Metashape.Image*

**renderNormalMap**(*transform, calibration, cull\_faces=True, add\_alpha=True*)

Render image with model normals for specified viewpoint.

#### Parameters

- **transform** (`Metashape.Matrix`) – Camera location.
- **calibration** (`Metashape.Calibration`) – Camera calibration.
- **cull\_faces** (*bool*) – Enable back-face culling.
- **add\_alpha** (*bool*) – Generate image with alpha channel.
- **smooth\_normals** – Enable normals smoothing.:type smooth\_normals: bool

#### Returns

Rendered image.

#### Return type

*Metashape.Image*

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

Generate model preview image.

**Parameters**

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

**Returns**

Preview image.

**Return type**

*Metashape.Image*

**saveTexture**(*path*)

Save texture to the specified file.

**Parameters**

**path** (*str*) – Path to the image file.

**selected**

Selects/deselects the model.

**Type**

bool

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

Set active texture.

**Parameters**

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

**setVertexColors**(*channels*='RGB', *datatype*='U8')

Clear vertex colors data and set layout.

**Parameters**

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

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

Return model statistics.

**Parameters**

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

**Returns**

Model statistics.

**Return type**

*Metashape.Model.Statistics*

**tex\_vertices**

Collection of model texture vertices.

**Type***Metashape.Model.TexVertices***textures**

List of model textures.

**Type**list[*Metashape.Model.Texture*]**transform**

4x4 model transformation matrix.

**Type***Metashape.Matrix***transformVertices**(*transform*)

Transform vertex coordinates.

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

Collection of model vertices.

**Type***Metashape.Model.Vertices***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, ModelFormat-TLS, ModelFormatABC, ModelFormatOSGB, ModelFormatOSGT, ModelFormatGLTF, ModelFormatX3D, ModelFormatLandXML]

**class Metashape.ModelGroup**

ModelGroup objects define groups of multiple models. The grouping is established by assignment of a ModelGroup instance to the Model.group attribute of participating models.

**key**

Model group identifier.

**Type**

int

**label**

Model group label.

**Type**

str

**meta**

Model group meta data.

**Type**

*Metashape.MetaData*

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

Generate block model preview image.

**Parameters**

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

**Returns**

Preview image.

**Return type**

*Metashape.Image*

**selected**

Current selection state.

**Type**

bool

**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.

**abortWorker**(*worker\_id*)

Abort worker.

**Parameters**

**worker\_id** (*int*) – Worker id.

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

Get batch information.

**Parameters**

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

**Returns**

Batch information.

**Return type**

dict

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

Get list of batches.

**Parameters**

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

**Returns**

List of batches.

**Return type**

dict

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

Connect to the server.

**Parameters**

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

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

Create new batch.

**Parameters**

- **path** (*str*) – Project path relative to root folder.
- **tasks** (*list* [*Metashape.NetworkTask*]) – List of processing tasks to execute.
- **meta** (*Metashape.MetaData*) – Batch metadata.

**Returns**

Batch id.

**Return type**

int

**disconnect**()

Disconnect from the server.

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

Export current state of batches.

**Parameters**

**batch\_ids** (*list* [*int*]) – List of batch ids to export.

**Returns**

Batches data.

**Return type**

str

**findBatch**(*path*)

Get batch id based on project path.

**Parameters**

**path** (*str*) – Project path relative to root folder.

**Returns**

Batch id.

**Return type**

int

**importBatches**(*data*)

Import batches from exported data.

**Parameters**

**data** (*str*) – Batches data.

**quitWorker**(*worker\_id*)

Quit worker.

**Parameters**

**worker\_id** (*int*) – Worker id.

**serverInfo**(*revision=0*)

Get server information.

**Parameters**

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

**Returns**

Server information.

**Return type**

dict

**serverVersion**()

Get server version.

**Returns**

Server version.

**Return type**

dict

**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).

**setBatchWorkerLimit**(*batch\_id, worker\_limit*)

Set worker limit of the batch.

**Parameters**

- **batch\_id** (*int*) – Batch id.

- **worker\_limit** (*int*) – Worker limit of the batch (0 - unlimited).

**setWorkerCapability**(*worker\_id, capability*)

Set worker capability.

**Parameters**

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

**setWorkerCpuEnabled**(*worker\_id, cpu\_enabled*)

Set worker CPU enabled flag.

**Parameters**

- **worker\_id** (*int*) – Worker id.
- **cpu\_enabled** (*bool*) – CPU enabled flag.

**setWorkerGpuMask**(*worker\_id, gpu\_mask*)

Set worker GPU mask.

**Parameters**

- **worker\_id** (*int*) – Worker id.
- **gpu\_mask** (*int*) – GPU device mask.

**setWorkerPaused**(*worker\_id, paused=True*)

Set worker paused state.

**Parameters**

- **worker\_id** (*int*) – Worker id.
- **paused** (*bool*) – Paused state.

**setWorkerPriority**(*worker\_id, priority*)

Set worker priority.

**Parameters**

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

**workerInfo**(*worker\_id, revision=0*)

Get worker information.

**Parameters**

- **worker\_id** (*int*) – Worker id.
- **revision** (*int*) – First revision to get.

**Returns**

Worker information.

**Return type**

dict

**workerList**(*revision=0*)

Get list of workers.

**Parameters**

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

**Returns**

List of workers.

