Metashape Python Reference

Release 1.5.3

Agisoft LLC

Jun 08, 2019

Copyright (c) 2018 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 3.5 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 funtionality 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:

```python
>>> import Metashape
>>> doc = Metashape.app.document
>>> doc.open("project.psz")
>>> chunk = doc.chunk
>>> chunk.matchPhotos(accuracy=Metashape.HighAccuracy, generic_preselection=True,
→reference_preselection=False)
>>> chunk.alignCameras()
>>> chunk.buildDepthMaps(quality=Metashape.MediumQuality, filter=Metashape.
→AggressiveFiltering)
>>> chunk.buildDenseCloud()
>>> chunk.buildModel(surface=Metashape.Arbitrary, interpolation=Metashape.
→EnabledInterpolation)
>>> chunk.buildUV(mapping=Metashape.GenericMapping)
>>> chunk.buildTexture(blending=Metashape.MosaicBlending, size=4096)
>>> doc.save()
```

class `Metashape.`**`Accuracy`**
> Alignment accuracy in [HighestAccuracy, HighAccuracy, MediumAccuracy, LowAccuracy, LowestAccuracy]

class `Metashape.`**`Antenna`**
> GPS antenna position relative to camera.

> **`fixed`**
> > Fix antenna flag.
> >
> > > **Type** bool

> **`location`**
> > Antenna coordinates.
> >
> > > **Type** *Vector*

> **`location_acc`**
> > Antenna location accuracy.
> >
> > > **Type** *Vector*

location_covariance
>   Antenna location covariance.
>
>> **Type** *Matrix*

location_ref
>   Antenna location reference.
>
>> **Type** *Vector*

rotation
>   Antenna rotation angles.
>
>> **Type** *Vector*

rotation_acc
>   Antenna rotation accuracy.
>
>> **Type** *Vector*

rotation_covariance
>   Antenna rotation covariance.
>
>> **Type** *Matrix*

rotation_ref
>   Antenna rotation reference.
>
>> **Type** *Vector*

class Metashape.**Application**
>   Application class provides access to several global application attributes, such as document currently loaded in the user interface, software version and GPU device configuration. It also contains helper routines to prompt the user to input various types of parameters, like displaying a file selection dialog or coordinate system selection dialog among others.
>
>   An instance of Application object can be accessed using Metashape.app attribute, so there is usually no need to create additional instances in the user code.
>
>   The following example prompts the user to select a new coordinate system, applies it to the ative chunk and saves the project under the user selected file name:

```
>>> import Metashape
>>> doc = Metashape.app.document
>>> crs = Metashape.app.getCoordinateSystem("Select Coordinate System", doc.chunk.
↪crs)
>>> doc.chunk.crs = crs
>>> path = Metashape.app.getSaveFileName("Save Project As")
>>> try:
...     doc.save(path)
... except RuntimeError:
...     Metashape.app.messageBox("Can't save project")
```

>   **class ConsolePane**
>>   ConsolePane class provides access to the console pane
>>
>>   **clear**()
>>>   Clear console pane.
>>
>>   **contents**
>>>   Console pane contents.
>>>>   **Type** string

class Application.**PhotosPane**
> PhotosPane class provides access to the photos pane

> **resetFilter**()
> > Reset photos pane filter.

> **setFilter**(*items*)
> > Set photos pane filter.
> > > **Parameters items** (list of [*Camera*](#) or [*Marker*](#)) – filter to apply.

class Application.**Settings**
> PySettings()

> Application settings

> **load**()
> > Load settings from disk.

> **log_enable**
> > Enable writing log to file.
> > > **Type** bool

> **log_path**
> > Log file path.
> > > **Type** string

> **network_enable**
> > Network processing enabled flag.
> > > **Type** bool

> **network_host**
> > Network server host name.
> > > **Type** string

> **network_path**
> > Network data root path.
> > > **Type** string

> **network_port**
> > Network server control port.
> > > **Type** int

> **save**()
> > Save settings on disk.

> **setValue**(*key*, *value*)
> > Set settings value. :arg key: Key. :type key: string :arg value: Value. :type value: object

> **value**(*key*)
> > Return settings value. :arg key: Key. :type key: string :return: Settings value. :rtype: object

Application.**activated**
> Metashape activation status.

> > **Type** bool

Application.**addMenuItem**(*label*, *func*[, *shortcut*][, *icon*])
> Create a new menu entry.

> > **Parameters**

> > > • **label** (*string*) – Menu item label.

> > > • **func** (*function*) – Function to be called.

- **shortcut** (*string*) – Keyboard shortcut.

- **icon** (*string*) – Icon.

`Application.`**`addMenuSeparator`**(*label*)
    Add menu separator.

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

`Application.`**`captureModelView`**($\big[$*width*$\big]\big[$, *height*$\big]\big[$, *transparent*$\big]\big[$, *hide_items*$\big]\big[$, *source*$\big]\big[$, *mode*$\big]$)
    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.
>
> - **source** (*DataSource*) – Data source. Note: DataSource.DenseCloudData value is not supported.
>
> - **mode** (*ModelViewMode*) – Model view mode.
>
> **Returns** Captured image.
>
> **Return type** *Image*

`Application.`**`captureOrthoView`**($\big[$*width*$\big]\big[$, *height*$\big]\big[$, *transparent*$\big]\big[$, *hide_items*$\big]\big[$, *source*$\big]$)
    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.
>
> - **source** (*DataSource*) – Data source.
>
> **Returns** Captured image.
>
> **Return type** *Image*

`Application.`**`console`**
    Console pane.

> **Type** ConsolePane

`Application.`**`cpu_enable`**
    Use CPU when GPU is active.

> **Type** bool

`Application.`**`document`**
    Main application document object.

> **Type** *Document*

`Application.`**`enumGPUDevices`**()
    Enumerate installed GPU devices.

> **Returns** A list of devices.
>
> **Return type** list

Application.**getBool**(*label=''*)
> Prompt user for the boolean value.
>
> > **Parameters** **label** (*string*) – Optional text label for the dialog.
> >
> > **Returns** Boolean value selected by the user.
> >
> > **Return type** bool

Application.**getCoordinateSystem**([*label*][, *value*])
> Prompt user for coordinate system.
>
> > **Parameters**
> >
> > - **label** (*string*) – Optional text label for the dialog.
> >
> > - **value** (*CoordinateSystem*) – Default value.
> >
> > **Returns** Selected coordinate system. If the dialog was cancelled, None is returned.
> >
> > **Return type** *CoordinateSystem*

Application.**getExistingDirectory**([*hint*])
> Prompt user for the existing folder.
>
> > **Parameters** **hint** (*string*) – Optional text label for the dialog.
> >
> > **Returns** Path to the folder selected. If the input was cancelled, empty string is returned.
> >
> > **Return type** string

Application.**getFloat**(*label='', value=0*)
> Prompt user for the floating point value.
>
> > **Parameters**
> >
> > - **label** (*string*) – Optional text label for the dialog.
> >
> > - **value** (*float*) – Default value.
> >
> > **Returns** Floating point value entered by the user.
> >
> > **Return type** float

Application.**getInt**(*label='', value=0*)
> Prompt user for the integer value.
>
> > **Parameters**
> >
> > - **label** (*string*) – Optional text label for the dialog.
> >
> > - **value** (*int*) – Default value.
> >
> > **Returns** Integer value entered by the user.
> >
> > **Return type** int

Application.**getOpenFileName**([*hint*][, *filter*])
> Prompt user for the existing file.
>
> > **Parameters**
> >
> > - **hint** (*string*) – Optional text label for the dialog.