**Return type**

dict

**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
>>> doc = Metashape.app.document
>>> match_photos = Metashape.Tasks.MatchPhotos()
>>> match_photos.keypoint_limit = 40000
>>> tasks = []
>>> tasks.append(match_photos.toNetworkTask(doc.chunk))
>>> client = Metashape.NetworkClient()
>>> client.connect('127.0.0.1')
>>> batch_id = client.createBatch(doc.path, tasks)
>>> client.setBatchPaused(batch_id, false)
```

**chunks**

List of chunks.

**Type**

list

**frames**

List of frames.

**Type**

list

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**params**

Task parameters.

**Type**

dict

**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**

*Metashape.OrthoProjection*

**crs**

Base coordinate system.

**Type**

*Metashape.CoordinateSystem*

**interior**

Interior projection flag (Cylindrical projection only).

**Type**

bool

**matrix**

Ortho transformation matrix.

**Type**

*Metashape.Matrix*

**radius**

Cylindrical projection radius.

**Type**

float

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

Transform point coordinates between coordinate systems.

**Parameters**

- **point** (*Metashape.Vector*) – 2D or 3D point coordinates.
- **source** (*Metashape.OrthoProjection* / *Metashape.CoordinateSystem*) – Source coordinate system.
- **target** (*Metashape.OrthoProjection* / *Metashape.CoordinateSystem*) – Target coordinate system.

**Returns**

Transformed point coordinates.

**Return type**

*Metashape.Vector*

**type**

Projection type.

**Type**

*Metashape.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

*Metashape.Orthomosaic.Patch*

#### excluded

Excluded flag.

#### Type

bool

#### image\_keys

Image keys.

#### Type

list[int]

#### inpaint

Inpaint flag.

#### Type

bool

### 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[str]

**bottom**

Y coordinate of the bottom side.

**Type**

float

**camera(*point*)**

Get camera used in orthomosaic at the specified point.

**Parameters****point** (*Metashape.Vector*) – Point coordinates in the orthomosaic coordinate system.**Returns**

Camera used in orthomosaic.

**Return type***Metashape.Camera***clear()**

Clears orthomosaic data.

**copy()**

Create a copy of the orthomosaic.

**Returns**

Copy of the orthomosaic.

**Return type***Metashape.Orthomosaic***coverageArea()**Calculate coverage area of the orthomosaic in m<sup>2</sup>. Only pixels with data are used.**Returns**

Area covered by the orthomosaic.

**Return type**

float

**crs**

Coordinate system of orthomosaic.

**Type***Metashape.CoordinateSystem***data\_type**

Data type used to store color values.

**Type***Metashape.DataType***height**

Orthomosaic height.

**Type**

int

**key**

Orthomosaic identifier.

**Type**  
int

**label**

Orthomosaic label.

**Type**  
str

**left**

X coordinate of the left side.

**Type**  
float

**meta**

Orthomosaic meta data.

**Type**  
*Metashape.MetaData*

**modified**

Modified flag.

**Type**  
bool

**patches**

Orthomosaic patches.

**Type**  
*Metashape.Orthomosaic.Patches*

**projection**

Orthomosaic projection.

**Type**  
*Metashape.OrthoProjection*

**removeOrthophotos()**

Remove orthorectified images from orthomosaic.

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

Generate orthomosaic preview image.

**Parameters**

- **width** (*int*) – Preview image width.
- **height** (*int*) – Preview image height.
- **progress** (*Callable[[float], None]*) – Progress callback.

**Returns**

Preview image.

**Return type**

*Metashape.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**

*Metashape.Image*

**copy**()

Returns a copy of the photo.

**Returns**

Copy of the photo.

**Return type**

*Metashape.Photo*

**image**([*channels* ][, *datatype* ])

Returns image data.

**Parameters**

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

**Returns**

Image data.

**Return type**

*Metashape.Image*

**imageMeta()**

Returns image meta data.

**Returns**

Image meta data.

**Return type**

*Metashape.MetaData*

**layer**

Layer index in the image file.

**Type**

int

**meta**

Photo meta data.

**Type**

*Metashape.MetaData*

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

Loads specified image file.

**Parameters**

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

**path**

Path to the image file.

**Type**

str

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

Creates new thumbnail with specified dimensions.

**Returns**

Thumbnail data.

**Return type**

*Metashape.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**

Point cloud data.

**class Criterion**

Point cloud filter criterion in [ScanAngle, Confidence]

**class Point**

Point of the point cloud

**classification**

Point classification.

**Type**

int

**color**

Point color.

**Type**

tuple[int | float, ...]

**column\_index**

Point column index.

**Type**

int

**confidence**

Point confidence.

**Type**

int

**intensity**

Point intensity.

**Type**

int

**normal**

Point normal.

**Type**

*Metashape.Vector*

**position**

Point coordinates.

**Type**

*Metashape.Vector*

**return\_count**

Point return count.

**Type**

int

**return\_index**

Point return index.

**Type**

int

**row\_index**

Point row index.

**Type**

int

**scan\_angle**

Point scan angle.

**Type**

float

**source\_id**

Point source id.

**Type**

int

**timestamp**

Point timestamp.

**Type**

float

**class Points**

List of point cloud points

**class Reader**

Point cloud reader.

**column\_count**

Column count.

**Type**

int

**has\_point\_classification**

Has point classification.

**Type**

bool

**has\_point\_color**

Has point color.

**Type**

bool

**has\_point\_confidence**

Has point confidence.

**Type**

bool

**has\_point\_index**

Has point row and column indices.

**Type**

bool

**has\_point\_intensity**

Has point intensity.

**Type**

bool

**has\_point\_normal**

Has point normal.

**Type**

bool

**has\_point\_return\_number**

Has point return number.

**Type**

bool

**has\_point\_scan\_angle**

Has point scan angle.

**Type**

bool

**has\_point\_source\_id**

Has point source id.

**Type**

bool

**has\_point\_timestamp**

Has point timestamp.

**Type**

bool

**open(*point\_cloud*)**

Open point cloud for reading.

**read(*count*)**

Read specified number of points from the point cloud starting from the current position.

**Parameters**

**count** (*int*) – Number of points to read.

**Returns**

Points data.

**Return type**

*Metashape.PointCloud.Points*

**reset()**

Reset reading position to the first point in the cloud.

**row\_count**

Row count.

**Type**

int

**alignNormals(*direction*[, *point\_classes*][, *progress*])**

Align selected point normals with specified direction.

**Parameters**

- **direction** (*Metashape.Vector*) – Normal direction in the chunk coordinate system.
- **point\_classes** (*Metashape.PointClass* | *list[Metashape.PointClass]*) – Classes of points to process.
- **progress** (*Callable[[float], None]*) – Progress callback.

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

Assign class to points.

**Parameters**

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

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

**bands**

List of color bands.

**Type**

`list[str]`

**classifyGroundPoints**(`max_angle=10.0`, `max_distance=1.0`, `max_terrain_slope=10.0`, `cell_size=50.0`, `erosion_radius=0.0`[, `source_class` ][, `return_number` ], `keep_existing=False`[, `progress` ])

Classify points into ground and non ground classes.

**Parameters**

- **max\_angle** (`float`) – Maximum angle (degrees).
- **max\_distance** (`float`) – Maximum distance (meters).
- **max\_terrain\_slope** (`float`) – Maximum terrain slope angle (degrees).
- **cell\_size** (`float`) – Cell size (meters).
- **erosion\_radius** (`float`) – Erosion radius (meters).
- **source\_class** (`Metashape.PointClass`) – Class of points to be re-classified.
- **return\_number** (`int`) – Point return number to use (0 - any return, 1 - first return, -1 - last return).
- **keep\_existing** (`bool`) – Keep existing ground points.
- **progress** (`Callable[[float], None]`) – Progress callback.

**classifyOverlapPoints**([`progress` ])

Classify overlap points.

**Parameters**

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

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

Multiclass classification of points.

**Parameters**

- **source** (`Metashape.PointClass`) – Class of points to be re-classified.
- **target** (`list[Metashape.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 point cloud data.

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

Permanently removes deleted points from point cloud.

**Parameters**

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

**component**

Point cloud component.

**Type**

*Metashape.Component*

**copy()**

Create a copy of the point cloud.

**Returns**

Copy of the point cloud.

**Return type**

*Metashape.PointCloud*

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

Crop selected points.

**Parameters**

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

**crs**

Reference coordinate system.

**Type**

*Metashape.CoordinateSystem* | None

**data\_type**

Data type used to store color values.

**Type**

*Metashape.DataType*

**enabled**

Enables/disables the point cloud.

**Type**

bool

**extent**([*transform*])

Get point cloud extent.

**Parameters**

- **transform** (*Metashape.Matrix*) – Optional transformation to apply to point coordinates.

**Returns**

Point cloud extent.

**Return type**

*Metashape.BBox*

**generatePanorama**(*ignore\_colors=False*[, *progress*])

Generate panorama for structured point cloud.

**Parameters**

- **ignore\_colors** (*bool*) – Use intensity information instead of point colors.
- **progress** (*Callable[[float], None]*) – Progress callback.

**group**

Point cloud group.

**Type**

*Metashape.PointCloudGroup*

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

Invert selected point normals.

**Parameters**

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

**invertSelection()**

Invert selection.

**is\_laser\_scan**

Use point cloud as laser scan.

**Type**

*bool*

**key**

Point cloud identifier.

**Type**

*int*

**label**

Point cloud label.

**Type**

*str*

**meta**

Point cloud meta data.

**Type**

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

**Returns**

Coordinates of the intersection point.

**Return type**

`Metashape.Vector`

**point\_count**

Number of points in point cloud.

**Type**

`int`

**point\_count\_by\_class**

Number of points in each class.

**Type**

`dict`

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

Remove points.

**Parameters**

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

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

Remove selected points.

**Parameters**

- **point\_classes** (`Metashape.PointClass` | `list[Metashape.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 point cloud depth image for specified viewpoint.

**Parameters**

- **transform** (`Metashape.Matrix`) – Camera location.
- **calibration** (`Metashape.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**

*Metashape.Image*

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

Render point cloud image for specified viewpoint.

**Parameters**

- **transform** (*Metashape.Matrix*) – Camera location.
- **calibration** (*Metashape.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** (*Metashape.RasterTransformType*) – Raster band transformation.

**Returns**

Rendered image.

**Return type**

*Metashape.Image*

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

Render point cloud mask image for specified viewpoint.

**Parameters**

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

*Metashape.Image*

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

Render image with point cloud normals for specified viewpoint.

**Parameters**

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

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

**Returns**

Preview image.

**Return type**

*Metashape.Image*

**resetFilters**()

Reset filters.

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

Restore deleted points.

**Parameters**

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

**selectMaskedPoints**(*cameras, softness=4, only\_visible=False*[, *progress* ])

Select points based on image masks.

**Parameters**

- **cameras** (*list[Metashape.Camera]*) – A list of cameras to use for selection.
- **softness** (*float*) – Mask edge softness.
- **only\_visible** (*bool*) – Select visible points only.
- **progress** (*Callable[[float], None]*) – Progress callback.

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

Select points based on point colors.

**Parameters**

- **color** (*list[int]*) – Color to select.
- **tolerance** (*int*) – Color tolerance.

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

**selectPointsByHeight**(*min\_height*, *max\_height*, *reference='absolute'*, [*progress*])

Select points based on height.

**Parameters**

- **min\_height** (*float*) – Minimum height (m).
- **max\_height** (*float*) – Maximum height (m).
- **reference** (*str*) – Elevation reference in ['absolute', 'ground', 'dem'].
- **progress** (*Callable[[float], None]*) – Progress callback.

**selectPointsByRegion**(*region*)

Select points inside box region.

**Parameters**

**region** (*Metashape.Region*) – Box region to select in chunk coordinates

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

Select points based on shapes.

**Parameters**

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

**selected**

Selects/deselects the point cloud.

**Type**

bool

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

Set filter by point classes.

**Parameters**

**point\_classes** (*Metashape.PointClass* | *list[Metashape.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.

**setPointReturnsFilter**(*first\_return=True*, *middle\_return=True*, *last\_return=True*, *single\_return=True*)

Set filter by point return.

**Parameters**

- **first\_return** (*bool*) – First return.
- **middle\_return** (*bool*) – Intermediate return.

- **last\_return** (*bool*) – Last return.
- **single\_return** (*bool*) – Single return.

**setSelectionFilter()**

Set filter by selection.

**transform**

4x4 point cloud transformation matrix.

**Type**

*Metashape.Matrix*

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

Update point cloud statistics.

**Parameters**

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

**class Metashape.PointCloudFormat**

Point cloud format in [PointCloudFormatNone, PointCloudFormatOBJ, PointCloudFormatPLY, PointCloudFormatXYZ, PointCloudFormatLAS, PointCloudFormatExpe, PointCloudFormatU3D, PointCloudFormatPDF, PointCloudFormatE57, PointCloudFormatOC3, PointCloudFormatPotree, PointCloudFormatLAZ, PointCloudFormatCL3, PointCloudFormatPTS, PointCloudFormatPTX, PointCloudFormatDXF, PointCloudFormatCesium, PointCloudFormatPCD, PointCloudFormatSLPK, PointCloudFormatCOPC, PointCloudFormatCSV]

**class Metashape.PointCloudGroup**

PointCloudGroup objects define groups of multiple laser scans. The grouping is established by assignment of a PointCloudGroup instance to the PointCloud.group attribute of participating laser scans.

**crs**

Reference coordinate system.

**Type**

*Metashape.CoordinateSystem* | None

**fixed**

Fix relative laser scan positions within the group.

**Type**

bool

**key**

Asset group identifier.

**Type**

int

**label**

Point cloud group label.

**Type**

str

**meta**

Point cloud group meta data.

**Type**

*Metashape.MetaData*

**selected**

Current selection state.

**Type**

bool

**transform**

4x4 asset group transformation matrix.

**Type**

*Metashape.Matrix*

**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**

*Metashape.RPCModel*

**error(*point*, *proj*)**

Returns projection error.

**Parameters**

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

**Returns**

2D projection error.

**Return type**

*Metashape.Vector*

**image\_offset**

Image coordinate offset.

**Type**

*Metashape.Vector*

**image\_scale**

Image coordinate scale.

**Type**

*Metashape.Vector*

**line\_den\_coeff**

Line denominator.

**Type**

*Metashape.Vector*

**line\_num\_coeff**

Line numerator.

**Type**

*Metashape.Vector*

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

Load RPC model from file.

**Parameters**

- **path** (*str*) – Path to RPC model file.
- **format** (*str*) – RPC model file format in ['rpc', 'rpb', 'dimap']. Tiled DIMAP files are not supported.

**object\_offset**

Object coordinate offset.

**Type**

*Metashape.Vector*

**object\_scale**

Object coordinate scale.

**Type**

*Metashape.Vector*

**project**(*point*)

Returns projected pixel coordinates of the point.

**Parameters**

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

**Returns**

2D projected point coordinates.

**Return type**

*Metashape.Vector*

**samp\_den\_coeff**

Sample denominator.

**Type**

*Metashape.Vector*

**samp\_num\_coeff**

Sample numerator.

**Type**

*Metashape.Vector*

**save**(*path*[, *format* ])

Save RPC model to file.

**Parameters**

- **path** (*str*) – Path to RPC model file.
- **format** (*str*) – RPC model file format in ['rpc', 'rpb'].

**unproject**(*point*)

Returns direction corresponding to the image point.

**Parameters**

**point** (*Metashape.Vector*) – Pixel coordinates of the point.

**Returns**

3D vector in the camera coordinate system.

**Return type**

*Metashape.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**

*Metashape.RasterTransform*

**enabled**

Enable flag.

**Type**

bool

**false\_color**

False color channels.

**Type**

list

**formula**

Raster calculator expression.

**Type**

str

**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, ReferenceItemsAll]

**class Metashape.ReferencePreselectionMode**

Reference preselection mode in [ReferencePreselectionSource, ReferencePreselectionEstimated, ReferencePreselectionSequential]

**class Metashape.Region**

Region parameters

**center**

Region center coordinates.

**Type**

*Metashape.Vector*

**copy()**

Return a copy of the object.

**Returns**

A copy of the object.

**Return type**

*Metashape.Region*

**rot**

Region rotation matrix.

**Type**

*Metashape.Matrix*

**size**

Region size.

**Type**

*Metashape.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**

*Metashape.Chunk*

**frames**

Scale bar frames.

**Type**

list[*Metashape.Scalebar*]

**group**

Scale bar group.

**Type**

*Metashape.ScalebarGroup*

**key**

Scale bar identifier.

**Type**

int

**label**

Scale bar label.

**Type**

str

**meta**

Scale bar meta data.

**Type**

*Metashape.MetaData*

**point0**

Start of the scale bar.

**Type**

*Metashape.Marker* | *Metashape.Camera*

**point1**

End of the scale bar.

**Type**

*Metashape.Marker* | *Metashape.Camera*

**reference**

Scale bar reference data.

**Type**

*Metashape.Scalebar.Reference*

**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.

**key**

Scale bar group identifier.

**Type**

int

**label**

Scale bar group label.

**Type**

str

**selected**

Current selection state.

**Type**

bool

**class Metashape.Sensor**

Sensor instance

**class Axes**

Local camera axis directions in [Aerial, Terrestrial]

**class FilmTransformType**

Film transformation type in [Conformal, Affine, Projective]

**class Reference**

Sensor reference data.

**accuracy**

Sensor location accuracy.

**Type**

*Metashape.Vector*

**enabled**

Location enabled flag.

**Type**

bool

**location**

Sensor coordinates.

**Type**

*Metashape.Vector*

**location\_accuracy**

Sensor location accuracy.

**Type**

*Metashape.Vector*

**location\_enabled**

Location enabled flag.

**Type**

bool

**rotation**

Sensor rotation angles.

**Type**

*Metashape.Vector*

**rotation\_accuracy**

Sensor rotation accuracy.

**Type**

*Metashape.Vector*

**rotation\_enabled**

Rotation enabled flag.

**Type**

bool

**class Type**

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

**antenna**

GPS antenna correction.

**Type**

*Metashape.Antenna*

**axes**

Local camera coordinate system orientation.

**Type**

*Metashape.Sensor.Axes*

**bands**

List of color bands.

**Type**

list[str]

**black\_level**

Black level for each band.

**Type**

list[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**

*Metashape.Calibration*

**chunk**

Chunk the sensor belongs to.

**Type**

*Metashape.Chunk*

**data\_type**

Data type used to store color values.

**Type**

*Metashape.DataType*

**fiducials**

Fiducial marks.

**Type**

list[*Metashape.Marker*]

**film\_camera**

Film camera flag.

**Type**

bool

**film\_transform\_type**

Film transformation type.

**Type**

*Metashape.Sensor.FilmTransformType*

**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[str]

**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**

str

**layer\_index**

Sensor layer index.

**Type**

int

**location**

Sensor plane location.

**Type**

*Metashape.Vector*

**location\_covariance**

Sensor plane location covariance.

**Type**

*Metashape.Matrix*

**makeMaster()**

Make this sensor master in the multi-camera system.

**master**

Master sensor.

**Type**

*Metashape.Sensor*

**meta**

Sensor meta data.

**Type**

*Metashape.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[str]

**pixel\_height**

Pixel height in mm.

**Type**

float

**pixel\_size**

Pixel size in mm.

**Type**

*Metashape.Vector*

**pixel\_width**

Pixel width in mm.

**Type**

float

**planes**

Sensor planes.

**Type**

list[*Metashape.Sensor*]

**reference**

Sensor reference data.

**Type**

*Metashape.Sensor.Reference*

**rolling\_shutter**

Enable rolling shutter compensation.

**Type**

*Metashape.Shutter.Model*

**rotation**

Sensor plane rotation.

**Type**

*Metashape.Matrix*

**rotation\_covariance**

Sensor plane rotation covariance.

**Type**

*Metashape.Matrix*

**sensitivity**

Sensitivity for each band.

**Type**

list[float]

**type**

Sensor projection model.

**Type**

*Metashape.Sensor.Type*

**user\_calib**

Custom calibration used as initial calibration during photo alignment.

**Type**

*Metashape.Calibration*

**vignetting**

Vignetting for each band.

**Type**

list[*Metashape.Vignetting*]

**width**

Image width.

**Type**

int

**class Metashape.ServiceType**

Service type in [ServiceSketchfab, ServiceMapbox, Service4DMapper, ServiceAgisoftCloud, Service-Pointscene, ServicePointbox, ServicePicterra, ServiceCesium, ServiceNira]

**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

**areaFitted()**

Return 2D area of the shape projected onto the best fitting plane.

**Returns**

Shape area.

**Return type**

float

**attributes**

Shape attributes.

**Type***Metashape.MetaData***boundary\_type**

Shape boundary type.

**Type***Metashape.Shape.BoundaryType***geometry**

Shape geometry.

**Type***Metashape.Geometry* | *Metashape.AttachedGeometry***group**

Shape group.

**Type***Metashape.ShapeGroup***is\_attached**

Attached flag.

**Type**

bool

**key**

Shape identifier.

**Type**

int

**label**

Shape label.

**Type**

str

**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[int, int, int, int]

**enabled**

Enable flag.

**Type**

bool

**key**

Shape group identifier.

**Type**

int

**label**

Shape group label.

**Type**

str

**meta**

Shape group meta data.

**Type**

*Metashape.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**

*Metashape.ShapeGroup*

**addShape()**

Add new shape to the set of shapes.

**Returns**

Created shape.

**Return type**

*Metashape.Shape*

**crs**

Shapes coordinate system.

**Type**

*Metashape.CoordinateSystem*

**group**

Default shape group.

**Type**

*Metashape.ShapeGroup*

**groups**

List of shape groups.

**Type**

list[*Metashape.ShapeGroup*]

**items()**

List of items.

**meta**

Shapes meta data.

**Type**

*Metashape.MetaData*

**modified**

Modified flag.

**Type**

bool

**projection**

Shapes projection.

**Type**

*Metashape.OrthoProjection*