- **filter** (*string*) – Optional file filter, e.g. "Text file (*.txt)" or ".txt". Multiple filters are separated with ";;".

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

>   **Return type** string

Application.**getOpenFileNames**([*hint*][, *filter*])
>   Prompt user for one or more existing files.

>   **Parameters**

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

>   - **filter** (*string*) – Optional file filter, e.g. "Text file (*.txt)" or ".txt". Multiple filters are separated with ";;".

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

>   **Return type** list

Application.**getSaveFileName**([*hint*][, *filter*])
>   Prompt user for the file. The file does not have to exist.

>   **Parameters**

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

>   - **filter** (*string*) – Optional file filter, e.g. "Text file (*.txt)" or ".txt". Multiple filters are separated with ";;".

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

>   **Return type** string

Application.**getString**(*label=''*, *value=''*)
>   Prompt user for the string value.

>   **Parameters**

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

>   - **value** (*string*) – Default value.

>   **Returns** String entered by the user.

>   **Return type** string

Application.**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

Application.**messageBox**(*message*)
>   Display message box to the user.

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

Application.**photos_pane**
>   Photos pane.

>   **Type** PhotosPane

Application.**quit**()
>   Exit application.

Application.**settings**
>   Application settings.

>>> **Type** Settings

`Application.`**`title`**
>>> Application name.

>>> **Type** string

`Application.`**`update`**`()`
>>> Update user interface during long operations.

`Application.`**`version`**
>>> Metashape version.

>>> **Type** string

`Application.`**`viewpoint`**
>>> Viewpoint in the model view.

>>> **Type** *[Viewpoint](#)*

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

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

> **`b1`**
>>> Affinity.

>>> **Type** float

> **`b2`**
>>> Non-orthogonality.

>>> **Type** float

> **`covariance_matrix`**
>>> Covariance matrix.

>>> **Type** *[Matrix](#)*

> **`covariance_params`**
>>> Covariance matrix parameters.

>>> **Type** list of string

> **`cx`**
>>> Principal point X coordinate.

>>> **Type** float

> **`cy`**
>>> Principal point Y coordinate.

>>> **Type** float

> **`error`**(*point*, *proj*)
>>> Returns projection error.

>>> **Parameters**

>>>> • **`point`** ([*Vector*](#)) – Coordinates of the point to be projected.

>>>> • **`proj`** ([*Vector*](#)) – Pixel coordinates of the point.

>>> **Returns** 2D projection error.

> **Return type** *Vector*

f
Focal length.

> **Type** float

height
Image height.

> **Type** int

k1
Radial distortion coefficient K1.

> **Type** float

k2
Radial distortion coefficient K2.

> **Type** float

k3
Radial distortion coefficient K3.

> **Type** float

k4
Radial distortion coefficient K4.

> **Type** float

load(*path*, *format='xml'*)
Loads calibration from file.

> **Parameters**
>
> - **path** (*string*) – path to calibration file
> - **format** (*string*) – Calibration format in ['xml', 'australis', 'photomodeler', 'calib-cam', 'calcam', 'inpho', 'usgs', 'opencv'].
>
> **Returns** success of operation
>
> **Return type** bool

p1
Tangential distortion coefficient P1.

> **Type** float

p2
Tangential distortion coefficiant P2.

> **Type** float

p3
Tangential distortion coefficient P3.

> **Type** float

p4
Tangential distortion coefficiant P4.

> **Type** float

project(*point*)

Returns projected pixel coordinates of the point.

> **Parameters point** (*Vector*) – Coordinates of the point to be projected.
>
> **Returns** 2D projected point coordinates.
>
> **Return type** *Vector*

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

Saves calibration to file.

> **Parameters**
>
> - **path** (*string*) – path to calibration file
> - **format** (*string*) – Calibration format in ['xml', 'australis', 'photomodeler', 'calib-cam', 'calcam', 'inpho', 'usgs', 'opencv'].
> - **pixel_size** (*Vector*) – Pixel size in mm used to convert normalized calibration coefficients to Australis and CalibCam coefficients.
> - **label** (*string*) – Calibration label used in Australis, CalibCam and CalCam formats.
>
> **Returns** success of operation
>
> **Return type** bool

type

Camera model.

> **Type** *Sensor.Type*

unproject(*point*)

Returns direction corresponding to the image point.

> **Parameters point** (*Vector*) – Pixel coordinates of the point.
>
> **Returns** 3D vector in the camera coordinate system.
>
> **Return type** *Vector*

width

Image width.

> **Type** int

class 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 multispectal camera layout:

```
>>> import Metashape
>>> doc = Metashape.app.document
>>> chunk = doc.chunk
>>> rgb = ["RGB_0001.JPG", "RGB_0002.JPG", "RGB_0003.JPG"]
>>> nir = ["NIR_0001.JPG", "NIR_0002.JPG", "NIR_0003.JPG"]
```

```
>>> images = [[rgb[0], nir[0]], [rgb[1], nir[1]], [[rgb[2], nir[2]]
>>> chunk.addPhotos(images, Metashape.MultiplaneLayout)
```

class **Reference**
    Camera reference data.

    accuracy
        Camera location accuracy.
            Type *Vector*

    enabled
        Location enabled flag.
            Type bool

    location
        Camera coordinates.
            Type *Vector*

    location_accuracy
        Camera location accuracy.
            Type *Vector*

    location_enabled
        Location enabled flag.
            Type bool

    rotation
        Camera rotation angles.
            Type *Vector*

    rotation_accuracy
        Camera rotation accuracy.
            Type *Vector*

    rotation_enabled
        Rotation enabled flag.
            Type bool

class Camera.**Type**
    Camera type in [Regular, Keyframe]

Camera.**calibration**
    Adjusted camera calibration including photo-invariant parameters.

        Type *Calibration*

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

        Type *Vector*

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

        Type *Chunk*

Camera.**enabled**
    Enables/disables the photo.

        Type bool

`Camera.``error`(*point*, *proj*)
> Returns projection error.

>> **Parameters**

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

>>> • `proj` (*Vector*) – Pixel coordinates of the point.

>> **Returns** 2D projection error.

>> **Return type** *Vector*

`Camera.``frames`
> Camera frames.

>> **Type** list of *Camera*

`Camera.``group`
> Camera group.

>> **Type** *CameraGroup*

`Camera.``image`()
> Returns image data.