**remove(*items*)**

Remove items from the shape layer.

**Parameters**

**items** (*list*[*Metashape.Shape* / *Metashape.ShapeGroup*]) – A list of items to be removed.

**shapes**

List of shapes.

**Type**

*list*[*Metashape.Shape*]

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

Update altitudes for items.

**Parameters**

- **items** (*list*[*Metashape.Shape* / *Metashape.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.

**class Model**

Rolling shutter model in [*Disabled*, *Regularized*, *Full*]

**copy()**

Return a copy of the object.

**Returns**

A copy of the object.

**Return type**

*Metashape.Shutter*

**rotation**

Rotation matrix of the rolling shutter model.

**Type**

*Metashape.Matrix*

**translation**

Translation vector of the rolling shutter model.

**Type**

*Metashape.Vector*

**class Metashape.SurfaceType**

Surface type in [Arbitrary, HeightField]

**class Metashape.Target**

Target parameters

**code**

Target code.

**Type**

int

**coord**

Target location.

**Type**

*Metashape.Vector*

**copy()**

Return a copy of the object.

**Returns**

A copy of the object.

**Return type**

*Metashape.Target*

**radius**

Target radius.

**Type**

float

**class Metashape.TargetType**

Target type in [CircularTarget12bit, CircularTarget14bit, CircularTarget16bit, CircularTarget20bit, CircularTarget, CrossTarget, AprilTag16h5, AprilTag25h9, AprilTag36h10, AprilTag36h11, AprilTagCircle21h7, AprilTagStandard41h12, AprilTagStandard52h13]

**class Metashape.Tasks**

Task classes.

**class AddFrames**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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\_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\_point\_cloud**

Copy point cloud.

**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[int]

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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. Can be used to create Multiframe / Multiplane layout. In that case list values are number of sequential photos for each camera.

**Type**  
list[int]

**filenames**

List of files to add.

**Type**  
list[str]

**gpu\_support**

GPU support flag.

**Type**  
bool

**group**

Camera group key.

**Type**  
int

**layout**

Image layout.

**Type**  
*Metashape.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 GNSS/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**  
str

**strip\_extensions**

Strip file extensions from camera labels.

**Type**  
bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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

**align\_laser\_scans**

Align laser scans using geometric features.

**Type**

bool

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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[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.

**gpu\_support**

GPU support flag.

**Type**

bool

**min\_image**

Minimum number of point projections.

**Type**

int

**name**

Task name.

**Type**

str

**point\_clouds**

List of point clouds to align.

**Type**

list[int]

**reset\_alignment**

Reset current alignment.

**Type**

bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type**

bool

**target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask([objects])**Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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[int]

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**downscale**

Alignment accuracy (0 - Highest, 1 - High, 2 - Medium, 4 - Low, 8 - Lowest).

**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

**gpu\_support**

GPU support flag.

**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[int]

**mask\_tiepoints**

Apply mask filter to tie points.

**Type**

bool

**method**

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

**Type**  
int

**name**

Task name.

**Type**  
str

**reference**

Chunk to be used as a reference.

**Type**  
int

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**  
int

**class AnalyzeImages**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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[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

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**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**

str

**prevent\_intersections**

Prevent contour intersections.

**Type**

bool

**source\_data**

Source data for contour generation.

**Type***Metashape.DataSource***target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask([objects])**Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.Document*) – Chunk or Document object to be processed.

- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**classes**

List of point classes to be used for surface extraction.

**Type**  
list[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[int]

**gpu\_support**

GPU support flag.

**Type**  
bool

**interpolation**

Interpolation mode.

**Type**  
*Metashape.Interpolation*

**max\_workgroup\_size**

Maximum workgroup size.

**Type**  
int

**name**

Task name.

**Type**  
str

**projection**

Output projection.

**Type**  
*Metashape.OrthoProjection*

**region**

Region to be processed.

**Type**  
*Metashape.BBox*

**replace\_asset**

Replace default asset with generated DEM.

**Type**

bool

**resolution**

Output resolution in meters.

**Type**

float

**source\_data**

Selects between point cloud and tie points.

**Type**

*Metashape.DataSource*

**subdivide\_task**

Enable fine-level task subdivision.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects ])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**

int

**workitem\_size\_tiles**

Number of tiles in a workitem.

**Type**

int

**class BuildDepthMaps**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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[int]

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(*json*)**

Initialize task parameters from a JSON string.

**downscale**

Depth map quality (1 - Ultra high, 2 - High, 4 - Medium, 8 - Low, 16 - Lowest).

**Type**

int

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**filter\_mode**

Depth map filtering mode.

**Type**

*Metashape.FilterMode*

**gpu\_support**

GPU support flag.

**Type**

bool

**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**

str

**reuse\_depth**

Enable reuse depth maps option.

**Type**

bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

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

**blocks\_crs**

Blocks grid coordinate system.

**Type**  
*Metashape.CoordinateSystem*

**blocks\_origin**

Blocks grid origin.

**Type**  
*Metashape.Vector*

**blocks\_size**

Blocks size in coordinate system units.

**Type**  
float

**build\_texture**

Generate preview textures.

**Type**  
bool

**cameras**

List of cameras to process.

**Type**  
*list[int]*

**classes**

List of point classes to be used for surface extraction.

**Type**  
*list[int]*

**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.

**export\_blocks**

Export completed blocks.

**Type**

bool

**face\_count**

Target face count.

**Type***Metashape.FaceCount***face\_count\_custom**

Custom face count.

**Type**

int

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**interpolation**

Interpolation mode.

**Type***Metashape.Interpolation***keep\_depth**

Enable store depth maps option.

**Type**

bool

**max\_workgroup\_size**

Maximum workgroup size.

**Type**

int

**name**

Task name.

**Type**  
str

**output\_folder**

Path to output folder.

**Type**  
str

**replace\_asset**

Replace default asset with generated model.

**Type**  
bool

**source\_data**

Selects between point cloud, tie points, depth maps and laser scans.

**Type**  
*Metashape.DataSource*

**split\_in\_blocks**

Split model in blocks.

**Type**  
bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type**  
bool

**surface\_type**

Type of object to be reconstructed.

**Type**  
*Metashape.SurfaceType*

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

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

**blending\_mode**

Orthophoto blending mode.

**Type***Metashape.BlendingMode***color\_enhancement**

Enable automatic color enhancement.

**Type**

bool

**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

**frames**

List of frames to process.

**Type**

list[int]

**ghosting\_filter**

Enable ghosting filter.

**Type**

bool

**gpu\_support**

GPU support flag.

**Type**

bool

**max\_workgroup\_size**

Maximum workgroup size.

**Type**

int

**name**

Task name.

**Type**

str

**projection**

Output projection.

**Type**

*Metashape.OrthoProjection*

**refine\_seamlines**

Refine seamlines based on image content.

**Type**

bool

**region**

Region to be processed.

**Type**

*Metashape.BBox*

**replace\_asset**

Replace default asset with generated orthomosaic.

**Type**

bool

**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

**surface\_data**

Orthorectification surface.

**Type**

*Metashape.DataSource*

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**transfer\_texture**

Transfer model texture to orthomosaic.

**Type**

bool

**use\_occlusion\_texture**

Use model occlusion texture.

**Type**

bool

**use\_point\_cloud\_intensity**

Use point cloud intensity as color source.

**Type**

bool

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

**blending\_mode**

Panorama blending mode.

**Type**  
*Metashape.BlendingMode*

**camera\_groups**

List of camera groups to process.

**Type**  
list[int]

**color\_enhancement**

Enable automatic color enhancement.

**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 frames to process.

**Type**  
list[int]

**ghosting\_filter**

Enable ghosting filter.

**Type**  
bool

**gpu\_support**

GPU support flag.

**Type**  
bool

**height**

Height of output panorama.

**Type**  
int

**name**

Task name.

**Type**  
str

**region**

Region to be generated.

**Type**  
*Metashape.BBox*

**rotation**

Panorama 3x3 orientation matrix.

**Type**  
*Metashape.Matrix*

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**width**

Width of output panorama.

**Type**  
int

**workitem\_count**

Work item count.

**Type**  
int

**class BuildPointCloud**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**asset**

Asset 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.

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**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**

str

**point\_colors**

Enable point colors calculation.

**Type**

bool

**point\_confidence**

Enable point confidence calculation.

**Type**

bool

**points\_spacing**

Desired point spacing (m).

**Type**

float

**replace\_asset**

Replace default asset with generated point cloud.

**Type**

bool

**source\_data**

Source data to extract points from.

**Type**

*Metashape.DataSource*

**subdivide\_task**

Enable fine-level task subdivision.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**uniform\_sampling**

Enable uniform point sampling.

**Type**

bool

**workitem\_count**

Work item count.

**Type**

int

**workitem\_size\_cameras**

Number of cameras in a workitem.

**Type**

int

**class BuildSeamlines**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**

int

**class BuildTexture**

Task class containing processing parameters.

**anti\_aliasing**

Anti-aliasing coefficient for baking

**Type**

int

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

Apply task to specified object.

**Parameters**

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

**blending\_mode**

Texture blending mode.

**Type**

*Metashape.BlendingMode*

**cameras**

A list of cameras to be used for texturing.

**Type**

list[int]

**color\_enhancement**

Enable automatic color enhancement.

**Type**

bool

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**downscale**

Images downscale (natural blending only).

**Type**

int

**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

**gpu\_support**

GPU support flag.

**Type**

bool

**max\_workgroup\_size**

Maximum workgroup size (block model only).

**Type**

int

**name**

Task name.

**Type**

str

**out\_of\_focus\_filter**

Enable out-of-focus filter (natural blending only).

**Type**

bool

**sharpening**

Sharpening strength (natural blending only).

**Type**

float

**source\_asset**

Source asset.

**Type**

int

**source\_data**

Source data to create texture from.

**Type**

*Metashape.DataSource*

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**texture\_size**

Texture page size.

**Type**

int

**texture\_type**

Texture type.

**Type**

*Metashape.Model.TextureType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**transfer\_texture**

Transfer texture.

**Type**

bool

**use\_assigned\_images**

Use assigned images when blending.

**Type**

bool

**workitem\_count**

Work item count.

**Type**

int

**workitem\_size\_cameras**

Number of cameras in a workitem (block model only).

**Type**  
int

**class BuildTiledModel**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**classes**

List of point classes to be used for surface extraction.

**Type**  
list[int]

**color\_enhancement**

Enable automatic color enhancement.

**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.

**face\_count**

Number of faces per megapixel of texture resolution.

**Type**  
int

**frames**

List of frames to process.

**Type**  
list[int]

**ghosting\_filter**

Enable ghosting filter.

**Type**  
bool

**gpu\_support**

GPU support flag.

**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**

str

**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

**replace\_asset**

Replace default asset with generated tiled model.

**Type**

bool

**source\_data**

Selects between point cloud and mesh.

**Type**

*Metashape.DataSource*

**subdivide\_task**

Enable fine-level task subdivision.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**tile\_size**

Size of tiles in pixels.

**Type**

int

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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 CameraMapping 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.

**gpu\_support**

GPU support flag.

**Type**

bool

**mapping\_mode**

Texture mapping mode.

**Type**

*Metashape.MappingMode*

**name**

Task name.

**Type**

str

**page\_count**

Number of texture pages to generate.

**Type**

int

**pixel\_size**

Texture resolution in meters.

**Type**

float

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**texture\_size**

Expected size of texture page at texture generation step.

**Type**

int

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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.

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**point\_cloud**

Point cloud key to process.

**Type**

int

**point\_clouds**

List of point clouds to process.

**Type**

list[int]

**point\_neighbors**

Number of point neighbors to use for normal estimation.

**Type**

int

**replace\_asset**

Replace source point cloud with processed one.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (`Metashape.Document` / `Metashape.Chunk` / `list[Metashape.Chunk]`) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**  
int

**class CalibrateCamera**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (`Metashape.Chunk` / `Metashape.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

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

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

**cameras**

List of cameras to calibrate.

**Type**

list[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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**source\_data**

Source data for calibration.

**Type**

[Metashape.DataSource](#)

**target**

Task target.

**Type**

[Metashape.Tasks.TargetType](#)

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

Convert task to [Metashape.NetworkTask](#) to be applied to specified objects.

**Parameters**

- **objects** ([Metashape.Document](#) / [Metashape.Chunk](#) / *list*[[Metashape.Chunk](#)]) – Objects to be processed.

**white\_balance**

Calibrate white balance.

**Type**

bool

**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** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask**([*objects* ])Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters**

- **objects** (*Metashape.Document* / *Metashape.Chunk* / *list*[*Metashape.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** (*Metashape.Chunk* / *Metashape.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

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**keep\_existing**

Keep existing ground points.

**Type**

bool

**max\_angle**

Maximum angle (degrees).

**Type**

float

**max\_distance**

Maximum distance (meters).

**Type**

float

**max\_terrain\_slope**

Maximum terrain slope angle (degrees).

**Type**

float

**name**

Task name.

**Type**

str

**point\_cloud**

Point cloud key to classify.

**Type**

int

**point\_clouds**

List of point clouds to classify.

**Type**

list[int]

**replace\_asset**

Replace source point cloud with classified one.

**Type**

bool

**return\_number**

Point return number to use (0 - any return, 1 - first return, -1 - last return).

**Type**

int

**source\_class**

Class of points to be re-classified.

**Type**

int

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**  
int

**class ClassifyOverlapPoints**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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.

**frames**

List of frames to process.

**Type**  
list[int]

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**point\_cloud**

Point cloud key to classify.

**Type**  
int

**point\_clouds**

List of point clouds to classify.

**Type**  
list[int]

**replace\_asset**

Replace source point cloud with classified one.

**Type**  
bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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.

**frames**

List of frames to process.

**Type**

*list[int]*

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**  
str

**point\_cloud**

Point cloud key to classify.

**Type**  
int

**point\_clouds**

List of point clouds to classify.

**Type**  
list[int]

**replace\_asset**

Replace source point cloud with classified one.

**Type**  
bool

**source\_class**

Class of points to be re-classified.

**Type**  
int

**subdivide\_task**

Enable fine-level task subdivision.

**Type**  
bool

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**target\_classes**

Target point classes for classification.

**Type**  
list[int]

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**  
int

**class CleanModel**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

• **object** (*Metashape.Chunk* / *Metashape.Document*) – Chunk or Document object to be processed.

- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**criterion**

Model filtering criterion.

**Type**

*Metashape.Model.Criterion*

**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.

**gpu\_support**

GPU support flag.

**Type**

bool

**level**

Filtering threshold in percents.

**Type**

int

**model**

Model to filter.

**Type**

int

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**

int

**class CleanPointCloud**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**criterion**

Point cloud filtering criterion.

**Type**

*Metashape.PointCloud.Criterion*

**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[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**point\_cloud**

Point cloud key to filter.

**Type**

int

**point\_clouds**

List of point clouds to filter.

**Type**

list[int]

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**threshold**

Filtering threshold.

**Type**

float

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**

int

**class CleanTiePoints**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**criterion**

Tie points filtering criterion.

**Type**

*Metashape.TiePoints.Criterion*

**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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**target**

Task target.

**Type***Metashape.Tasks.TargetType***threshold**

Filtering threshold.

**Type**

float

**toNetworkTask([objects])**Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**level**

Hole size threshold in percents.

**Type**

int

**name**

Task name.

**Type**  
str

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

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

**color\_enhancement**

Enable automatic color enhancement.

**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.

**gpu\_support**

GPU support flag.

**Type**  
bool

**model**

Key of model to colorize.

**Type**  
int

**name**

Task name.

**Type**  
str

**source\_data**

Source data to extract colors from.

**Type**  
*Metashape.DataSource*

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**  
int

**class ColorizePointCloud**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**color\_enhancement**

Enable automatic color enhancement.

**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 frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**max\_workgroup\_size**

Maximum workgroup size.

**Type**

int

**name**

Task name.

**Type**

str

**point\_cloud**

Point cloud key to colorize.

**Type**

int

**point\_clouds**

List of point clouds to colorize.

**Type**

list[int]

**replace\_asset**

Replace source point cloud with colored one.

**Type**

bool

**source\_data**

Source data to extract colors from.

**Type**

*Metashape.DataSource*

**subdivide\_task**

Enable fine-level task subdivision.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**  
int

**workitem\_size\_cameras**

Number of cameras in a workitem.

**Type**  
int

**class CompactPointCloud**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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.

**frames**

List of frames to process.

**Type**  
list[int]

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**point\_cloud**

Point cloud key to process.

**Type**  
int

**point\_clouds**

List of point clouds to process.

**Type**  
list[int]

**replace\_asset**

Replace source point cloud with compacted one.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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[int]*

**color\_correction**

Apply color correction.

**Type**

bool

**color\_enhancement**

Enable automatic color enhancement.

**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.

**gpu\_support**

GPU support flag.

**Type**

bool

**image\_compression**

Image compression parameters.

**Type**

*Metashape.ImageCompression*

**merge\_planes**

Merge multispectral images.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to output file.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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

**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

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**model**

Model to process.

**Type**

int

**name**

Task name.

**Type**

str

**replace\_asset**

Replace source model with decimated model.

**Type**

bool

**target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask**([*objects* ])Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list*[*Metashape.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** (*Metashape.Chunk* / *Metashape.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[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.

**fiducials\_position\_corners**

Search corners for fiducials.

**Type**

bool

**fiducials\_position\_sides**

Search sides for fiducials.

**Type**

bool

**frame\_detector**

Detect frame.

**Type**

bool

**frames**

List of frames to process.

**Type**  
list[int]

**generate\_masks**

Generate background masks.

**Type**  
bool

**generic\_detector**

Use generic detector.

**Type**  
bool

**gpu\_support**

GPU support flag.

**Type**  
bool

**mask\_dark\_pixels**

Mask out dark pixels near frame edge.

**Type**  
bool

**name**

Task name.

**Type**  
str

**right\_angle\_detector**

Use right angle detector.

**Type**  
bool

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**v\_shape\_detector**

Detect V-shape fiducials.

**Type**  
bool

**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** (*Metashape.Chunk* / *Metashape.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[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[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**inverted**

Detect markers on black background.

**Type**

bool

**maximum\_residual**

Maximum residual for non-coded targets in pixels.

**Type**

float

**merge\_markers**

Merge detected targets with existing markers.

**Type**

bool

**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**  
str

**noparity**

Disable parity checking.

**Type**  
bool

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**target\_type**

Type of targets.

**Type**  
*Metashape.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**tolerance**

Detector tolerance (0 - 100).

**Type**  
int

**workitem\_count**

Work item count.

**Type**  
int

**class DetectPowerlines**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**max\_quantization\_error**

Maximum allowed distance between polyline and smooth continuous curve.

**Type**

float

**min\_altitude**

Minimum altitude for reconstructed powerlines.

**Type**

float

**n\_points\_per\_line**

Maximum number of vertices per detected line.

**Type**

int

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**use\_model**

Use model for visibility checks.

**Type**

bool

**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** (*Metashape.Chunk* / *Metashape.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**

*Metashape.DataSource*

**clip\_to\_boundary**

Clip to boundary shapes.

**Type**

bool

**clip\_to\_region**

Clip to chunk region.

**Type**

bool

**copy\_orthophotos**

Copy orthophotos (orthomosaic asset type only).

**Type**

bool

**copy\_patches**

Copy patches.

**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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**  
str**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType***toNetworkTask**(*[objects]*)Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.Document*) – Chunk or Document object to be processed.
- **workitem** (*int*) – Workitem index.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cameras**

List of camera keys to copy.

**Type**  
*list[int]***chunk**

Chunk to copy.

**Type**  
int**copy\_depth\_maps**

Copy depth maps.

**Type**  
bool**copy\_elevations**

Copy DEMs.

**Type**  
bool**copy\_keypoints**

Copy keypoints.

**Type**  
bool

**copy\_laser\_scans**

Copy laser scans.

**Type**

bool

**copy\_masks**

Copy masks.

**Type**

bool

**copy\_models**

Copy models.

**Type**

bool

**copy\_orthomosaics**

Copy orthomosaics.

**Type**

bool

**copy\_point\_clouds**

Copy point clouds.

**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[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**label**

New chunk label.

**Type**

str

**laser\_scans**

List of laser scan keys to copy.

**Type**

list[int]

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / list[*Metashape.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** (*Metashape.Chunk* / *Metashape.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 (Colmap and FBX formats only).

**Type**

bool

**bingo\_path\_geoin**

Path to BINGO GEO INPUT file.

**Type**

str

**bingo\_path\_gps**

Path to BINGO GPS/IMU file.

**Type**

str

**bingo\_path\_image**

Path to BINGO IMAGE COORDINATE file.

**Type**

str

**bingo\_path\_itera**

Path to BINGO ITERA file.

**Type**  
str

**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**  
str

**bundler\_save\_list**

Enables/disables export of Bundler image list file.

**Type**  
bool

**cameras**

List of cameras to export (except AgisoftXML format).

**Type**  
list[int]

**chan\_rotation\_order**

Rotation order (CHAN format only).

**Type**  
*Metashape.RotationOrder*

**convert\_to\_pinhole**

Transform images to pinhole model without distortions.

**Type**  
bool

**crs**

Output coordinate system (except AgisoftXML format).

**Type**  
*Metashape.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**

*Metashape.CamerasFormat*

**gpu\_support**

GPU support flag.

**Type**

bool

**image\_compression**

Image compression parameters.

**Type**

*Metashape.ImageCompression*

**image\_orientation**

Image coordinate system (0 - X right, 1 - X up, 2 - X left, 3 - X down).

**Type**

int

**image\_path**

Image name template

**Type**

str

**name**

Task name.

**Type**

str

**path**

Path to output file.

**Type**

str

**save\_absolute\_paths**

Save absolute image paths (BlocksExchange and RZML formats only).

**Type**

bool

**save\_images**

Enables/disables images export (Colmap format only).

**Type**

bool

**save\_invalid\_matches**

Enables/disables export of invalid image matches.

**Type**

bool

**save\_markers**

Enables/disables export of manual matching points.

**Type**

bool

**save\_masks**

Enables/disables image masks export (Colmap format only).

**Type**

bool

**save\_points**

Enables/disables export of automatic tie points.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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 ExportMarkers**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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**

*Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to output file.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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[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.

**gpu\_support**

GPU support flag.

**Type**

bool

**masks**

Masks key to export.

**Type**

int

**name**

Task name.

**Type**

str

**path**

Path to output file.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

- **objects** (*Metashape.Document* / *Metashape.Chunk* / *list*[*Metashape.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** (*Metashape.Chunk* / *Metashape.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

**block\_margin**

Block margin (m).

**Type**  
float

**clip\_to\_block**

Clip model to block region.

**Type**  
bool

**clip\_to\_boundary**

Clip model to boundary shapes.

**Type**  
bool

**clip\_to\_region**

Clip model to chunk region.

**Type**  
bool

**colors\_rgb\_8bit**

Convert colors to 8 bit RGB.

**Type**  
bool

**comment**

Optional comment (if supported by selected format).

**Type**  
str

**crs**

Output coordinate system.

**Type**  
*Metashape.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**

*Metashape.ModelFormat*

**gltf\_y\_up**

Enables/disables y-up axes notation used in glTF.

**Type**

bool

**gpu\_support**

GPU support flag.

**Type**

bool

**model**

Model key to export.

**Type**

int

**name**

Task name.

**Type**

str

**path**

Path to output model.

**Type**

str

**precision**

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

**Type**

int

**raster\_transform**

Raster band transformation.

**Type**

*Metashape.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\_metadata\_xml**

Save metadata.xml file.

**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**

*Metashape.Vector*

**strip\_extensions**

Strips camera label extensions during export.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**texture\_format**

Texture format.

**Type**

*Metashape.ImageFormat*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**viewpoint**

Default view.

**Type**

*Metashape.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** (*Metashape.Chunk* / *Metashape.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[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.

**gpu\_support**

GPU support flag.

**Type**

bool

**image\_compression**

Image compression parameters.

**Type**

*Metashape.ImageCompression*

**name**

Task name.

**Type**

str

**north\_up**

Use north-up orientation for export.

**Type**

bool

**path**

Path to output orthophoto.

**Type**

str

**projection**

Output projection.

**Type**

*Metashape.OrthoProjection*

**raster\_transform**

Raster band transformation.

**Type**

*Metashape.RasterTransformType*

**region**

Region to be exported.

**Type**

*Metashape.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

**target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask([objects])**Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.**white\_background**

Enable white background.

**Type**

bool

**workitem\_count**

Work item count.

**Type**

int

**class ExportPointCloud**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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 point classes to be exported.

**Type**

list[int]

**clip\_to\_boundary**

Clip point cloud to boundary shapes.

**Type**

bool

**clip\_to\_region**

Clip point cloud to chunk region.

**Type**

bool

**colors\_rgb\_8bit**

Convert colors to 8 bit RGB.

**Type**

bool

**comment**

Optional comment (if supported by selected format).

**Type**

str

**compression**

Enable compression (Cesium format only).

**Type**

bool

**crs**

Output coordinate system.

**Type**

*Metashape.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.

**folder\_depth**

Tileset subdivision depth (Cesium format only).

**Type**  
int

**format**

Export format.

**Type**  
*Metashape.PointCloudFormat*

**gpu\_support**

GPU support flag.

**Type**  
bool

**image\_format**

Image data format.

**Type**  
*Metashape.ImageFormat*

**name**

Task name.

**Type**  
str

**no\_double\_precision**

Use single precision float coordinates for binary ply format.

**Type**  
bool

**path**

Path to output file.

**Type**  
str

**point\_clouds**

Point cloud keys to export.

**Type**  
list[int]

**raster\_transform**

Raster band transformation.

**Type**  
*Metashape.RasterTransformType*

**region**

Region to be exported.

**Type**  
*Metashape.BBox*

**save\_comment**

Enable comment export.

**Type**  
bool

**save\_images**

Enable image export.

**Type**

bool

**save\_point\_classification**

Enables/disables export of point classification.

**Type**

bool

**save\_point\_color**

Enables/disables export of point color.

**Type**

bool

**save\_point\_confidence**

Enables/disables export of point confidence.

**Type**

bool

**save\_point\_index**

Enables/disables export of point row and column indices.

**Type**

bool

**save\_point\_intensity**

Enables/disables export of point intensity.

**Type**

bool

**save\_point\_normal**

Enables/disables export of point normal.

**Type**

bool

**save\_point\_return\_number**

Enables/disables export of point return number.

**Type**

bool

**save\_point\_scan\_angle**

Enables/disables export of point scan angle.

**Type**

bool

**save\_point\_source\_id**

Enables/disables export of point source ID.

**Type**

bool

**save\_point\_timestamp**

Enables/disables export of point timestamp.

**Type**

bool

**screen\_space\_error**

Target screen space error (Cesium format only).

**Type**

float

**shift**

Optional shift to be applied to point coordinates.

**Type**

*Metashape.Vector*

**source\_data**

Selects between point cloud and tie points. If not specified, uses point cloud if available.

**Type**

*Metashape.DataSource*

**split\_in\_blocks**

Enable tiled export.

**Type**

bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**tileset\_version**

Cesium 3D Tiles format version to export (1.0 or 1.1).

**Type**

str

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**viewpoint**

Default view.

**Type**

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

**asset**

Asset key to export.

**Type**  
int

**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**  
str

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**format**

Export format.

**Type**  
*Metashape.RasterFormat*

**global\_profile**

Use global profile (GeoPackage and TMS formats only).

**Type**  
bool

**gpu\_support**

GPU support flag.

**Type**  
bool

**height**

Raster height.

**Type**  
int

**image\_compression**

Image compression parameters.

**Type**  
*Metashape.ImageCompression*

**image\_description**

Optional description to be added to image files.

**Type**  
str

**image\_format**

Tile format.

**Type**  
*Metashape.ImageFormat*

**max\_zoom\_level**

Maximum zoom level (GeoPackage, Google Map Tiles, MBTiles and World Wind Tiles formats only).

**Type**  
int

**min\_zoom\_level**

Minimum zoom level (GeoPackage, Google Map Tiles, MBTiles and World Wind Tiles formats only).

**Type**  
int

**name**

Task name.

**Type**  
str

**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**  
str

**projection**

Output projection.

**Type**

*Metashape.OrthoProjection*

**raster\_transform**

Raster band transformation.

**Type**

*Metashape.RasterTransformType*

**region**

Region to be exported.

**Type**

*Metashape.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**

*Metashape.DataSource*

**split\_in\_blocks**

Split raster in blocks.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**tile\_height**

Tile height in pixels.

**Type**

int

**tile\_width**

Tile width in pixels.

**Type**

int

**title**

Export title.

**Type**

str

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list*[*Metashape.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**

*Metashape.Matrix*

**class ExportReference**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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**  
str

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**delimiter**

Column delimiter (CSV format only).

**Type**  
str

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

Export format.

**Type**  
*Metashape.ReferenceFormat*

**gpu\_support**

GPU support flag.

**Type**  
bool

**items**

Items to export (CSV format only).

**Type**  
*Metashape.ReferenceItems*

**name**

Task name.

**Type**  
str

**path**

Path to the output file.

**Type**  
str

**precision**

Number of digits after the decimal point (CSV format only).

**Type**  
int

**save\_enabled**

Save enabled flag (CSV format only).

**Type**  
bool

**save\_errors**

Save errors (CSV format only).

**Type**  
bool

**save\_estimated**

Save estimated values (CSV format only).

**Type**  
bool

**save\_location\_accuracy**

Save location accuracy (CSV format only).

**Type**  
bool

**save\_rotation**

Save rotation angles (CSV format only).

**Type**  
bool

**save\_rotation\_accuracy**

Save rotation accuracy (CSV format only).

**Type**  
bool

**save\_variance**

Save variance (CSV format only).

**Type**  
bool

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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**

str

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**font\_size**

Font size (pt).

**Type**

int

**gpu\_support**

GPU support flag.

**Type**

bool

**logo\_path**

Path to company logo file.

**Type**

str

**name**

Task name.

**Type**

str

**page\_numbers**

Enable page numbers.

**Type**

bool

**path**

Path to output report.

**Type**

str

**save\_lidar\_separation**

Include lidar swath separation image.

**Type**

bool

**save\_system\_info**

Include system information.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**title**

Report title.

**Type**

str

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**user\_settings**

A list of user defined settings to include on the Processing Parameters page.

**Type**

list[tuple[str, str]]

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

**crs**

Output coordinate system.

**Type**

*Metashape.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**

*Metashape.ShapesFormat*

**gpu\_support**

GPU support flag.

**Type**

bool

**groups**

A list of shape groups to export.

**Type**

list[int]

**name**

Task name.

**Type**

str

**path**

Path to shape file.

**Type**

str

**polygons\_as\_polylines**

Save polygons as polylines.

**Type**

bool

**save\_attributes**

Export attributes.

**Type**

bool

**save\_elevation**

Export elevation values for 3D shapes.

**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***Metashape.Vector***target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask**(*[objects ]*)Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**path**

Path to output file.

**Type**  
str

**raster\_transform**

Raster band transformation.

**Type**  
*Metashape.RasterTransformType*

**save\_alpha**

Enable alpha channel export.

**Type**  
bool

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**texture\_type**

Texture type.

**Type**  
*Metashape.Model.TextureType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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

**clip\_to\_region**

Clip tiled model to chunk region.

**Type**

bool

**crs**

Output coordinate system.

**Type**

*Metashape.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.

**face\_count**

Number of faces per megapixel of texture resolution (block model export only).

**Type**

int

**folder\_depth**

Tileset subdivision depth (Cesium format only).

**Type**

int

**format**

Export format.

**Type**

*Metashape.TiledModelFormat*

**gpu\_support**

GPU support flag.

**Type**

bool

**image\_compression**

Image compression parameters.

**Type**

*Metashape.ImageCompression*

**model\_compression**

Enable mesh compression (Cesium format only).

**Type**

bool

**model\_format**

Model format for zip export.

**Type**

*Metashape.ModelFormat*

**model\_group**

Block model key to export.

**Type**

int

**name**

Task name.

**Type**

str

**path**

Path to output model.

**Type**

str

**pixel\_size**

Target model resolution in meters (block model export only).

**Type**

float

**raster\_transform**

Raster band transformation.

**Type**

*Metashape.RasterTransformType*

**screen\_space\_error**

Target screen space error (Cesium format only).

**Type**

float

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**texture\_format**

Texture format.

**Type**

*Metashape.ImageFormat*

**tile\_size**

Size of tiles in pixels (block model export only).

**Type**

int

**tiled\_model**

Tiled model key to export.

**Type**

int

**tileset\_version**

Cesium 3D Tiles format version to export (1.0 or 1.1).

**Type**  
str

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**use\_tileset\_transform**

Use tileset transform instead of individual tile transforms (Cesium format only).

**Type**  
bool

**workitem\_count**

Work item count.

**Type**  
int

**zip\_compression**

Enable zip compression (Cesium format only).

**Type**  
bool

**class FilterPointCloud**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**clip\_to\_region**

Clip point cloud to chunk region.

**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 frames to process.

**Type**  
list[int]

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**point\_cloud**

Point cloud key to filter.

**Type**  
int

**point\_clouds**

List of point clouds to filter.

**Type**  
list[int]

**point\_spacing**

Desired point spacing (m).

**Type**  
float

**replace\_asset**

Replace default point cloud with filtered one.

**Type**  
bool

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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[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

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**mask\_defocus**

Mask defocus areas.

**Type**

bool

**mask\_operation**

Mask operation.

**Type**

*Metashape.MaskOperation*

**masking\_mode**

Mask generation mode.

**Type***Metashape.MaskingMode***name**

Task name.

**Type**

str

**path**

Mask file name template.

**Type**

str

**replace\_asset**

Update default set of masks with generated masks.

**Type**

bool

**target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask**(*objects* )Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list*[*Metashape.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** (*Metashape.Chunk* / *Metashape.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[float]

**cell\_size**

Step of prescription grid, meters.

**Type**  
float

**class\_count**

Number of classes.

**Type**  
int

**classification\_method**

Index values classification method.

**Type**  
*Metashape.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.

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**rates**

Fertilizer rate for each class.

**Type**  
list[float]

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

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

**crs**

Ground coordinate system.

**Type**  
*Metashape.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**  
*Metashape.CamerasFormat*

**gpu\_support**

GPU support flag.

**Type**  
bool

**image\_list**

Path to image list file (Bundler format only).

**Type**  
str

**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**  
str

**path**

Path to the file.

**Type**  
str

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**  
int

**class ImportDepthImages**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**color\_filenames**

List of corresponding color files, if present.

**Type**  
list[str]

**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. Either smartphone depth photos or depth images with corresponding color images in *color\_filenames* parameter.

**Type**  
list[str]

**format**

Point cloud format.

**Type**  
*Metashape.PointCloudFormat*

**gpu\_support**

GPU support flag.

**Type**  
bool

**image\_path**

Path template to output pre-processed files.

**Type**  
str

**multiplane**

Import as a multi-camera system

**Type**  
bool

**name**

Task name.

**Type**  
str

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / list[*Metashape.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** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to the file.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

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

**crs**

Model coordinate system.

**Type**

*Metashape.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**

*Metashape.ModelFormat*

**frame\_paths**

List of model paths to import in each frame of a multiframe chunk.

**Type**

list[str]

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to model.

**Type**

str

**replace\_asset**

Replace default asset with imported model.

**Type**

bool

**shift**

Optional shift to be applied to vertex coordinates.

**Type**

*Metashape.Vector*

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**

int

**class ImportPointCloud**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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

**columns**

Column order (x/y/z - coordinates, X/Y/Z - normal, r/g/b - color, i - intensity, t - time, c - classification, space - skip column).

**Type**

str

**crs**

Point cloud coordinate system.

**Type**

*Metashape.CoordinateSystem*

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**delimiter**

CSV delimiter.

**Type**

str

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**format**

Point cloud format.

**Type**

*Metashape.PointCloudFormat*

**frame\_paths**

List of point cloud paths to import in each frame of a multiframe chunk.

**Type**

list[str]

**generate\_panorama**

Generate panorama from point colors.

**Type**

bool

**gpu\_support**

GPU support flag.

**Type**

bool

**group\_delimiters**

Combine consecutive delimiters in csv format.

**Type**

bool

**ignore\_scanner\_origin**

Do not use laser scan origin as scanner position for structured point clouds.

**Type**

bool

**ignore\_trajectory**

Do not attach trajectory to imported point cloud.

**Type**

bool

**is\_laser\_scan**

Import point clouds as laser scans.

**Type**

bool

**load\_images**

Import images embedded in laser scan.

**Type**

bool

**load\_point\_classification**

Import point classification.

**Type**

bool

**load\_point\_color**

Import point color.

**Type**

bool

**load\_point\_confidence**

Import point confidence.