>> **Returns** Image data.

>> **Return type** *Image*

`Camera.``key`
> Camera identifier.

>> **Type** int

`Camera.``label`
> Camera label.

>> **Type** string

`Camera.``layer_index`
> Camera layer index.

>> **Type** int

`Camera.``location_covariance`
> Camera location covariance.

>> **Type** *Matrix*

`Camera.``mask`
> Camera mask.

>> **Type** *Mask*

`Camera.``master`
> Master camera.

>> **Type** *Camera*

`Camera.``meta`
> Camera meta data.

>> **Type** *MetaData*

`Camera.``open`(*path*[, *layer*])
> Loads specified image file.

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

Camera.**orientation**
> Image orientation (1 - normal, 6 - 90 degree, 3 - 180 degree, 8 - 270 degree).
>
> > **Type** int

Camera.**photo**
> Camera photo.
>
> > **Type** *Photo*

Camera.**planes**
> Camera planes.
>
> > **Type** list of *Camera*

Camera.**project**(*point*)
> Returns coordinates of the point projection on the photo.
>
> > **Parameters** **point** (*Vector*) – Coordinates of the point to be projected.
> >
> > **Returns** 2D point coordinates.
> >
> > **Return type** *Vector*

Camera.**reference**
> Camera reference data.
>
> > **Type** CameraReference

Camera.**rotation_covariance**
> Camera rotation covariance.
>
> > **Type** *Matrix*

Camera.**selected**
> Selects/deselects the photo.
>
> > **Type** bool

Camera.**sensor**
> Camera sensor.
>
> > **Type** *Sensor*

Camera.**shutter**
> Camera shutter.
>
> > **Type** *Shutter*

Camera.**thumbnail**
> Camera thumbnail.
>
> > **Type** *Thumbnail*

Camera.**transform**
> 4x4 matrix describing photo location in the chunk coordinate system.
>
> > **Type** *Matrix*

Camera.**type**
> Camera type.

> > **Type** *Camera.Type*

Camera.**unproject**(*point*)
> Returns coordinates of the point which will have specified projected coordinates.

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

> > **Returns** 3D point coordinates.

> > **Return type** *Vector*

Camera.**vignetting**
> Vignetting for each band.

> > **Type** list of *Vignetting*

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

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

> **class Type**
> > Camera group type in [Folder, Station]

> CameraGroup.**label**
> > Camera group label.

> > > **Type** string

> CameraGroup.**selected**
> > Current selection state.

> > > **Type** bool

> CameraGroup.**type**
> > Camera group type.

> > > **Type** *CameraGroup.Type*

class Metashape.**CameraTrack**
> Camera track.

> **duration**
> > Animation duration.

> > > **Type** float

> **field_of_view**
> > Vertical field of view in degrees.

> > > **Type** float

> **keyframes**
> > Camera track keyframes.

> > > **Type** list of *Camera*

> **label**
> > Animation label.

> > > **Type** string

class `Metashape.`**`CamerasFormat`**

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

class `Metashape.`**`Chunk`**

    A Chunk object:

> •provides access to all chunk components (sensors, cameras, camera groups, markers, scale bars)
>
> •contains data inherent to individual frames (point cloud, model, etc)
>
> •implements processing methods (matchPhotos, alignCameras, buildDenseCloud, buildModel, etc)
>
> •provides access to other chunk attributes (transformation matrix, coordinate system, meta-data, etc..)

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

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

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

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

    `addCamera`([*sensor*])

        Add new camera to the chunk.

> **Parameters** **sensor** (*Sensor*) – Sensor to be assigned to this camera.
>
> **Returns** Created camera.
>
> **Return type** *Camera*

    `addCameraGroup`()

        Add new camera group to the chunk.

> **Returns** Created camera group.
>
> **Return type** *CameraGroup*

    `addCameraTrack`()

        Add new camera track to the chunk.

> **Returns** Created camera track.
>
> **Return type** *CameraTrack*

    `addDenseCloud`()

        Add new dense cloud to the chunk.

> **Returns** Created dense cloud.
>
> **Return type** *DenseCloud*

addDepthMaps()
> Add new depth maps set to the chunk.

>> **Returns** Created depth maps set.

>> **Return type** *DepthMaps*

addElevation()
> Add new elevation model to the chunk.

>> **Returns** Created elevation model.

>> **Return type** *Elevation*

addFrame()
> Add new frame to the chunk.

>> **Returns** Created frame.

>> **Return type** Frame

addFrames(*chunk*[, *frames*][, *items*][, *progress*])
> Add frames from specified chunk.

>> **Parameters**

>>> • **chunk** (*Chunk*) – Chunk to copy frames from.

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

>>> • **items** (list of *DataSource*) – A list of items to copy.

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

addMarker([*point*], *visibility=False*)
> Add new marker to the chunk.

>> **Parameters**

>>> • **point** (*Vector*) – Point to initialize marker projections.

>>> • **visibility** (*bool*) – Enables visibility check during projection assignment.

>> **Returns** Created marker.

>> **Return type** *Marker*

addMarkerGroup()
> Add new marker group to the chunk.

>> **Returns** Created marker group.

>> **Return type** *MarkerGroup*

addModel()
> Add new model to the chunk.

>> **Returns** Created model.

>> **Return type** *Model*

addOrthomosaic()
> Add new orthomosaic to the chunk.

>> **Returns** Created orthomosaic.

>> **Return type** *Orthomosaic*

addPhotos (*filenames* $\big[$, *layout* $\big]$, *strip_extensions=True* $\big[$, *progress* $\big]$)
    Add a list of photos to the chunk.

    Parameters

- **filenames** (`list of string`) – A list of file paths.
- **layout** (`ImageLayout`) – Image layout in the chunk.
- **strip_extensions** (`bool`) – Strip file extensions from camera labels.
- **progress** (`Callable[[float], None]`) – Progress callback.

addScalebar (*point1*, *point2*)
    Add new scale bar to the chunk.

    Parameters

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

    Returns Created scale bar.

    Return type `Scalebar`

addScalebarGroup ()
    Add new scale bar group to the chunk.

    Returns Created scale bar group.

    Return type `ScalebarGroup`

addSensor ()
    Add new sensor to the chunk.

    Returns Created sensor.

    Return type `Sensor`

addTiledModel ()
    Add new tiled model to the chunk.

    Returns Created tiled model.

    Return type `TiledModel`

alignCameras ($\big[$*cameras* $\big]$ $\big[$, *min_image* $\big]$, *adaptive_fitting=False* $\big[$, *progress* $\big]$)
    Perform photo alignment for the chunk.

    Parameters

- **cameras** (list of `Camera`) – A list of cameras to be aligned to the existing cameras.
- **min_image** (`int`) – Minimum number of point projections.
- **adaptive_fitting** (`bool`) – Enables adaptive fitting of distortion coefficients.
- **progress** (`Callable[[float], None]`) – Progress callback.

buildContours (*source_data=ElevationData*, *interval=1* $\big[$, *min_value* $\big]$ $\big[$, *max_value* $\big]$ $\big[$, *progress* $\big]$)
    Build contours for the chunk.

    Parameters

- **source_data** (`DataSource`) – Source data for contour generation.
- **interval** (`float`) – Contour interval.

- **min_value** (*float*) – Minimum value of contour range.

- **max_value** (*float*) – Maximum value of contour range.

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

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

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

buildDenseCloud(*point_colors=True*, *keep_depth=True*[, *max_neighbors*][, *progress*])
Generate dense cloud for the chunk.

> **Parameters**
>
> - **point_colors** (*bool*) – Enables/disables point colors calculation.
>
> - **keep_depth** (*bool*) – Enables keep depth maps option.
>
> - **max_neighbors** (*int*) – Maximum number of neighbor images to use for depth map filtering.
>
> - **progress** (*Callable[[float], None]*) – Progress callback.

buildDepthMaps(*quality=MediumQuality*, *filter=MildFiltering*[, *cameras*], *reuse_depth=True*[, *max_neighbors*][, *progress*])
Generate depth maps for the chunk.

> **Parameters**
>
> - **quality** (*Quality*) – Depth map quality.
>
> - **filter** (*FilterMode*) – Depth map filtering level.
>
> - **cameras** (list of *Camera*) – A list of cameras to be processed.
>
> - **reuse_depth** (*bool*) – Enables reuse depth maps option.
>
> - **max_neighbors** (*int*) – Maximum number of neighbor images to use for depth map generation.
>
> - **progress** (*Callable[[float], None]*) – Progress callback.

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

>   **Parameters**
>
>   - **surface** (*SurfaceType*) – Type of object to be reconstructed.
>
>   - **interpolation** (*Interpolation*) – Interpolation mode.
>
>   - **face_count** (*FaceCount* or int) – Target face count.
>
>   - **source** (*DataSource*) – Selects between dense point cloud, sparse point cloud and depth maps. If not specified, uses dense cloud if available.
>
>   - **classes** (list of *PointClass*) – List of dense point classes to be used for surface extraction.
>
>   - **vertex_colors** (*bool*) – Enables/disables vertex colors calculation.
>
>   - **volumetric_masks** (*bool*) – Enables/disables strict volumetric masking.
>
>   - **keep_depth** (*bool*) – Enables keep depth maps option.
>
>   - **progress** (*Callable[[float], None]*) – Progress callback.

buildOrthomosaic (*surface=ElevationData*, *blending=MosaicBlending*, *fill_holes=True*, *cull_faces=False*, *refine_seamlines=False*[, *projection*][, *region*][, *dx*][, *dy*], *flip_x=False*, *flip_y=False*, *flip_z=False*[, *progress*])
    Build orthomosaic for the chunk.

>   **Parameters**
>
>   - **surface** (*DataSource*) – Orthorectification surface.
>
>   - **blending** (*BlendingMode*) – Orthophoto blending mode.
>
>   - **fill_holes** (*bool*) – Enable hole filling.
>
>   - **cull_faces** (*bool*) – Enable back-face culling.
>
>   - **refine_seamlines** (*bool*) – Refine seamlines based on image content.
>
>   - **projection** (*OrthoProjection* or *CoordinateSystem* or *Matrix*) – Sets output projection.
>
>   - **region** (*tuple of 4 floats*) – Region to be exported in the (x0, y0, x1, y1) format.
>
>   - **dx** (*float*) – Pixel size in the X dimension in projected units.
>
>   - **dy** (*float*) – Pixel size in the Y dimension in projected units.
>
>   - **flip_x** (*bool*) – Flip X axis direction.
>
>   - **flip_y** (*bool*) – Flip X axis direction.
>
>   - **flip_z** (*bool*) – Flip X axis direction.
>
>   - **progress** (*Callable[[float], None]*) – Progress callback.

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

>   **Parameters**
>
>   - **error** (*float*) – Reprojection error threshold.

- **min_image** (*int*) – Minimum number of point projections.

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

buildSeamlines (*epsilon=1.5*[, *progress*])
    Generate shapes for orthomosaic seamlines.

    Parameters

- **epsilon** (*float*) – Contour simplificaion threshold.

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

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

    Parameters

- **blending** (*BlendingMode*) – Texture blending mode.

- **size** (*int*) – Texture size.

- **fill_holes** (*bool*) – Enables hole filling.

- **ghosting_filter** (*bool*) – Enables ghosting filter.

- **cameras** (list of *Camera*) – A list of cameras to be used for texturing.

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

buildTiledModel ([*pixel_size*], *tile_size=256*[, *source*], *face_count=4000*, *ghosting_filter=True*, *keep_depth=True*[, *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** (*DataSource*) – Selects between depth maps, dense point cloud and mesh. If not specified, uses dense cloud if available.

- **face_count** (*int*) – Number of faces per megapixel of texture resolution.

- **ghosting_filter** (*bool*) – Enables ghosting filter.

- **keep_depth** (*bool*) – Enables keep depth maps option.

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

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

    Parameters

- **mapping** (*MappingMode*) – Texture mapping mode.

- **count** (*int*) – Texture count.

- **adaptive_resolution** (*bool*) – Enable adaptive face detalization.

- **camera** (*Camera*) – Camera to be used for texturing in MappingCamera mode.

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

calibrateColors(*source_data=ModelData*,　*color_balance=False*[, 　*cameras*][, 　*frames*][, *progress*])
    Perform radiometric calibration.

>   **Parameters**

>   • **source_data** (*DataSource*) – Source data for calibration.

>   • **color_balance** (*bool*) – Turn color balance compensation on/off.

>   • **cameras** (list of *Camera*) – List of cameras to process.

>   • **frames** (list of `Frame`) – List of frames to process.

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

calibrateReflectance(*use_reflectance_panels=True*, *use_sun_sensor=False*[, *progress*])
    Calibrate reflectance factors based on calibration panels and/or sun sensor.

>   **Parameters**

>   • **use_reflectance_panels** (*bool*) – Use calibrated reflectance panels.

>   • **use_sun_sensor** (*bool*) – Apply irradiance sensor measurements.

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

camera_crs
    Coordinate system used for camera reference data.

>   **Type** *CoordinateSystem*

camera_groups
    List of camera groups in the chunk.

>   **Type** list of *CameraGroup*

camera_location_accuracy
    Expected accuracy of camera coordinates in meters.

>   **Type** *Vector*

camera_rotation_accuracy
    Expected accuracy of camera orientation angles in degrees.

>   **Type** *Vector*

camera_track
    Camera track.

>   **Type** *CameraTrack*

camera_tracks
    List of camera tracks in the chunk.

>   **Type** list of *CameraTrack*

cameras
    List of Regular and Keyframe cameras in the chunk.

>   **Type** list of *Camera*

cir_transform
    CIR calibration matrix.

>   **Type** *CirTransform*

copy ([*frames*] [, *items*] , *keypoints=True* [, *progress*])
    Make a copy of the chunk.