**Type**

bool

**load\_point\_index**

Import point row and column indices.

**Type**

bool

**load\_point\_intensity**

Import point intensity.

**Type**

bool

**load\_point\_normal**

Import point normal.

**Type**

bool

**load\_point\_return\_number**

Import point return number.

**Type**

bool

**load\_point\_scan\_angle**

Import point scan angle.

**Type**

bool

**load\_point\_source\_id**

Import point source ID.

**Type**

bool

**load\_point\_timestamp**

Import point timestamp.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to point cloud.

**Type**

str

**point\_neighbors**

Number of point neighbors to use for normal estimation.

**Type**

int

**precision**

Coordinate precision (m). For default precision use 0.

**Type**

float

**replace\_asset**

Replace default asset with imported point cloud.

**Type**

bool

**scanner\_at\_origin**

Use laser scan origin as scanner position for unstructured point clouds.

**Type**

bool

**shift**

Optional shift to be applied to point coordinates.

**Type**

*Metashape.Vector*

**skip\_rows**

Number of rows to skip.

**Type**

int

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**trajectory**

Trajectory key to attach.

**Type**

int

**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** (*Metashape.Chunk* / *Metashape.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**

*Metashape.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.

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**has\_nodata\_value**

No-data value valid flag.

**Type**

bool

**name**

Task name.

**Type**

str

**nodata\_value**

No-data value.

**Type**

float

**path**

Path to elevation model in GeoTIFF format.

**Type**

str

**raster\_type**

Type of raster layer to import.

**Type**

*Metashape.DataSource*

**replace\_asset**

Replace default raster with imported one.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

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

**column\_a**

1st rotation angle column (CSV format only).

**Type**

int

**column\_b**

2nd rotation angle column (CSV format only).

**Type**

int

**column\_c**

3rd rotation angle column (CSV format only).

**Type**

int

**column\_enabled**

Enabled flag column (CSV format only).

**Type**

int

**column\_label**

Label column (CSV format only).

**Type**

int

**column\_sa**

1st rotation angle accuracy column (CSV format only).

**Type**

int

**column\_sb**

2nd rotation angle accuracy column (CSV format only).

**Type**  
int

**column\_sc**

3rd rotation angle accuracy column (CSV format only).

**Type**  
int

**column\_sx**

X coordinate accuracy column (CSV format only).

**Type**  
int

**column\_sy**

Y coordinate accuracy column (CSV format only).

**Type**  
int

**column\_sz**

Z coordinate accuracy column (CSV format only).

**Type**  
int

**column\_x**

X coordinate column (CSV format only).

**Type**  
int

**column\_y**

Y coordinate column (CSV format only).

**Type**  
int

**column\_z**

Z coordinate column (CSV format only).

**Type**  
int

**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**  
str

**create\_markers**

Create markers for missing entries (CSV format only).

**Type**  
bool

**crs**

Reference data coordinate system (CSV format only).

**Type**  
*Metashape.CoordinateSystem*

**decode(dict)**

Initialize task parameters with a dictionary.

**decodeJSON(json)**

Initialize task parameters from a JSON string.

**delimiter**

Column delimiter (CSV format only).

**Type**  
str

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

File format.

**Type**  
*Metashape.ReferenceFormat*

**gpu\_support**

GPU support flag.

**Type**  
bool

**group\_delimiters**

Combine consecutive delimiters (CSV format only).

**Type**  
bool

**ignore\_labels**

Matches reference data based on coordinates alone (CSV format only).

**Type**  
bool

**items**

Items to load reference for (CSV format only).

**Type**  
*Metashape.ReferenceItems*

**load\_enabled**

Load enabled flag (CSV format only).

**Type**  
bool

**load\_location\_accuracy**

Load location accuracy (CSV format only).

**Type**  
bool

**load\_rotation**

Load rotation angles (CSV format only).

**Type**  
bool

**load\_rotation\_accuracy**

Load rotation accuracy (CSV format only).

**Type**  
bool

**name**

Task name.

**Type**  
str

**path**

Path to the file with reference data.

**Type**  
str

**rotation\_angles**

Euler angles triplet used for rotation reference (CSV format only).

**Type**  
*Metashape.EulerAngles*

**shutter\_lag**

Shutter lag in seconds (APM format only).

**Type**  
float

**skip\_rows**

Number of rows to skip (CSV format only).

**Type**  
int

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**threshold**

Error threshold in meters used when ignore\_labels is set (CSV format only).

**Type**  
float

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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**

*Metashape.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**

str

**crs**

Reference data coordinate system (csv format only).

**Type**

*Metashape.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**

str

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

Shapes format.

**Type**

*Metashape.ShapesFormat*

**gpu\_support**

GPU support flag.

**Type**

bool

**group\_delimiters**

Combine consecutive delimiters in csv format.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to shape file.

**Type**  
str

**replace**

Replace current shapes with new data.

**Type**  
bool

**skip\_rows**

Number of rows to skip in (csv format only).

**Type**  
int

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to tiled model.

**Type**

str

**target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask([objects])**Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.**workitem\_count**

Work item count.

**Type**

int

**class ImportTrajectory**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**columns**

Column order in csv format (t - time, x/y/z - coordinates, a/b/c - rotation angles, space - skip column).

**Type**

str

**crs**

Point cloud coordinate system.

**Type***Metashape.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**  
str

**encode()**

Create a dictionary with task parameters.

**encodeJSON()**

Create a JSON string with task parameters.

**format**

Trajectory format.

**Type**  
*Metashape.TrajectoryFormat*

**gpu\_support**

GPU support flag.

**Type**  
bool

**group\_delimiters**

Combine consecutive delimiters in csv format.

**Type**  
bool

**name**

Task name.

**Type**  
str

**path**

Trajectory file path.

**Type**  
str

**replace\_asset**

Replace default asset with imported trajectory.

**Type**  
bool

**shift**

Optional shift to be applied to point coordinates.

**Type**  
*Metashape.Vector*

**skip\_rows**

Number of rows to skip in (csv format only).

**Type**  
int

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**

int

**class ImportVideo**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**custom\_frame\_step**

Every *custom\_frame\_step*'th frame will be saved. Used for *frame\_step=CustomFrameStep*.

**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.

**frame\_step**

Frame step type.

**Type**

*Metashape.FrameStep*

**gpu\_support**

GPU support flag.

**Type**

bool

**image\_path**

Path to directory where to save frames with filename template. For example: */path/to/dir/frame{filenum}.png*.

**Type**

str

**name**

Task name.

**Type**  
str

**path**

Path to source video.

**Type**  
str

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**time\_end**

The endpoint for importing video, seconds.

**Type**  
float

**time\_start**

The starting point for importing video, seconds.

**Type**  
float

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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[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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

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

**archive**

Override project format when using non-standard file extension.

**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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to project file.

**Type**

str

**read\_only**

Open project in read only mode.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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[int]*

**decode(*dict*)**

Initialize task parameters with a dictionary.

**decodeJSON(*json*)**

Initialize task parameters from a JSON string.

**downscale**

Image alignment accuracy (0 - Highest, 1 - High, 2 - Medium, 4 - Low, 8 - Lowest).

**Type**

int

**downscale\_3d**

Laser scan alignment accuracy (1 - Highest, 2 - High, 4 - Medium, 8 - Low, 16 - Lowest).

**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

**gpu\_support**

GPU support flag.

**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\_3d**

Key point limit for laser scans.

**Type**  
int

**keypoint\_limit\_depth\_maps**

Key point limit for depth maps.

**Type**  
int

**keypoint\_limit\_per\_mpx**

Key point limit per megapixel.

**Type**  
int

**laser\_scans\_use\_initial\_orientation**

Use initial laser scan orientation for keypoint matching.

**Type**  
bool

**laser\_scans\_vertical\_axis**

Common laser scans axis.

**Type**  
int

**mask\_tiepoints**

Apply mask filter to tie points.

**Type**  
bool

**match\_depth\_maps**

Match images with laser scans using geometric features.

**Type**  
bool

**match\_laser\_scans**

Match laser scans using geometric features.

**Type**  
bool

**max\_workgroup\_size**

Maximum workgroup size.

**Type**  
int

**name**

Task name.

**Type**  
str

**pairs**

User defined list of camera pairs to match.

**Type**  
list[tuple[int, int]]

**reference\_preselection**

Enable reference preselection.

**Type**

bool

**reference\_preselection\_mode**

Reference preselection mode.

**Type**

*Metashape.ReferencePreselectionMode*

**reset\_matches**

Reset current matches.

**Type**

bool

**subdivide\_task**

Enable fine-level task subdivision.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**tiepoint\_limit**

Tie point limit.

**Type**

int

**toNetworkTask([objects])**

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

**Parameters**

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

**apply\_boresight\_offset**

Apply boresight offset when merging point clouds.

**Type**

bool

**apply\_trajectory\_mask**

Apply trajectory mask when merging point clouds.

**Type**

bool

**assets**

List of assets to process.

**Type**

list[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.

**gpu\_support**

GPU support flag.

**Type**

bool

**merge\_orthophotos**

Merge orthophotos when merging orthomosaics.

**Type**

bool

**name**

Task name.

**Type**

str

**source\_data**

Asset type.

**Type**

*Metashape.DataSource*

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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[int]

**copy\_depth\_maps**

Copy depth maps.

**Type**

bool

**copy\_elevations**

Copy DEMs.

**Type**

bool

**copy\_laser\_scans**

Copy laser scans.

**Type**

bool

**copy\_masks**

Copy masks.

**Type**

bool

**copy\_models**

Copy models.

**Type**

bool

**copy\_orthomosaics**

Copy orthomosaics.

**Type**

bool

**copy\_point\_clouds**

Copy point clouds.

**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.

**gpu\_support**

GPU support flag.

**Type**

bool

**merge\_assets**

Merge default assets.

**Type**

bool

**merge\_markers**

Merge markers.

**Type**

bool

**merge\_tiepoints**

Merge tie points.

**Type**

bool

**name**

Task name.

**Type**

str

**target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask([objects])**Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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 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

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**tiepoint\_covariance**

Estimate tie point covariance matrices.

**Type**

bool

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**

int

**class PansharpenOrthomosaic**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**channels**

Orthomosaic channel mask (boolean flags, e.g. 0b0010 means only 1st channel is used and the rest ignored).

**Type**

int

**clip\_to\_pan\_data**

Clip result to high resolution orthomosaic.

**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 frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**  
str

**orthomosaic**

Orthomosaic to pansharpener.

**Type**  
int

**pan\_channels**

Detailed orthomosaic channel mask (boolean flags, e.g. 0b0010 means only 1st channel is used and the rest ignored).

**Type**  
int

**pan\_orthomosaic**

Detailed orthomosaic.

**Type**  
int

**replace\_asset**

Replace source orthomosaic with pansharpenered result.

**Type**  
bool

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**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**

str

**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

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**use\_selection**

Focus on model selection.

**Type**

bool

**workitem\_count**

Work item count.

**Type**  
int

**class PublishData**

Task class containing processing parameters.

**account**

Account name (Nira (Key ID) service).

**Type**  
str

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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**

Dataset description.

**Type**  
str

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**gpu\_support**

GPU support flag.

**Type**  
bool

**hostname**

Service hostname (4DMapper and Nira services).

**Type**  
str

**image\_compression**

Image compression parameters.

**Type**  
*Metashape.ImageCompression*

**is\_draft**

Mark dataset as draft (Sketchfab service).

**Type**  
bool

**is\_private**

Set dataset access to private (Pointbox and Sketchfab services).

**Type**

bool

**is\_protected**

Set dataset access to protected (Pointbox service).

**Type**

bool

**max\_zoom\_level**

Maximum zoom level.

**Type**

int

**min\_zoom\_level**

Minimum zoom level.

**Type**

int

**name**

Task name.

**Type**

str

**owner**

Account owner (Cesium and Mapbox services).

**Type**

str

**password**

Account password (4DMapper, Agisoft Cloud, Pointscene and Sketchfab services).

**Type**

str

**point\_classes**

List of point classes to be exported.

**Type**

list[int]

**project\_id**

Id of a target project (from Agisoft Cloud project URL).

**Type**

str

**projection**

Output projection.

**Type**

*Metashape.CoordinateSystem*

**raster\_transform**

Raster band transformation.

**Type**

*Metashape.RasterTransformType*

**resolution**

Output resolution in meters.

**Type**

float

**save\_camera\_track**

Enables/disables export of camera track.

**Type**

bool

**save\_point\_color**

Enables/disables export of point colors.

**Type**

bool

**service**

Service to upload on.

**Type**

*Metashape.ServiceType*

**source\_data**

Asset type to upload.

**Type**

*Metashape.DataSource*

**tags**

Dataset tags.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**tile\_size**

Tile size in pixels.

**Type**

int

**title**

Dataset title.

**Type**

str

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**token**

Account token (Cesium, Mapbox, Nira (Key Secret), Picterra, Pointbox and Sketchfab services).

**Type**

str

**upload\_images**

Attach photos to Nira publication.

**Type**

bool

**username**

Account username (4DMapper, Agisoft Cloud and Pointscene services).

**Type**

str

**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** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**overlap**

Target number of cameras observing each point of the surface.

**Type**

int

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**use\_selection**

Focus on model selection.

**Type**

bool

**workitem\_count**

Work item count.

**Type**

int

**class RefineModel**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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[int]

**decode**(*dict*)

Initialize task parameters with a dictionary.

**decodeJSON**(*json*)

Initialize task parameters from a JSON string.

**downscale**

Refinement quality (1 - Ultra high, 2 - High, 4 - Medium, 8 - Low, 16 - Lowest).

**Type**

int

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**  
bool

**iterations**

Number of refinement iterations.

**Type**  
int

**model**

Model to process.

**Type**  
int

**name**

Task name.

**Type**  
str

**replace\_asset**

Replace default asset with refined model.

**Type**  
bool

**smoothness**

Smoothing strength. Should be in range [0, 1].

**Type**  
float

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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**  
str

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**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**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

- **objects** (*Metashape.Document* / *Metashape.Chunk* / *list*[*Metashape.Chunk*]) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**  
int

**class RenderDepthMaps**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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[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.

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**path\_depth**

Path to depth map.

**Type**  
str

**path\_diffuse**

Path to diffuse map.

**Type**  
str

**path\_normals**

Path to normal map.

**Type**  
str

**save\_depth**

Enable export of depth map.

**Type**

bool

**save\_diffuse**

Enable export of diffuse map.

**Type**

bool

**save\_normals**

Enable export of normal map.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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[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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

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

**args**

Script arguments.

**Type**

str

**code**

Script code.

**Type**

str

**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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Script path.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

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

**archive**

Override project format when using non-standard file extension.

**Type**

bool

**chunks**

List of chunks to be saved.

**Type**

*list[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.

**gpu\_support**

GPU support flag.

**Type**

bool

**name**

Task name.

**Type**

str

**path**

Path to project.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**version**

Project version to save.

**Type**

str

**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** (*Metashape.Chunk* / *Metashape.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

**frames**

List of frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**model**

Key of model to smooth.

**Type**

int

**name**

Task name.

**Type**

str

**preserve\_edges**

Preserve edges.

**Type**

bool

**replace\_asset**

Replace default asset with smoothed model.

**Type**

bool

**strength**

Smoothing strength.

**Type**

float

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.

**workitem\_count**

Work item count.

**Type**

int

**class SmoothPointCloud**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

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

**apply\_to\_selection**

Smooth points within selection.

**Type**

bool

**classes**

List of point classes to be smoothed.

**Type**

*list[int]*

**clip\_to\_region**

Clip point cloud to chunk region.

**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 frames to process.

**Type**  
list[int]

**gpu\_support**

GPU support flag.

**Type**  
bool

**name**

Task name.

**Type**  
str

**point\_cloud**

Key of point cloud to smooth.

**Type**  
int

**point\_clouds**

List of point clouds to smooth.

**Type**  
list[int]

**replace\_asset**

Replace default point cloud with smoothed one.

**Type**  
bool

**smoothing\_radius**

Desired smoothing radius (m).

**Type**  
float

**target**

Task target.

**Type**  
*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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** ([Metashape.Chunk](#) / [Metashape.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

**gpu\_support**

GPU support flag.

**Type**

bool

**last\_frame**

Ending frame index.

**Type**

int

**name**

Task name.

**Type**

str

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

**toNetworkTask([objects])**

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

**Parameters**

- **objects** ([Metashape.Document](#) / [Metashape.Chunk](#) / *list[Metashape.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** (*Metashape.Chunk* / *Metashape.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

**clip\_to\_boundary**

Clip raster to boundary shapes.

**Type**

bool

**copy\_orthophotos**

Copy orthophotos (orthomosaic asset type only).

**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 frames to process.

**Type**

list[int]

**gpu\_support**

GPU support flag.

**Type**

bool

**height**

Raster height.

**Type**

int

**name**

Task name.

**Type**

str

**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**  
*Metashape.OrthoProjection*

**region**

Region to be processed.

**Type**  
*Metashape.BBox*

**replace\_asset**

Replace default raster with transformed one.

**Type**  
bool

**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

**source\_data**

Selects between DEM and orthomosaic.

**Type**  
*Metashape.DataSource*

**subtract**

Subtraction flag.

**Type**

bool

**target**

Task target.

**Type**

*Metashape.Tasks.TargetType*

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

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

**Parameters**

**objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.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**

*Metashape.Matrix*

**class TriangulateTiePoints**

Task class containing processing parameters.

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

Apply task to specified object.

**Parameters**

- **object** (*Metashape.Chunk* / *Metashape.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.

**gpu\_support**

GPU support flag.

**Type**

bool

**max\_error**

Reprojection error threshold.

**Type**

float

**min\_image**

Minimum number of point projections.

**Type**

int

**name**

Task name.

**Type**

str

**target**

Task target.

**Type***Metashape.Tasks.TargetType***toNetworkTask**(*[objects ]*)Convert task to *Metashape.NetworkTask* to be applied to specified objects.**Parameters****objects** (*Metashape.Document* / *Metashape.Chunk* / *list[Metashape.Chunk]*) – Objects to be processed.**workitem\_count**

Work item count.

**Type**

int

**createTask**(*name*)

Create task object by its name.

**Parameters****name** (*str*) – 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***Metashape.Thumbnail*

**image()**

Returns image data.

**Returns**

Image data.

**Return type**

*Metashape.Image*

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

Loads thumbnail from file.

**Parameters**

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

**setImage(image)**

**Parameters**

**image** (*Metashape.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**

*Metashape.MetaData*

**modified**

Modified flag.

**Type**

bool

**values()**

List of item values.

**class Metashape.TiePoints**

Tie point cloud instance

**class Cameras**

Collection of *Metashape.TiePoints.Projections* objects indexed by corresponding cameras

**class Criterion**

Tie points filter criterion in [ReprojectionError, ReconstructionUncertainty, ImageCount, ProjectionAccuracy]

**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.TiePoints.Filter()
>>> f.init(chunk, criterion = Metashape.TiePoints.Filter.ReprojectionError)
>>> f.selectPoints(threshold)

```

### class Criterion

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

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

Initialize tie points filter based on specified criterion.

#### Parameters

- **points** (*Metashape.TiePoints* / *Metashape.Chunk*) – Tie points to filter.
- **criterion** (*Metashape.TiePoints.Filter.Criterion*) – Point filter criterion.
- **progress** (*Callable[[float], None]*) – Progress callback.

### max\_value

Maximum value.

#### Type

int | float

### min\_value

Minimum value.

#### Type

int | float

### 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[int] | list[float]

### class Point

3D point in the tie point cloud

### coord

Point coordinates.

#### Type

*Metashape.Vector*

### cov

Point coordinates covariance matrix.

**Type***Metashape.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 tie point cloud

**copy()**

Returns a copy of points buffer.

**Returns**

Copy of points buffer.

**Return type***Metashape.TiePoints.Points***resize(count)**

Resize points list.

**Parameters****count** (*int*) – new point count**class Projection**

Projection of the 3D point on the photo

**coord**

2D projection coordinates.

**Type***Metashape.Vector***size**

Point size.

**Type**

float

**track\_id**

Track index.

**Type**

int

**class Projections**Collection of *Metashape.TiePoints.Projection* for the camera**copy()**

Returns a copy of projections buffer.

**Returns**

Copy of projections buffer.

**Return type**

*Metashape.TiePoints.Projections*

**resize(count)**

Resize projections list.

**Parameters**

**count** (*int*) – new projections count

**class Track**

Track in the tie point cloud

**color**

Track color.

**Type**

tuple[int | float, ...]

**class Tracks**

Collection of tracks in the tie point cloud

**copy()**

Returns a copy of tracks buffer.

**Returns**

Copy of tracks buffer.

**Return type**

*Metashape.TiePoints.Tracks*

**resize(count)**

Resize track list.

**Parameters**

**count** (*int*) – new track count

**bands**

List of color bands.

**Type**

list[str]

**cleanup([progress])**

Remove points with insufficient number of projections.

**Parameters**

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

**copy(keypoints=True)**

Returns a copy of the tie point cloud.

**Parameters**

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

**Returns**

Copy of the tie point cloud.

**Return type**

*Metashape.TiePoints*

**cropSelectedPoints()**

Crop selected points.

**cropSelectedTracks()**

Crop selected tie points.

**data\_type**

Data type used to store color values.

**Type**

*Metashape.DataType*

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

Export tie points.

**Parameters**

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

**invertSelection()**

Invert selection.

**meta**

Tie points meta data.

**Type**

*Metashape.MetaData*

**modified**

Modified flag.

**Type**

bool

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

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

**Parameters**

- **origin** (*Metashape.Vector*) – Ray origin.
- **target** (*Metashape.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**

*Metashape.Vector*

**points**

List of points.

**Type**

*Metashape.TiePoints.Points*

**projections**

Point projections for each photo.

**Type***Metashape.TiePoints.Projections***projections\_3d**

Point projections for each laser scan.

**Type***Metashape.TiePoints.Projections***removeKeypoints()**

Remove keypoints from tie point cloud.

**removeSelectedPoints()**

Remove selected points.

**removeSelectedTracks()**

Remove selected tie points.

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

Render tie points depth image for specified viewpoint.

**Parameters**

- **transform** (*Metashape.Matrix*) – Camera location.
- **calibration** (*Metashape.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***Metashape.Image***renderImage**(*transform, calibration, point\_size=1, cull\_points=False, add\_alpha=True, raster\_transform=RasterTransformNone*)

Render tie points image for specified viewpoint.

**Parameters**

- **transform** (*Metashape.Matrix*) – Camera location.
- **calibration** (*Metashape.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** (*Metashape.RasterTransformType*) – Raster band transformation.

**Returns**

Rendered image.

**Return type***Metashape.Image*

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

Render tie points mask image for specified viewpoint.

**Parameters**

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

**Returns**

Rendered image.

**Return type**

*Metashape.Image*

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

Render image with tie points normals for specified viewpoint.

**Parameters**

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

*Metashape.Image*

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

Generate tie points preview image.

**Parameters**

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

**Returns**

Preview image.

**Return type**

*Metashape.Image*

**tracks**

List of tracks.

**Type**

*Metashape.TiePoints.Tracks*

**class Metashape.TiledModel**

Tiled model data.

**class FaceCount**

Tiled model face count in [LowFaceCount, MediumFaceCount, HighFaceCount]

**bands**

List of color bands.

**Type**

list[str]

**clear()**

Clears tiled model data.

**copy()**

Create a copy of the tiled model.

**Returns**

Copy of the tiled model.

**Return type**

*Metashape.TiledModel*

**crs**

Reference coordinate system.

**Type**

*Metashape.CoordinateSystem* | None

**data\_type**

Data type used to store color values.

**Type**

*Metashape.DataType*

**key**

Tiled model identifier.

**Type**

int

**label**

Tiled model label.

**Type**

str

**meta**

Tiled model meta data.

**Type**

*Metashape.MetaData*

**modified**

Modified flag.

**Type**

bool

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

Returns ray intersection with the tiled model.

**Parameters**

- **origin** (*Metashape.Vector*) – Ray origin.
- **target** (*Metashape.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**

*Metashape.Vector*

**renderDepth**(*transform, calibration, resolution=1, cull\_faces=True, add\_alpha=True*)

Render tiled model depth image for specified viewpoint.

**Parameters**

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

*Metashape.Image*

**renderImage**(*transform, calibration, resolution=1, cull\_faces=True, add\_alpha=True, raster\_transform=RasterTransformNone, matcap\_image, smooth\_normals=True, color*)

Render tiled model image for specified viewpoint.

**Parameters**

- **transform** (*Metashape.Matrix*) – Camera location.
- **calibration** (*Metashape.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** (*Metashape.RasterTransformType*) – Raster band transformation.
- **matcap\_image** (*Metashape.Image*) – Matcap image used to shade model.
- **smooth\_normals** – Enable normals smoothing.:type smooth\_normals: bool
- **color** (*list[int]:return: Rendered image.*) – Solid view color.

**Return type**

*Metashape.Image*

**renderMask**(*transform*, *calibration*, *resolution=1*, *cull\_faces=True*)

Render tiled model mask image for specified viewpoint.

**Parameters**

- **transform** (*Metashape.Matrix*) – Camera location.
- **calibration** (*Metashape.Calibration*) – Camera calibration.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull\_faces** (*bool*) – Enable back-face culling.

**Returns**

Rendered image.

**Return type**

*Metashape.Image*

**renderNormalMap**(*transform*, *calibration*, *resolution=1*, *cull\_faces=True*, *add\_alpha=True*)

Render image with tiled model normals for specified viewpoint.

**Parameters**

- **transform** (*Metashape.Matrix*) – Camera location.
- **calibration** (*Metashape.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.
- **smooth\_normals** – Enable normals smoothing.:type smooth\_normals: bool

**Returns**

Rendered image.

**Return type**

*Metashape.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** (*Metashape.Matrix*) – 4x4 viewpoint transformation matrix.
- **progress** (*Callable[[float], None]*) – Progress callback.

**Returns**

Preview image.

**Return type**

*Metashape.Image*

**resolution**

Tiled model resolution in m/pix.

**Type**

float

**transform**

4x4 tiled model transformation matrix.

**Type**

*Metashape.Matrix*

**class Metashape.TiledModelFormat**

Tiled model format in [TiledModelFormatNone, TiledModelFormatTLS, TiledModelFormatLOD, TiledModelFormatZIP, TiledModelFormatCesium, TiledModelFormatSLPK, TiledModelFormatOSGB, TiledModelFormatOSGT, TiledModelFormat3MX]

**class Metashape.Trajectory**

Trajectory data.

**class Position**

Trajectory position

**location**

Position coordinates.

**Type**

*Metashape.Vector*

**masked**

Position mask flag.

**Type**

bool

**rotation**

Rotation vector.

**Type**

*Metashape.Vector*

**selected**

Position selection flag.

**Type**

bool

**time**

Position timestamp.

**Type**

float

**class Positions**

Collection of trajectory positions

**copy()**

Returns a copy of position list.

**Returns**

Copy of position list.

**Return type**

*Metashape.Trajectory.Positions*

**resize(count)**

Resize position list.

**Parameters**

**count** (*int*) – new position count

**addTimeOffset**(*offset*)

Add time offset to trajectory positions.

**Parameters**

**offset** (*float*) – Time offset to add, in seconds.

**antenna**

GPS antenna correction.

**Type**

*Metashape.Antenna*

**clear**()

Clear trajectory data.

**copy**()

Create a copy of the trajectory.

**Returns**

Copy of the trajectory.

**Return type**

*Metashape.Trajectory*

**createMask**(*heading\_threshold=10, min\_duration=5, min\_distance=20, crop\_with\_data=False, trim\_ends=True*[, *progress* ])

Create automatic trajectory mask.

**Parameters**

- **heading\_threshold** (*float*) – Maximum difference of heading in segment, in degrees.
- **min\_duration** (*float*) – Minimal time duration of segment, in seconds.
- **min\_distance** (*float*) – Minimal distance of segment, in meters.
- **crop\_with\_data** (*bool*) – Enable segment cropping with corresponding point clouds.
- **trim\_ends** (*bool*) – Enable segment ends trimming.
- **progress** (*Callable[[float], None]*) – Progress callback.

**cropMaskBySelection**()

Crop trajectory mask by selection.

**crs**

Reference coordinate system.

**Type**

*Metashape.CoordinateSystem* | None

**extent**

Trajectory extent in local trajectory coordinate system.

**Type**

*Metashape.BBox*

**interpolate**(*time, max\_interval=0*)

Interpolate trajectory at specified time moment.

**Parameters**

- **time** (*float*) – Time to interpolate at.

- **max\_interval** (*float*) – Maximum sampling interval allowed for interpolation (unlimited if 0).

**Returns**

Interpolated trajectory position.

**Return type**

*Metashape.Trajectory.Position*

**invertSelection()**

Invert selection.

**key**

Trajectory identifier.

**Type**

int

**label**

Trajectory label.

**Type**

str

**maskSelectedPositions()**

Mask selected trajectory positions.

**meta**

Trajectory meta data.

**Type**

*Metashape.MetaData*

**modified**

Modified flag.

**Type**

bool

**position\_count**

Number of positions in trajectory.

**Type**

int

**positions**

List of trajectory positions.

**Type**

*Metashape.Trajectory.Positions*

**resetMask()**

Reset trajectory mask.

**sampling\_interval**

Trajectory sampling interval.

**Type**

float

**selected**

Selects/deselects the trajectory.

**Type**

bool

**transform**

4x4 trajectory transformation matrix.

**Type**

*Metashape.Matrix*

**unmaskSelectedPositions()**

Unmask selected trajectory positions.

**class Metashape.TrajectoryFormat**

Trajectory format in [TrajectoryFormatNone, TrajectoryFormatCSV, TrajectoryFormatSBET, TrajectoryFormatSOL, TrajectoryFormatTRJ]

**class Metashape.Utils**

Utility functions.

**createChessboardImage(*calib*, *cell\_size=150*, *max\_tilt=30*)**

Synthesizes photo of a chessboard.

**Parameters**

- **calib** (*Metashape.Calibration*) – Camera calibration.
- **cell\_size** (*float*) – Chessboard cell size.
- **max\_tilt** (*float*) – Maximum camera tilt in degrees.

**Returns**

Resulting image.

**Return type**

*Metashape.Image*

**createDifferenceMask(*image*, *background*, *tolerance=10*, *fit\_colors=True*)**

Creates mask from a pair of images or an image and specified color.

**Parameters**

- **image** (*Metashape.Image*) – Image to be masked.
- **background** (*Metashape.Image* | *tuple[int, ...]*) – Background image or color value.
- **tolerance** (*int*) – Tolerance value.
- **fit\_colors** (*bool*) – Enables white balance correction.

**Returns**

Resulting mask.

**Return type**

*Metashape.Image*

**createMarkers(*chunk*, *projections*)**

Creates markers from a list of non-coded projections.

**Parameters**

- **chunk** (`Metashape.Chunk`) – Chunk to create markers in.
- **projections** (`list[tuple[Metashape.Camera, Metashape.Target]]`) – 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** (`Metashape.Image`) – Image to process.
- **type** (`Metashape.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[Metashape.Target]`

**dmat2euler**(*R*, *dR*, *euler\_angles=EulerAnglesYPR*)

Calculate tangent euler rotation vector from tangent rotation matrix.

**Parameters**

- **R** (`Metashape.Matrix`) – Rotation matrix.
- **dR** (`Metashape.Matrix`) – Tangent rotation matrix.
- **euler\_angles** (`Metashape.EulerAngles`) – Euler angles to use.

**Returns**

Tangent rotation angles in degrees.

**Return type**

`Metashape.Vector`

**estimateImageQuality**(*image* [, *mask*])

Estimate image sharpness.

**Parameters**

- **image** (`Metashape.Image`) – Image to be analyzed.
- **mask** (`Metashape.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** (*Metashape.Vector*) – Rotation vector.
- **euler\_angles** (*Metashape.EulerAngles*) – Euler angles to use.

**Returns**

Rotation matrix.

**Return type**

*Metashape.Matrix*

**mat2euler**(*R*, *euler\_angles=EulerAnglesYPR*)

Calculate euler rotation angles from camera to world rotation matrix.

**Parameters**

- **R** (*Metashape.Matrix*) – Rotation matrix.
- **euler\_angles** (*Metashape.EulerAngles*) – Euler angles to use.

**Returns**

Rotation angles in degrees.

**Return type**

*Metashape.Vector*

**mat2opk**(*R*)

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

**Parameters**

**R** (*Metashape.Matrix*) – Rotation matrix.

**Returns**

Omega, phi, kappa angles in degrees.

**Return type**

*Metashape.Vector*

**mat2rvec**(*R*)

Calculate rotation vector from rotation matrix.

**Parameters**

**R** (*Metashape.Matrix*) – Rotation matrix.

**Returns**

Rotation vector.

**Return type**

*Metashape.Vector*

**mat2ypr**(*R*)

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

**Parameters**

**R** (*Metashape.Matrix*) – Rotation matrix.

**Returns**

Yaw, pitch roll angles in degrees.

**Return type***Metashape.Vector***opk2mat** (*angles*)

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

**Parameters****angles** (*Metashape.Vector*) – Omega, phi, kappa angles in degrees.**Returns**

Rotation matrix.

**Return type***Metashape.Matrix***rvec2mat** (*rvec*)

Calculate rotation matrix from rotation vector.

**Parameters****rvec** (*Metashape.Vector*) – Rotation vector.**Returns**

Rotation matrix.

**Return type***Metashape.Matrix***ypr2mat** (*angles*)

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

**Parameters****angles** (*Metashape.Vector*) – Yaw, pitch, roll angles in degrees.**Returns**

Rotation matrix.

**Return type***Metashape.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***Metashape.Vector*

**cross(*a*, *b*)**

Cross product of 2 vectors.

**Parameters**

- **a** (*Metashape.Vector*) – First vector.
- **b** (*Metashape.Vector*) – Second vector.

**Returns**

Cross product.

**Return type**

*Metashape.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**

*Metashape.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**

*Metashape.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**

*Metashape.Vector*

**coo**

Center of orbit.

**Type**

*Metashape.Vector*

**copy()**

Return a copy of the object.

**Returns**

A copy of the object.