    Parameters

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

- **items** (list of [`DataSource`](#)) – A list of items to copy.

- **keypoints** (`bool`) – copy key points data.

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

    Returns Copy of the chunk.

    Return type [`Chunk`](#)

crs
    Coordinate system used for reference data.

    Type [`CoordinateSystem`](#)

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

    Parameters

- **face_count** (`int`) – Target face count.

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

dense_cloud
    Default dense point cloud for the current frame.

    Type [`DenseCloud`](#)

dense_clouds
    List of dense clouds for the current frame.

    Type list of [`DenseCloud`](#)

depth_maps
    Default depth maps set for the current frame.

    Type [`DepthMaps`](#)

depth_maps_sets
    List of depth maps sets for the current frame.

    Type list of [`DepthMaps`](#)

detectFiducials ([*progress*])
    Detect fiducial marks on film cameras.

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

detectMarkers (*type=TargetCircular12bit*, *tolerance=50*, *filter_mask=False*, *inverted=False*, *noparity=False*, *maximum_residual=5* [, *minimum_size*] [, *minimum_dist*] [, *cameras*] [, *progress*])
    Create markers from coded targets.

    Parameters

- **type** ([`TargetType`](#)) – Type of targets.

- **tolerance** (`int`) – Detector tolerance (0 - 100).

- **filter_mask** (`bool`) – Ignore masked image regions.

- **inverted** (*bool*) – Detect markers on black background.

- **noparity** (*bool*) – Disable parity checking.

- **maximum_residual** (*float*) – Maximum residual for non-coded targets in pixels..

- **minimum_size** (*int*) – Minimum target radius in pixels to be detected (CrossTarget type only).

- **minimum_dist** (*int*) – Minimum distance between targets in pixels (CrossTarget type only).

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

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

elevation
: Default elevation model for the current frame.

    Type *Elevation*

elevations
: List of elevation models for the current frame.

    Type list of *Elevation*

enabled
: Enables/disables the chunk.

    Type bool

estimateImageQuality ([*cameras*], *filter_mask=False* [, *progress*])
: Estimate image quality.

    Parameters

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

    - **filter_mask** (*bool*) – Constrain analyzed image region by mask.

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

euler_angles
: Euler angles triplet used for rotation reference.

    Type *EulerAngles*

exportCameras (*path*, *format=CamerasFormatXML* [, *projection*], *export_points=True*, *export_markers=False*, *use_labels=False*, *rotation_order=RotationOrderXYZ* [, *progress*])
: Export point cloud and/or camera positions.

    Parameters

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

    - **format** (*CamerasFormat*) – Export format.

    - **projection** (*CoordinateSystem*) – Output coordinate system.

    - **export_points** (*bool*) – Enables/disables export of automatic tie points.

    - **export_markers** (*bool*) – Enables/disables export of manual matching points.

    - **use_labels** (*bool*) – Enables/disables label based item identifiers.

    - **rotation_order** (*RotationOrder*) – Rotation order (CHAN format only)

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

exportDem(*path*[, *format*][, *image_format*], *raster_transform=RasterTransformNone*[, *projection*][, *region*][, *dx*][, *dy*][, *blockw*][, *blockh*][, *width*][, *height*][, *world_transform*], *nodata=-32767*, *write_kml=False*, *write_world=False*, *write_scheme=False*[, *image_description*], *tiff_big=False*, *tiff_tiled=True*, *tiff_overviews=True*, *network_links=True*[, *min_zoom_level*][, *max_zoom_level*][, *progress*])

    Export digital elevation model.

        Parameters

- **path** (`string`) – Path to output DEM.

- **format** (`RasterFormat`) – Export format.

- **image_format** (`ImageFormat`) – Tile format.

- **raster_transform** (`RasterTransformType`) – Raster transformation. Can be RasterTransformNone or RasterTransformPalette.

- **projection** (`OrthoProjection` or `CoordinateSystem`) – Output coordinate system.

- **region** (`tuple of 4 floats`) – Region to be exported in the (x0, y0, x1, y1) format.

- **dx** (`float`) – Pixel size in the X dimension in projected units.

- **dy** (`float`) – Pixel size in the Y dimension in projected units.

- **blockw** (`int`) – Specifies block width of the DEM mosaic in pixels.

- **blockh** (`int`) – Specifies block height of the DEM mosaic in pixels.

- **width** (`int`) – Total width of the orthomosaic in pixels.

- **height** (`int`) – Total height of the orthomosaic in pixels.

- **world_transform** (`Matrix`) – 2x3 raster-to-world transformation matrix.

- **nodata** (`float`) – No-data value.

- **write_kml** (`bool`) – Enables/disables kml file generation.

- **write_world** (`bool`) – Enables/disables world file generation.

- **write_scheme** (`bool`) – Enables/disables tile scheme files generation.

- **image_description** (`string`) – Optional description to be added to image files.

- **tiff_big** (`bool`) – Enables/disables BigTIFF compression for TIFF files.

- **tiff_tiled** (`bool`) – Export tiled TIFF.

- **tiff_overviews** (`bool`) – Enables/disables image pyramid deneration for TIFF files.

- **network_links** (`bool`) – Enables/disables network links generation for KMZ format.

- **min_zoom_level** (`int`) – Minimum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).

- **max_zoom_level** (`int`) – Maximum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).

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

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

    Export markers.

        Parameters

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

- **projection** (*CoordinateSystem*) – Output coordinate system.

exportModel(*path*, *binary=True*, *precision=6*, *texture_format=ImageFormatJPEG*, *texture=True*, *uv=True*, *normals=True*, *colors=True*, *cameras=True*, *markers=True*, *udim=False*, *alpha=False*, *strip_extensions=False*, *raster_transform=RasterTransformNone*, *colors_rgb_8bit=True*[, *comment*][, *format*][, *projection*][, *shift*][, *progress*])

Export generated model for the chunk.

    Parameters

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

- **binary** (*bool*) – Enables/disables binary encoding (if supported by format).

- **precision** (*int*) – Number of digits after the decimal point (for text formats).

- **texture_format** (*ImageFormat*) – Texture format.

- **texture** (*bool*) – Enables/disables texture export.

- **uv** (*bool*) – Enables/disables export of uv coordinates.

- **normals** (*bool*) – Enables/disables export of vertex normals.

- **colors** (*bool*) – Enables/disables export of vertex colors.

- **cameras** (*bool*) – Enables/disables camera export.

- **markers** (*bool*) – Enables/disables marker export.

- **udim** (*bool*) – Enables/disables UDIM texture layout.

- **alpha** (*bool*) – Enables/disables alpha channel export.

- **strip_extensions** (*bool*) – Strips camera label extensions during export.

- **raster_transform** (*RasterTransformType*) – Raster band transformation.

- **colors_rgb_8bit** (*bool*) – Convert colors to 8 bit RGB.

- **comment** (*string*) – Optional comment (if supported by selected format).

- **format** (*ModelFormat*) – Export format.