**Return type**

*Metashape.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**

*Metashape.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**

*Metashape.Vignetting*



## PYTHON API CHANGE LOG

### 3.1 Metashape version 2.3.1

- Added `Sensor.FilmTransformType` enum
- Added `PointCloudViewTime` to `ModelView.PointCloudViewMode` enum
- Added `Calibration.todict()` method
- Added `PointCloud.selectPointsByHeight()` method
- Added `Sensor.film_transform_type` attribute
- Added `film_transform`, `width` and `height` attributes to `Camera` class
- Added `color_enhancement` attribute to `BuildOrthomosaic`, `BuildPanorama`, `BuildTexture`, `BuildTiledModel`, `ColorizeModel`, `ColorizePointCloud` and `ConvertImages` classes
- Added `film_size` argument to `Calibration.save()` method
- Added `color_enhancement` argument to `Chunk.buildOrthomosaic()`, `Chunk.buildPanorama()`, `Chunk.buildTexture()`, `Chunk.buildTiledModel()`, `Chunk.colorizeModel()`, `Chunk.colorizePointCloud()` and `Chunk.convertImages()` methods

### 3.2 Metashape version 2.3.0

- Added `CleanPointCloud` class
- Added `Sensor.Axes` enum
- Added `PointCloud.Criterion` enum
- Added `EquidistantFisheye` and `EquisolidFisheye` to `Sensor.Type` enum
- Added `NaturalBlending` to `BlendingMode` enum
- Added `ModelViewAssignedImage` to `ModelView.ModelViewMode` enum
- Added `PointCloudViewHeightAGL` to `ModelView.PointCloudViewMode` enum
- Added `Chunk.cleanPointCloud()` method
- Added `Sensor.axes` attribute
- Added `size` and `time` attribute to `Marker.Projection` class
- Added `TiePoints.projections_3d` attribute
- Added `OrthoProjection.interior` attribute

- Added `Elevation.legend_range` attribute
- Added `ImageCompression.tiff_planar` attribute
- Added `match_depth_maps` and `keypoint_limit_depth_maps` attributes to `MatchPhotos` class
- Added `downscale`, `sharpening`, `use_assigned_images` and `out_of_focus_filter` attribute to `BuildTexture` class
- Added `ExportReport.save_lidar_separation` attribute
- Added `ExportPointCloud.no_double_precision` attribute
- Added `point_clouds`, `frames` and `replace_asset` attributes to `CalculatePointNormals`, `ClassifyGroundPoints`, `ClassifyOverlapPoints`, `ClassifyPoints` and `CompactPointCloud` classes
- Added `point_clouds` attribute to `ColorizePointCloud`, `FilterPointCloud` and `SmoothPointCloud` classes
- Added `match_depth_maps` and `keypoint_limit_depth_maps` arguments to `Chunk.matchPhotos()` method
- Added `downscale`, `sharpening`, `use_assigned_images` and `out_of_focus_filter` arguments to `Chunk.buildTexture()` method
- Added `save_lidar_separation` argument to `Chunk.exportReport()` method
- Added `no_double_precision` argument to `Chunk.exportPointCloud()` method
- Added `point_clouds`, `frames` and `replace_asset` arguments to `Chunk.calculatePointNormals()` method
- Added `point_clouds` argument to `Chunk.colorizePointCloud()`, `Chunk.filterPointCloud()` and `Chunk.smoothPointCloud()` methods
- Changed default value of `BuildTexture.blending_mode` attribute to `NaturalBlending`
- Changed default value of `blending_mode` argument to `NaturalBlending` in `Chunk.buildTexture()` method
- Renamed `ExportReport.include_system_info` attribute to `save_system_info`
- Renamed `include_system_info` argument to `save_system_info` in `Chunk.exportReport()` method
- Removed `Elevation.palette_absolute_values` attribute
- Removed `flip_x`, `flip_y` and `flip_z` attributes from `BuildDem` and `BuildOrthomosaic` classes
- Removed `flip_x`, `flip_y` and `flip_z` arguments from `Chunk.buildDem()` and `Chunk.buildOrthomosaic()` methods

### 3.3 Metashape version 2.2.3

- Added `TiffCompressionWebP` to `TiffCompression` enum

### 3.4 Metashape version 2.2.2

- Added `EulerAnglesUndefined` to `EulerAngles` enum
- Added `mat2rvec()` and `rvec2mat()` methods to `Utils` class
- Added `Trajectory.interpolate()` method
- Added `Trajectory.sampling_interval` attribute
- Added `Orthomosaic.Patch.inpaint` attribute
- Added `use_occlusion_texture` and `use_point_cloud_intensity` arguments to `Chunk.buildOrthomosaic()` method

- Added `save_rotation`, `save_location_accuracy`, `save_rotation_accuracy`, `save_errors`, `save_estimated`, `save_variance` and `save_enabled` arguments to `Chunk.exportReference()` method
- Added `rotation_angles`, `load_rotation`, `load_location_accuracy`, `load_rotation_accuracy`, `load_enabled`, `column_enabled`, `column_label`, `column_x`, `column_y`, `column_z`, `column_sx`, `column_sy`, `column_sz`, `column_a`, `column_b`, `column_c`, `column_sa`, `column_sb` and `column_sc` arguments to `Chunk.importReference()` method
- Added `use_occlusion_texture` and `use_point_cloud_intensity` attributes to `BuildOrthomosaic` task
- Added `save_rotation`, `save_location_accuracy`, `save_rotation_accuracy`, `save_errors`, `save_estimated`, `save_variance` and `save_enabled` attributes to `ExportReference` class
- Added `rotation_angles`, `load_rotation`, `load_location_accuracy`, `load_rotation_accuracy`, `load_enabled`, `column_enabled`, `column_label`, `column_x`, `column_y`, `column_z`, `column_sx`, `column_sy`, `column_sz`, `column_a`, `column_b`, `column_c`, `column_sa`, `column_sb` and `column_sc` attributes to `ImportReference` class
- Added `apply_boresight_offset`, `apply_trajectory_mask` and `merge_orthophotos` attributes to `MergeAssets` class

### 3.5 Metashape version 2.2.1

- Added `CleanTiePoints` and `CleanModel` classes
- Added `TiePoints.Criterion` and `Model.Criterion` enums
- Added `alignCamerasByReference()`, `cleanTiePoints()` and `cleanModel()` methods to `Chunk` class
- Added `Model.cleanModel()` method
- Added `Trajectory.addTimeOffset()` method
- Added `Trajectory.antenna` attribute
- Added `location_fixed` and `rotation_fixed` attributes to `Antenna` class
- Added `DuplicateAsset.copy_patches` attribute
- Added `AlignCameras.align_laser_scans` attribute
- Added `ExportTiledModel.zip_compression` attribute
- Added `align_laser_scans` argument to `Chunk.alignCameras()` method
- Added `zip_compression` argument to `Chunk.exportTiledModel()` method
- Removed `ServiceMelown` from `ServiceType` enum

### 3.6 Metashape version 2.2.0

- Added `Geoid` class
- Added `PansharpenOrthomosaic`, `ClassifyOverlapPoints` and `ImportVideo` classes
- Added `MasksData` and `BlockModelData` to `DataSource` enum
- Added `MaskingModeAI` to `MaskingMode` enum
- Added `AprilTag16h5`, `AprilTag25h9`, `AprilTag36h10`, `AprilTag36h11`, `AprilTagCircle21h7`, `AprilTagStandard41h12` and `AprilTagStandard52h13` to `TargetType` enum
- Added `PointCloudFormatCSV` to `PointCloudFormat` enum
- Added `PointCloudViewAccuracy` to `ModelView.PointCloudViewMode` enum

- Added TiePointsViewImageCount to ModelView.TiePointsViewMode enum
- Added Document.addGeoid() method
- Added Chunk.pansharpenOrthomosaic() method
- Added clear() and copy() methods to Masks class
- Added PointCloud.classifyOverlapPoints() method
- Added listGeoids() and setContext() methods to CoordinateSystem class
- Added Document.geoids attribute
- Added mask\_sets attribute to Chunk class
- Added bias, bias\_acc, bias\_covariance, bias\_fixed and bias\_ref attributes to Antenna class
- Added key and label attributes to Masks class
- Added transfer\_texture attribute to BuildOrthomosaic class
- Added source\_data attribute to BuildTexture class
- Added replace\_asset and frames attributes to ColorizePointCloud, SmoothPointCloud, SmoothModel and RefineModel classes
- Added model attribute to RefineModel class
- Added copy\_masks attribute to DuplicateChunk and MergeChunks classes
- Added image\_path, convert\_to\_pinhole and save\_images attributes to ExportCameras class
- Added laser\_scans\_use\_initial\_orientation attribute to MatchPhotos class
- Added ExportMasks.masks attribute
- Added frames and replace\_asset attributes to GenerateMasks class
- Added columns, delimiter, generate\_panorama, group\_delimiters, load\_images, load\_point\_classification, load\_point\_color, load\_point\_confidence, load\_point\_index, load\_point\_intensity, load\_point\_normal, load\_point\_return\_number, load\_point\_scan\_angle, load\_point\_source\_id, load\_point\_timestamp and skip\_rows attributes to ImportPointCloud class
- Added ExportPointCloud.point\_clouds attribute
- Added transfer\_texture argument to Chunk.buildOrthomosaic() method
- Added source\_data argument to Chunk.buildTexture() method
- Added replace\_asset and frames arguments to Chunk.colorizePointCloud(), Chunk.smoothPointCloud(), Chunk.smoothModel() and Chunk.refineModel() methods
- Added model argument to Chunk.refineModel() method
- Added copy\_masks argument to Chunk.mergeChunks() method
- Added image\_path, save\_images and convert\_to\_pinhole arguments to Chunk.exportCameras() method
- Added laser\_scans\_use\_initial\_orientation argument to Chunk.matchPhotos() method
- Added point\_clouds argument to Chunk.exportPointCloud() method
- Added replace\_asset and frames arguments to Chunk.generateMasks() method
- Added columns, delimiter, generate\_panorama, group\_delimiters, load\_images, load\_point\_classification, load\_point\_color, load\_point\_confidence, load\_point\_index, load\_point\_intensity, load\_point\_normal, load\_point\_return\_number, load\_point\_scan\_angle, load\_point\_source\_id, load\_point\_timestamp and skip\_rows arguments to Chunk.importPointCloud() method

- Added progress argument to `Chunk.importVideo()` method
- Added `matcap_image`, `smooth_normals` and `color` arguments to `Model.renderImage()` method
- Added `smooth_normals` argument to `Model.renderNormalMap()` method
- Added `matcap_image`, `smooth_normals` and `color` arguments to `TiledModel.renderImage()` method
- Added `smooth_normals` argument to `TiledModel.renderNormalMap()` method
- Changed units for `time_start` and `time_end` arguments to seconds in `Chunk.importVideo()` method
- Changed default value of `clip_to_region` attribute to `False` for `ExportPointCloud`, `FilterPointCloud` and `SmoothPointCloud` classes
- Changed default value of `clip_to_region` argument to `False` in `Chunk.exportPointCloud()`, `Chunk.filterPointCloud()` and `Chunk.smoothPointCloud()` methods
- Renamed `BuildTexture.source_model` attribute to `source_asset`
- Removed `NetworkClient.setMasterServer()` method
- Removed `NetworkTask.encode()` method
- Removed `ExportPointCloud.point_cloud` attribute
- Removed `ignore_normals` and `import_images` attributes from `ImportPointCloud` class
- Removed `point_cloud` argument from `Chunk.exportPointCloud()` method
- Removed `ignore_normals` and `import_images` arguments from `Chunk.importPointCloud()` method

### 3.7 Metashape version 2.1.4

- Added `PointCloud.generatePanorama()` method
- Added `enabled` and `selected` attributes to `Model` class
- Added `key` and `selected` attributes to `CameraTrack` class
- Added `block_margin` and `clip_to_block` attributes to `ExportModel` class
- Added `group_delimiters` attribute to `ImportTrajectory` class
- Added `clip_to_block` and `block_margin` arguments to `Chunk.exportModel()` method
- Added `group_delimiters` argument to `Chunk.importTrajectory()` method

### 3.8 Metashape version 2.1.3

- Added `Trajectory` class
- Added `PointCloud.Point`, `PointCloud.Points` and `PointCloud.Reader` classes
- Added `Elevation.Patch` and `Elevation.Patches` classes
- Added `Application.PhotoView` class
- Added `CamerasFormatColmap` to `CamerasFormat` enum
- Added `addModelGroup()`, `addTrajectory()`, `findModelGroup()` and `findTrajectory()` methods to `Chunk` class
- Added `invertSelection()` method to `TiePoints`, `PointCloud` and `Model` classes

- Added extent() and selectPointsByRegion() methods to PointCloud class
- Added altitudeSlopeAspect(), coverageArea() and update() methods to Elevation class
- Added camera() and coverageArea() methods to Orthomosaic class
- Added trajectory and trajectories attributes to Chunk class
- Added palette\_absolute\_values and patches attributes to Elevation class
- Added bands and data\_type attributes to Model.Texture class
- Added block\_index and block\_region attributes to Model class
- Added point\_count\_by\_class attribute to PointCloud class
- Added resolution attribute to TiledModel class
- Added show\_basemap, show\_cameras, show\_elevation, show\_laser\_scans, show\_markers, show\_orthomosaic, show\_shapes and show\_trajectory attributes to ModelView class
- Added center, width, height, projection and scale attributes to OrthoView class
- Added Application.photo\_view attribute
- Added merge\_markers attribute to DetectMarkers class
- Added copy\_orthophotos attribute to DuplicateAsset class
- Added copy\_laser\_scans, cameras and laser\_scans attributes to DuplicateChunk class
- Added cameras, image\_compression and save\_masks attributes to ExportCameras class
- Added logo\_path attribute to ExportReport class
- Added clip\_to\_boundary and copy\_orthophotos attributes to TransformRaster class
- Added cameras and laser\_scans arguments to Chunk.copy() method
- Added merge\_markers argument to Chunk.detectMarkers() method
- Added save\_masks, image\_compression, cameras and chan\_rotation\_order arguments to Chunk.exportCameras() method
- Added logo\_path argument to Chunk.exportReport() method
- Added clip\_to\_boundary and copy\_orthophotos arguments to Chunk.transformRaster() method

### 3.9 Metashape version 2.1.2

- Added ServiceNira and ServiceAgisoftCloud to ServiceType enum
- Added ReferenceItemsAll to ReferenceItems enum
- Added Chunk.convertImages() method
- Added alignNormals(), invertNormals() and setPointReturnsFilter() methods to PointCloud class
- Added ModelGroup.renderPreview() method
- Added Camera.point\_cloud attribute
- Added borrowed, floating and rehostable attributes to License class
- Added clip\_to\_region attribute to ExportModel, ExportTiledModel, ExportPointCloud, FilterPointCloud, SmoothPointCloud and DuplicateAsset classes

- Added `save_absolute_paths` attribute to `ExportCameras` class
- Added `save_elevation` attribute to `ExportShapes` class
- Added `project_id` and `upload_images` attributes to `PublishData` class
- Added `clip_to_region` argument to `Chunk.exportModel()`, `Chunk.exportTiledModel()`, `Chunk.exportPointCloud()`, `Chunk.filterPointCloud()` and `Chunk.smoothPointCloud()` methods
- Added `save_absolute_paths` argument to `Chunk.exportCameras()` method
- Added `save_elevation` argument to `Chunk.exportShapes()` method
- Added `project_id` and `upload_images` arguments to `Chunk.publishData()` method
- Added `only_visible` argument to `PointCloud.selectMaskedPoints()` class

### 3.10 Metashape version 2.1.1

- Added `Document.sortChunks()` method
- Added `Model.setVertexColors()` method
- Added `key` attribute to `CameraGroup`, `MarkerGroup` and `ScalebarGroup` classes
- Added `BuildTexture.anti_aliasing` attribute
- Added `ExportModel.glTF_y_up` attribute
- Added `anti_aliasing` argument to `Chunk.buildTexture()` method
- Added `glTF_y_up` argument to `Chunk.exportModel()` method

### 3.11 Metashape version 2.1.0

- Added `Component` and `ModelGroup` classes
- Added `TrajectoryData`, `LaserScansData` and `DepthMapsAndLaserScansData` to `DataSource` enum
- Added `PointCloudFormatCOPC` to `PointCloudFormat` enum
- Added `ModelViewElevation` to `ModelView.ModelViewMode` enum
- Added `TiePointsViewElevation` to `ModelView.TiePointsViewMode` enum
- Added `TiledModelViewElevation` to `ModelView.TiledModelViewMode` enum
- Added `Chunk.mergeComponents()` and `Chunk.splitComponents()` methods
- Added `Elevation.pickPoint()` method
- Added `ModelView.captureVideo()` method
- Added `Camera.component` attribute
- Added `loop` and `smooth` attributes to `CameraTrack` class
- Added `component`, `components`, `model_group` and `model_groups` attributes to `Chunk` class
- Added `crs`, `group` and `transform` attributes to `Model` class
- Added `PointCloud.component` attribute

- Added `replace_asset` and `frames` attributes to `BuildModel`, `BuildTiledModel`, `BuildPointCloud`, `BuildDem`, `BuildOrthomosaic`, `DecimateModel`, `FilterPointCloud`, `ImportRaster` and `TransformRaster` classes
- Added `split_in_blocks`, `blocks_crs`, `blocks_size`, `blocks_origin`, `clip_to_boundary`, `export_blocks`, `build_texture` and `output_folder` attributes to `BuildModel` class
- Added `workitem_size_cameras` and `max_workgroup_size` attributes to `BuildTexture` class
- Added `BuildUV.pixel_size` attribute
- Added `ClassifyGroundPoints.max_terrain_slope` attribute
- Added `ExportPointCloud.tileset_version` attribute
- Added `ExportRaster.asset` attribute
- Added `model` attribute to `ColorizeModel` and `SmoothModel` classes
- Added `tiled_model`, `tileset_version`, `model_group`, `pixel_size`, `tile_size` and `face_count` attributes to `ExportTiledModel` class
- Added `replace_asset` and `frame_paths` attributes to `ImportModel` class
- Added `match_laser_scans`, `downscale_3d`, `keypoint_limit_3d` and `laser_scans_vertical_axis` attributes to `MatchPhotos` class
- Added `classes` and `apply_to_selection` attributes to `SmoothPointCloud` class
- Added `ImportPointCloud.ignore_normals` attribute
- Added `replace_asset` and `frames` arguments to `Chunk.buildModel()`, `Chunk.buildTiledModel()`, `Chunk.buildPointCloud()`, `Chunk.buildDem()`, `Chunk.buildOrthomosaic()`, `Chunk.decimateModel()`, `Chunk.filterPointCloud()`, `Chunk.importRaster()` and `Chunk.transformRaster()` methods
- Added `replace_asset` and `frame_paths` arguments to `Chunk.importModel()` method
- Added `split_in_blocks`, `blocks_crs`, `blocks_size`, `blocks_origin`, `clip_to_boundary`, `export_blocks`, `build_texture` and `output_folder` arguments to `Chunk.buildModel()` method
- Added `workitem_size_cameras` and `max_workgroup_size` arguments to `Chunk.buildTexture()` method
- Added `pixel_size` argument to `Chunk.buildUV()` method
- Added `max_terrain_slope` argument to `Chunk.classifyGroundPoints()` method
- Added `tileset_version` argument to `Chunk.exportPointCloud()` method
- Added `asset` argument to `Chunk.exportRaster()` method
- Added `model` argument to `Chunk.colorizeModel()` and `Chunk.smoothModel()` methods
- Added `tiled_model`, `tileset_version`, `model_group`, `pixel_size`, `tile_size` and `face_count` arguments to `Chunk.exportTiledModel()` method
- Added `match_laser_scans`, `downscale_3d`, `keypoint_limit_3d` and `laser_scans_vertical_axis` arguments to `Chunk.matchPhotos()` method
- Added `classes` and `apply_to_selection` arguments to `Chunk.smoothPointCloud()` method
- Added `ignore_normals` argument to `Chunk.importPointCloud()` method
- Added `publish` argument to `CloudClient.uploadProject()` method
- Replaced `ExportTiledModel.use_rtc_center` attribute with `use_tileset_transform`
- Replaced `use_rtc_center` argument in `Chunk.exportTiledModel()` method with `use_tileset_transform`
- Renamed `RefineMesh` class to `RefineModel`

- Renamed `Chunk.refineMesh()` method to `refineModel()`
- Renamed `Model.transform()` method to `transformVertices()`
- Renamed `NetworkClient.serverInfo()` method to `serverVersion()`
- Renamed `NetworkClient.serverStatus()` method to `serverInfo()`
- Renamed `NetworkClient.batchStatus()` method to `batchInfo()`
- Renamed `NetworkClient.dumpBatches()` method to `exportBatches()`
- Renamed `NetworkClient.loadBatches()` method to `importBatches()`
- Renamed `NetworkClient.setBatchNodeLimit()` method to `setBatchWorkerLimit()`
- Renamed `NetworkClient.nodeList()` method to `workerList()`
- Renamed `NetworkClient.nodeStatus()` method to `workerInfo()`
- Renamed `NetworkClient.quitNode()` method to `quitWorker()`
- Renamed `NetworkClient.abortNode()` method to `abortWorker()`
- Renamed `NetworkClient.setNodeCPUEnable()` method to `setWorkerCpuEnabled()`
- Renamed `NetworkClient.setNodeCapability()` method to `setWorkerCapability()`
- Renamed `NetworkClient.setNodeGPUMask()` method to `setWorkerGpuMask()`
- Renamed `NetworkClient.setNodePaused()` method to `setWorkerPaused()`
- Renamed `NetworkClient.setNodePriority()` method to `setWorkerPriority()`
- Renamed `NetworkTask.supports_gpu` attribute to `gpu_support`
- Renamed `supports_gpu` attribute to `gpu_support` in task classes
- Renamed `DecimateModel.asset` attribute to `model`
- Renamed `TransformRaster.data_source` attribute to `source_data`
- Renamed `RenderDepthMaps.export_depth`, `export_diffuse` and `export_normals` attributes to `save_depth`, `save_diffuse` and `save_normals`
- Renamed `asset` argument in `Chunk.decimateModel()` method to `model`
- Renamed `data_source` argument in `Chunk.transformRaster()` method to `source_data`
- Added `.pyi` stub file to stand-alone Python module for autocompletion in external IDEs

## 3.12 Metashape version 2.0.4

- Added `borrowLicense()` and `returnLicense()` methods to `License` class
- Added `removeTextures()`, `removeUV()`, `removeVertexColors()` and `removeVertexConfidence()` methods to `Model` class
- Added `License.expiration` attribute
- Added `publish` argument to `CloudClient.uploadProject()` method
- Added `format` argument to `RPCModel.load()` and `RPCModel.save()` methods

### 3.13 Metashape version 2.0.3

- Added SmoothPointCloud class
- Added Chunk.smoothPointCloud() method
- Added enabled and selected attributes to PointCloud class
- Added mask\_dark\_pixels and frame\_detector attributes to DetectFiducials class
- Added mask\_dark\_pixels and frame\_detector arguments to Chunk.detectFiducials() method

### 3.14 Metashape version 2.0.2

- Added PointCloudGroup class
- Added TiledModelFormat3MX to TiledModelFormat enum
- Added Chunk.addPointCloudGroup() and Chunk.findPointCloudGroup() methods
- Added Chunk.point\_cloud\_groups attribute
- Added PointCloud.group and PointCloud.is\_laser\_scan attributes

### 3.15 Metashape version 2.0.1

- Added License.install() method
- Added DetectFiducials.v\_shape\_detector attribute
- Added model and save\_metadata\_xml attributes to ExportModel task
- Added v\_shape\_detector argument to Chunk.detectFiducials() method
- Added model and save\_metadata\_xml arguments to Chunk.exportModel() method
- Replaced license\_key argument with activation\_params in License.activateOffline() method

### 3.16 Metashape version 2.0.0

- Added TrajectoryFormat enum
- Added DisplacementMap to Model.TextureType enum
- Added ImportTrajectory class
- Added ImportDepthImages class
- Added Chunk.importTrajectory() method
- Added Chunk.importDepthImages() method
- Added AlignCameras.point\_clouds attribute
- Added ImportDepthImages.color\_filenames attribute
- Added precision, is\_laser\_scan, replace\_asset, import\_images, scanner\_at\_origin, ignore\_scanner\_origin, ignore\_trajectory, trajectory and frame\_paths attributes to ImportPointCloud class

- Added `keep_existing`, `return_number` and `point_cloud` attributes to `ClassifyGroundPoints` class
- Added `point_cloud` attribute to `ClassifyPoints`, `ColorizePointCloud`, `CalculatePointNormals`, `CompactPointCloud` and `ExportPointCloud` classes
- Added `max_quantization_error` attribute to `DetectPowerlines` class
- Added `use_rtc_center` attribute to `ExportTiledModel` class
- Added `merge_assets`, `copy_laser_scans`, `copy_depth_maps`, `copy_point_clouds`, `copy_models`, `copy_tiled_models`, `copy_elevations` and `copy_orthomosaics` attributes to `MergeChunks` class
- Added `point_clouds` argument to `Chunk.alignCameras()` method
- Added `color_filenames` argument to `Chunk.importDepthImages()` method
- Added `precision`, `is_laser_scan`, `replace_asset`, `import_images`, `scanner_at_origin`, `ignore_scanner_origin`, `ignore_trajectory`, `trajectory` and `frame_paths` arguments to `Chunk.importPointCloud()` method
- Added `point_cloud` argument to `Chunk.calculatePointNormals()`, `Chunk.colorizePointCloud()` and `Chunk.exportPointCloud()` methods
- Added `max_quantization_error` argument to `Chunk.detectPowerlines()` method
- Added `keep_existing` and `return_number` arguments to `PointCloud.classifyGroundPoints()` method
- Added `use_rtc_center` argument to `Chunk.exportTiledModel()` method
- Added `merge_assets`, `copy_laser_scans`, `copy_depth_maps`, `copy_point_clouds`, `copy_models`, `copy_tiled_models`, `copy_elevations` and `copy_orthomosaics` arguments to `Document.mergeChunks()` method
- Added `drone_name`, `payload_name` and `payload_position` arguments to `CameraTrack.save()` method
- Change default `source_data` argument value for `Chunk.buildModel()` and `Chunk.buildTiledModel()` methods to `DepthMapsData`
- Renamed `PointsFormat` enum to `PointCloudFormat`
- Renamed `ModelView.PointCloudViewMode` enum to `ModelView.TiePointsViewMode`
- Renamed `ModelView.DenseCloudViewMode` enum to `ModelView.PointCloudViewMode` and added `PointCloudViewSolid`, `PointCloudViewIntensity`, `PointCloudViewElevation`, `PointCloudViewReturnNumber`, `PointCloudViewScanAngle`, `PointCloudViewSourceId` enumeration values
- Renamed `DataSource.PointCloudData` enum value to `DataSource.TiePointsData`
- Renamed `DataSource.DenseCloudData` enum value to `DataSource.PointCloudData`
- Renamed `PointCloud` class to `TiePoints`
- Renamed `DenseCloud` class to `PointCloud`
- Renamed `AnalyzePhotos` class to `AnalyzeImages`
- Renamed `BuildDenseCloud` class to `BuildPointCloud`
- Renamed `CalibrateLens` class to `CalibrateCamera`
- Renamed `ColorizeDenseCloud` class to `ColorizePointCloud`
- Renamed `CompactDenseCloud` class to `CompactPointCloud`
- Renamed `ExportDepth` class to `RenderDepthMaps`
- Renamed `ExportPoints` class to `ExportPointCloud`
- Renamed `FilterDenseCloud` class to `FilterPointCloud`
- Renamed `ImportPoints` class to `ImportPointCloud`

- Renamed TriangulatePoints class to TriangulateTiePoints
- Renamed Chunk.addDenseCloud() method to addPointCloud()
- Renamed Chunk.analyzePhotos() method to analyzeImages()
- Renamed Chunk.buildDenseCloud() method to buildPointCloud()
- Renamed Chunk.colorizeDenseCloud() method to colorizePointCloud()
- Renamed Chunk.exportPoints() method to exportPointCloud()
- Renamed Chunk.filterDenseCloud() method to filterPointCloud()
- Renamed Chunk.findDenseCloud() method to findPointCloud()
- Renamed Chunk.importPoints() method to importPointCloud()
- Renamed Chunk.thinPointCloud() method to thinTiePoints()
- Renamed Chunk.triangulatePoints() method to triangulateTiePoints()
- Renamed Chunk.point\_cloud attribute to tie\_points
- Renamed Chunk.dense\_cloud attribute to point\_cloud
- Renamed Chunk.dense\_clouds attribute to point\_clouds
- Renamed ModelView.point\_cloud\_view\_mode attribute to tie\_points\_view\_mode
- Renamed ModelView.dense\_cloud\_view\_mode attribute to point\_cloud\_view\_mode
- Renamed AddFrames.copy\_dense\_cloud attribute to copy\_point\_cloud
- Renamed DuplicateChunk.copy\_dense\_clouds attribute to copy\_point\_clouds
- Renamed FilterPointCloud.asset attribute to point\_cloud
- Renamed PublishData.save\_point\_colors attribute to save\_point\_color
- Renamed copy\_dense\_cloud argument in Chunk.addFrames() method to copy\_point\_cloud
- Renamed save\_point\_colors argument in Chunk.publishData() method to save\_point\_color
- Renamed asset argument in Chunk.filterPointCloud() method to point\_cloud
- Renamed source argument in PointCloud.classifyGroundPoints() method to source\_class
- Revised parameter names for point attributes in ExportPointCloud class and Chunk.exportPointCloud() methods
- Removed ImportLaserScans class
- Removed Chunk.importLaserScans() method
- Removed Chunk.samplePoints() method
- Removed use\_trajectory, traj\_path, traj\_columns, traj\_delimiter and traj\_skip\_rows attributes from ImportPointCloud class
- Removed use\_trajectory, traj\_path, traj\_columns, traj\_delimiter and traj\_skip\_rows arguments from Chunk.importPointCloud() method
- Removed merge\_depth\_maps, merge\_dense\_clouds, merge\_models, merge\_elevations and merge\_orthomosaics attributes from MergeChunks class
- Removed merge\_depth\_maps, merge\_dense\_clouds, merge\_models, merge\_elevations and merge\_orthomosaics arguments from Document.mergeChunks() method

### 3.17 Metashape version 1.8.5

- Added DetectPowerlines class
- Added Chunk.detectPowerlines() method
- Added CameraTrack.interpolate() method
- Added generic\_detector, right\_angle\_detector, fiducials\_position\_corners and fiducials\_position\_sides attributes to DetectFiducials class
- Added archive attribute to LoadProject and SaveProject classes
- Added generic\_detector, right\_angle\_detector, fiducials\_position\_corners and fiducials\_position\_sides arguments to Chunk.detectFiducials() method
- Added archive argument to Document.open() and Document.save() methods

### 3.18 Metashape version 1.8.4

- Added Shutter.Model enum
- Added ImageFormatBZ2, ImageFormatASCII and ImageFormatKTX to ImageFormat enum
- Added Shape.areaFitted() method
- Added ExportPoints.folder\_depth and ExportTiledModel.folder\_depth attributes
- Added ImportLaserScans.multipane attribute
- Added folder\_depth argument to Chunk.exportPoints() and Chunk.exportTiledModel() methods
- Added multipane argument to Chunk.importLaserScans() method
- Changed type of Sensor.rolling\_shutter attribute to Shutter.Model
- Fixed Antenna.location and Antenna.rotation attributes to return non-None values

### 3.19 Metashape version 1.8.3

- Added CloudClient class
- Added PublishData class
- Added CalibrationFormatSTMap to CalibrationFormat enum
- Reorganized arguments of Chunk.publishData() method

## 3.20 Metashape version 1.8.2

No Python API changes

## 3.21 Metashape version 1.8.1

- Added CamerasFormatMA to CamerasFormat enum
- Added global\_profile attribute to ExportRaster class
- Added traj\_columns, traj\_delimiter, traj\_path, traj\_skip\_rows and use\_trajectory attributes to ImportPoints class
- Added global\_profile argument to Chunk.exportRaster() method
- Added use\_trajectory, traj\_path, traj\_columns, traj\_delimiter and traj\_skip\_rows arguments to Chunk.importPoints() method
- Removed fix\_pixel\_aspect, fix\_principal\_point, and remove\_distortions attributes from ConvertImages class

## 3.22 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

### 3.23 Metashape version 1.7.6

- Added Cylindrical to Sensor.Type enum

### 3.24 Metashape version 1.7.5

- Added ClassifyGroundPoints.erosion\_radius attribute
- Added erosion\_radius argument to DenseCloud.classifyGroundPoints() method

### 3.25 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

## 3.26 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

## 3.27 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

## 3.28 Metashape version 1.7.1

- Removed LegacyMapping from MappingMode enum
- Removed ReduceOverlap.sensor attribute
- Removed sensor argument from Chunk.reduceOverlap() method

## 3.29 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

### 3.30 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

### 3.31 Metashape version 1.6.5

- Added Sensor.meta attribute

### 3.32 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

### 3.33 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

### 3.34 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`

### 3.35 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`

### 3.36 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 Tiled-Model.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

### 3.37 Metashape version 1.5.5

No Python API changes

### 3.38 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

### 3.39 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

### 3.40 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

### 3.41 Metashape version 1.5.1

- Added License class
- Added Tasks.MergeAssets class
- Added Metashape.license attribute
- Renamed Tasks.OptimizeCoverage class to Tasks.PlanMotion

### 3.42 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

### 3.43 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

### 3.44 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

## 3.45 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

## 3.46 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

## 3.47 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

### 3.48 PhotoScan version 1.3.5

No Python API changes

### 3.49 PhotoScan version 1.3.4

No Python API changes

### 3.50 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

### 3.51 PhotoScan version 1.3.2

- Added `vertex_colors` argument to `Chunk.buildModel()` method
- Added `Shape.vertex_ids` attribute

### 3.52 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

### 3.53 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

### 3.54 PhotoScan version 1.2.6

No Python API changes

### 3.55 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

### 3.56 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

### 3.57 PhotoScan version 1.2.3

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

### 3.58 PhotoScan version 1.2.2

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

### 3.59 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`

## 3.60 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

## 3.61 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

## 3.62 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

### 3.63 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\_cxcy, 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

### 3.64 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

### 3.65 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)

### 3.66 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

### 3.67 PhotoScan version 0.8.4

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

### 3.68 PhotoScan version 0.8.3

Initial version of PhotoScan Python API



## PYTHON MODULE INDEX

m

Metashape, 5