- **projection** (*CoordinateSystem*) – Output coordinate system.

- **shift** (*3-element vector*) – Optional shift to be applied to vertex coordinates.

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

exportOrthomosaic(*path*[, *format*][, *image_format*], *raster_transform=RasterTransformNone*[, *projection*][, *region*][, *dx*][, *dy*][, *blockw*][, *blockh*][, *width*][, *height*][, *world_transform*], *write_kml=False*, *write_world=False*, *write_scheme=False*, *write_alpha=True*[, *image_description*], *tiff_compression=TiffCompressionLZW*, *tiff_big=False*, *tiff_tiled=True*, *tiff_overviews=True*, *jpeg_quality=90*, *network_links=True*[, *min_zoom_level*][, *max_zoom_level*], *white_background=True*[, *progress*])

Export orthomosaic for the chunk.

    Parameters

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

- **format** (*RasterFormat*) – Export format.

- **image_format** (*ImageFormat*) – Tile format.

- **raster_transform** (*RasterTransformType*) – Raster band transformation.
- **projection** (*OrthoProjection* or *CoordinateSystem*) – Output coordinate system.
- **region** (*tuple of 4 floats*) – Region to be exported in the (x0, y0, x1, y1) format.
- **dx** (*float*) – Pixel size in the X dimension in projected units.
- **dy** (*float*) – Pixel size in the Y dimension in projected units.
- **blockw** (*int*) – Specifies block width of the orthomosaic in pixels.
- **blockh** (*int*) – Specifies block height of the orthomosaic in pixels.
- **width** (*int*) – Total width of the orthomosaic in pixels.
- **height** (*int*) – Total height of the orthomosaic in pixels.
- **world_transform** (*Matrix*) – 2x3 raster-to-world transformation matrix.
- **write_kml** (*bool*) – Enables/disables kml file generation.
- **write_world** (*bool*) – Enables/disables world file generation.
- **write_scheme** (*bool*) – Enables/disables tile scheme files generation.
- **write_alpha** (*bool*) – Enables/disables alpha channel generation.
- **image_description** (*string*) – Optional description to be added to image files.
- **tiff_compression** (*TiffCompression*) – Tiff compression.
- **tiff_big** (*bool*) – Enables/disables BigTIFF compression for TIFF files.
- **tiff_tiled** (*bool*) – Export tiled TIFF.
- **tiff_overviews** (*bool*) – Enables/disables image pyramid deneration for TIFF files.
- **jpeg_quality** (*int*) – JPEG quality.
- **network_links** (*bool*) – Enables/disables network links generation for KMZ format.
- **min_zoom_level** (*int*) – Minimum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).
- **max_zoom_level** (*int*) – Maximum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).
- **white_background** (*bool*) – Enables/disables white background.
- **progress** (*Callable[[float], None]*) – Progress callback.

exportOrthophotos (*path*, *cameras*, *raster_transform=RasterTransformNone*[, *projection*][, *region*][, *dx*][, *dy*], *write_kml=False*, *write_world=False*, *write_alpha=True*, *tiff_compression=TiffCompressionLZW*, *tiff_big=False*, *tiff_tiled=True*, *tiff_overviews=True*, *jpeg_quality=90*, *white_background=True*[, *progress*])
Export orthophoto for the chunk.

> **Parameters**
>
> - **path** (*string*) – Path to output orthophoto.
> - **cameras** (list of *Camera*) – A list of cameras. If not specified or empty, all enabled cameras will be used.
> - **raster_transform** (*RasterTransformType*) – Raster band transformation.

- **projection** (*OrthoProjection* or *CoordinateSystem*) – Output coordinate system.

- **region** (*tuple of 4 floats*) – Region to be exported in the (x0, y0, x1, y1) format.

- **dx** (*float*) – Pixel size in the X dimension in projected units.

- **dy** (*float*) – Pixel size in the Y dimension in projected units.

- **write_kml** (*bool*) – Enables/disables kml file generation.

- **write_world** (*bool*) – Enables/disables world file generation.

- **write_alpha** (*bool*) – Enables/disables alpha channel generation.

- **tiff_compression** (*TiffCompression*) – Tiff compression.

- **tiff_big** (*bool*) – Enables/disables BigTIFF compression for TIFF files.

- **tiff_tiled** (*bool*) – Export tiled TIFF.

- **tiff_overviews** (*bool*) – Enables/disables image pyramid deneration for TIFF files.

- **jpeg_quality** (*int*) – JPEG quality.

- **white_background** (*bool*) – Enables/disables white background.

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

exportPoints (*path*[, *source*], *binary=True*, *precision=6*, *normals=True*, *colors=True*, *raster_transform=RasterTransformNone*, *colors_rgb_8bit=True*[, *comment*][, *format*][, *image_format*][, *projection*][, *shift*][, *blockw*][, *blockh*][, *classes*][, *progress*])

Export point cloud.

Parameters

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

- **source** (*DataSource*) – Selects between dense point cloud and sparse point cloud. If not specified, uses dense cloud if available.

- **binary** (*bool*) – Enables/disables binary encoding for selected format (if applicable).

- **precision** (*int*) – Number of digits after the decimal point (for text formats).

- **normals** (*bool*) – Enables/disables export of point normals.

- **colors** (*bool*) – Enables/disables export of point colors.

- **raster_transform** (*RasterTransformType*) – Raster band transformation.

- **colors_rgb_8bit** (*bool*) – Convert colors to 8 bit RGB.

- **comment** (*string*) – Optional comment (if supported by selected format).

- **format** (*PointsFormat*) – Export format.

- **image_format** (*ImageFormat*) – Image data format.

- **projection** (*CoordinateSystem*) – Output coordinate system.

- **shift** (*3-element vector*) – Optional shift to be applied to vertex coordinates.

- **blockw** (*float*) – Tile width in meters.

- **blockh** (*float*) – Tile height in meters.

- **classes** (list of *PointClass*) – List of dense point classes to be exported.

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

exportReport (*path* [, *title*] [, *description*] [, *settings*] [, *page_numbers*] [, *progress*])
    Export processing report in PDF format.

> **Parameters**
>
> - **path** (`string`) – Path to output report.
>
> - **title** (`string`) – Report title.
>
> - **description** (`string`) – Report description.
>
> - **settings** (`list of (string, string) tuples`) – A list of user defined settings to include on the Processing Parameters page.
>
> - **page_numbers** (`bool`) – Enable page numbers.
>
> - **progress** (`Callable[[float], None]`) – Progress callback.

exportShapes (*path*, *items=Shape.Polygon* [, *groups*] [, *format*] [, *projection*] [, *shift*], *polygons_as_polylines=False*, *export_labels=True*, *export_attributes=True* [, *progress*])
    Export shapes layer to file.

> **Parameters**
>
> - **path** (`string`) – Path to shape file.
>
> - **items** (*Shape.Type*) – Items to export.
>
> - **groups** (list of *ShapeGroup*) – A list of shape groups to export.
>
> - **format** (*ShapesFormat*) – Export format.
>
> - **projection** (*CoordinateSystem*) – Output coordinate system.
>
> - **shift** (`3-element vector`) – Optional shift to be applied to vertex coordinates.
>
> - **polygons_as_polylines** (`bool`) – Save polygons as polylines.
>
> - **export_labels** (`bool`) – Export labels.
>
> - **export_attributes** (`bool`) – Export attributes.
>
> - **progress** (`Callable[[float], None]`) – Progress callback.

exportTiledModel (*path*, *format=TiledModelFormatTLS*, *mesh_format=ModelFormatCOLLADA*, *raster_transform=RasterTransformNone* [, *progress*])
    Export generated tiled model for the chunk.

> **Parameters**
>
> - **path** (`string`) – Path to output model.
>
> - **format** (*TiledModelFormat*) – Export format.
>
> - **mesh_format** (*ModelFormat*) – Mesh format for zip export.
>
> - **raster_transform** (*RasterTransformType*) – Raster band transformation.
>
> - **progress** (`Callable[[float], None]`) – Progress callback.

findFrame (*key*)
    Find frame by its key.

> **Returns** Found frame.
>
> **Return type** *Chunk*

frame
　　Current frame index.

　　　　Type int

frames
　　List of frames in the chunk.

　　　　Type list of `Frame`

image_brightness
　　Image brightness as percentage.

　　　　Type float

image_contrast
　　Image contrast as percentage.

　　　　Type float

importCameras(*path*, *format=CamerasFormatXML*[, *image_list*])
　　Import camera positions.

　　　　Parameters

　　　　　　• **path** (`string`) – Path to the file.

　　　　　　• **format** ([`CamerasFormat`](#)) – File format.

　　　　　　• **image_list** (`string`) – Path to image list file (Bundler format only).

importDem(*path*[, *projection*][, *progress*])
　　Import elevation model from file.

　　　　Parameters

　　　　　　• **path** (`string`) – Path to elevation model in GeoTIFF format.

　　　　　　• **projection** ([`CoordinateSystem`](#)) – Default coordinate system if not specified in GeoTIFF file.

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

importMarkers(*path*)
　　Import markers.

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

importMasks(*path=''*, *source=MaskSourceAlpha*, *operation=MaskOperationReplacement*, *tolerance=10*[, *cameras*][, *progress*])
　　Import masks for multiple cameras.

　　　　Parameters

　　　　　　• **path** (`string`) – Mask file name template.

　　　　　　• **source** ([`MaskSource`](#)) – Mask source.

　　　　　　• **operation** ([`MaskOperation`](#)) – Mask operation.

　　　　　　• **tolerance** (`int`) – Background masking tolerance.

　　　　　　• **cameras** (list of [`Camera`](#)) – Optional list of cameras to be processed.

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

importModel(*path*[, *format*][, *projection*][, *shift*][, *progress*])
　　Import model from file.

> **Parameters**
>
> - **path** (*string*) – Path to model.
> - **format** (*ModelFormat*) – Model format.
> - **projection** (*CoordinateSystem*) – Model coordinate system.
> - **shift** (*3-element vector*) – Optional shift to be applied to vertex coordinates.
> - **progress** (*Callable[[float], None]*) – Progress callback.

importPoints (*path*[, *format*][, *projection*][, *shift*][, *progress*])
    Import point cloud from file.

> **Parameters**
>
> - **path** (*string*) – Path to point cloud.
> - **format** (*PointsFormat*) – Point cloud format.
> - **projection** (*CoordinateSystem*) – Point cloud coordinate system.
> - **shift** (*3-element vector*) – Optional shift to be applied to point coordinates.
> - **progress** (*Callable[[float], None]*) – Progress callback.

importShapes (*path=''*, *replace=False*, *boundary=Shape.NoBoundary*[, *format*])
    Import shapes layer from file.

> **Parameters**
>
> - **path** (*string*) – Path to shape file.
> - **replace** (*bool*) – Replace current shapes with new data.
> - **boundary** (*Shape.BoundaryType*) – Boundary type to be applied to imported shapes.
> - **format** (*ShapesFormat*) – Shapes format.

key
    Chunk identifier.

> **Type** int

label
    Chunk label.

> **Type** string

loadReference (*path*[, *format*], *columns='nxyzabc'*, *delimiter=' '*, *group_delimiters=False*, *skip_rows=0*[, *items*][, *crs*], *ignore_labels=False*, *create_markers=False*, *threshold=0.1*, *shutter_lag=0*[, *progress*])
    Import reference data from the specified file.

> **Parameters**
>
> - **path** (*string or stream object*) – Path to the file with reference data.
> - **format** (*ReferenceFormat*) – File format.
> - **columns** (*string*) – column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, [] - group of multiple values, | - column separator within group).
> - **delimiter** (*string*) – column delimiter in csv format.
> - **group_delimiters** (*bool*) – combine consecutive delimiters in csv format.

- **skip_rows** (*int*) – number of rows to skip in (csv format only).

- **items** (list of *ReferenceItems*) – list of items to load reference for (csv format only).

- **crs** (*CoordinateSystem*) – reference data coordinate system (csv format only).

- **ignore_labels** (*bool*) – matches reference data based on coordinates alone (csv format only).

- **create_markers** (*bool*) – create markers for missing entries (csv format only).

- **threshold** (*float*) – error threshold in meters used when ignore_labels is set (csv format only).

- **shutter_lag** (*float*) – shutter lag in seconds (APM format only).

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

Example

```
>>> loadReference('reference.csv', 'nxyz[XYZ]abc[ABC]')
>>> loadReference('reference.csv', '[n|x|y|z|XYZ|a|b|c|ABC]')
```

loadReferenceExif (*load_rotation=False*, *load_accuracy=False*)
Import camera locations from EXIF meta data.

Parameters

- **load_rotation** (*bool*) – load yaw, pitch and roll orientation angles.

- **load_accuracy** (*bool*) – load camera location accuracy.

loadReflectancePanelCalibration (*path*[, *cameras*])
Load reflectance panel calibration from CSV file.

Parameters

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

- **cameras** (list of *Camera*) – List of cameras to process.

locateReflectancePanels ([*progress*])
Locate reflectance panels based on QR-codes.

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

marker_crs
Coordinate system used for marker reference data.

Type *CoordinateSystem*

marker_groups
List of marker groups in the chunk.

Type list of *MarkerGroup*

marker_location_accuracy
Expected accuracy of marker coordinates in meters.

Type *Vector*

marker_projection_accuracy
Expected accuracy of marker projections in pixels.

Type float

markers
> List of Regular, Vertex and Fiducial markers in the chunk.
>
> > **Type** list of *Marker*

masks
> Image masks.
>
> > **Type** *Masks*

matchPhotos(*accuracy=HighAccuracy*, *preselection=ReferencePreselection*, *generic_preselection=True*, *reference_preselection=True*, *filter_mask=False*, *mask_tiepoints=False*, *keypoint_limit=40000*, *tiepoint_limit=4000*, *keep_keypoints=True*[, *pairs*][, *progress*])
> Perform image matching for the chunk frame.
>
> > **Parameters**
> >
> > - **accuracy** (*Accuracy*) – Alignment accuracy.
> >
> > - **preselection** (*Preselection*) – Image pair preselection method (obsolete).
> >
> > - **generic_preselection** (*bool*) – Enables generic image pair preselection.
> >
> > - **reference_preselection** (*bool*) – Enables reference 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 key points to look for in each photo.
> >
> > - **tiepoint_limit** (*int*) – Maximum number of tie points to generate for each photo.
> >
> > - **keep_keypoints** (*bool*) – Store keypoints in the project.
> >
> > - **pairs** (list of *Camera* tuples) – User defined list of camera pairs to match.
> >
> > - **progress** (*Callable[[float], None]*) – Progress callback.

meta
> Chunk meta data.
>
> > **Type** *MetaData*

model
> Default model for the current frame.
>
> > **Type** *Model*

models
> List of models for the current frame.
>
> > **Type** list of *Model*

modified
> Modified flag.
>
> > **Type** bool

optimizeCameras(*fit_f=True*, *fit_cx=True*, *fit_cy=True*, *fit_b1=True*, *fit_b2=True*, *fit_k1=True*, *fit_k2=True*, *fit_k3=True*, *fit_k4=False*, *fit_p1=True*, *fit_p2=True*, *fit_p3=False*, *fit_p4=False*, *adaptive_fitting=False*, *tiepoint_covariance=False*[, *progress*])
> Perform optimization of point cloud / camera parameters.
>
> > **Parameters**
> >
> > - **fit_f** (*bool*) – Enables optimization of focal length coefficient.

- **fit_cx** (*bool*) – Enables optimization of X principal point coordinates.
- **fit_cy** (*bool*) – Enables optimization of Y principal point coordinates.
- **fit_b1** (*bool*) – Enables optimization of aspect ratio.
- **fit_b2** (*bool*) – Enables optimization of skew coefficient.
- **fit_k1** (*bool*) – Enables optimization of k1 radial distortion coefficient.
- **fit_k2** (*bool*) – Enables optimization of k2 radial distortion coefficient.
- **fit_k3** (*bool*) – Enables optimization of k3 radial distortion coefficient.
- **fit_k4** (*bool*) – Enables optimization of k4 radial distortion coefficient.
- **fit_p1** (*bool*) – Enables optimization of p1 tangential distortion coefficient.
- **fit_p2** (*bool*) – Enables optimization of p2 tangential distortion coefficient.
- **fit_p3** (*bool*) – Enables optimization of p3 tangential distortion coefficient.
- **fit_p4** (*bool*) – Enables optimization of p4 tangential distortion coefficient.
- **adaptive_fitting** (*bool*) – Enables adaptive fitting of calibration oefficients.
- **tiepoint_covariance** (*bool*) – Estimate tie point covariance matrices.
- **progress** (*Callable[[float], None]*) – Progress callback.

orthomosaic
    Default orthomosaic for the current frame.

        Type *Orthomosaic*

orthomosaics
    List of orthomosaics for the current frame.

        Type list of *Orthomosaic*

point_cloud
    Generated sparse point cloud.

        Type *PointCloud*

primary_channel
    Primary channel index (-1 for default).

        Type int

raster_transform
    Raster transform.

        Type *RasterTransform*

refineMarkers ([*markers*][, *progress*])
    Refine markers based on images content.

        Parameters

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

refineModel (*quality=MediumQuality*, *iterations=10*, *smoothness=0.5*[, *progress*])
    Generate model for the chunk frame.

        Parameters

- **quality** ([*Quality*](#)) – Quality of refinement.

- **iterations** (*int*) – Number of refinement iterations.

- **smoothness** (*float*) – Smoothing strength. Should be in range [0, 1].

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

region
> Reconstruction volume selection.

> > **Type** [*Region*](#)

remove(*items*)
> Remove items from the chunk.

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

removeLighting(*color_mode=SingleColor*, *internal_blur=1.0*, *mesh_noise_suppression=1.5*, *ambient_occlusion_path=''*, *ambient_occlusion_multiplier=1.0*[, *progress*])
> Generate model for the chunk frame.

> > **Parameters**

> > - **color_mode** (*DelightingColorMode*) – Color mode of model to be delighted.

> > - **internal_blur** (*float*) – Internal blur. Should be in range [0, 4].

> > - **mesh_noise_suppression** (*float*) – Mesh normals noise suppression strength. Should be in range [0, 4].

> > - **ambient_occlusion_path** (*string*) – Path to ambient occlusion texture atlas. Can be empty.

> > - **ambient_occlusion_multiplier** (*float*) – Ambient occlusion multiplier. Should be in range [0.25, 4].

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

resetRegion()
> Reset reconstruction volume selector to default position.

saveReference(*path*[, *format*], *items=ReferenceItemsCameras*[, *columns*], *delimiter=' '*[, *progress*])
> Export reference data to the specified file.

> > **Parameters**

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

> > - **format** ([*ReferenceFormat*](#)) – Export format.

> > - **items** ([*ReferenceItems*](#)) – Items to export in CSV format.

> > - **columns** (*string*) – column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, u/v/w - estimated coordinates, U/V/W - coordinate errors, d/e/f - estimated orientation angles, D/E/F - orientation errors, [] - group of multiple values, | - column separator within group)

> > - **delimiter** (*string*) – column delimiter in csv format

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

scalebar_accuracy
    Expected scale bar accuracy in meters.

        Type float

scalebar_groups
    List of scale bar groups in the chunk.

        Type list of *ScalebarGroup*

scalebars
    List of scale bars in the chunk.

        Type list of *Scalebar*

selected
    Selects/deselects the chunk.

        Type bool

sensors
    List of sensors in the chunk.

        Type list of *Sensor*

shapes
    Shapes for the current frame.

        Type *Shapes*

smoothModel (*strength = 3*, *selected_faces = False*, *fix_borders = True*[, *progress*])
    Smooth mesh using Laplacian smoothing algorithm.

        Parameters

            • **strength** (*float*) – Smoothing strength.

            • **selected_faces** (*bool*) – Smooth only selected faces.

            • **fix_borders** (*bool*) – Fix vertices on borders.

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

sortCameras()
    Sorts cameras by their labels.

sortMarkers()
    Sorts markers by their labels.

sortScalebars()
    Sorts scalebars by their labels.

thinPointCloud(*point_limit=1000*)
    Remove excessive tracks from the point cloud.

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

thumbnails
    Image thumbnails.

        Type *Thumbnails*

tiepoint_accuracy
    Expected tie point accuracy in pixels.

        Type float

tiled_model
>   Default tiled model for the current frame.
>
>>   **Type** *TiledModel*

tiled_models
>   List of tiled models for the current frame.
>
>>   **Type** list of *TiledModel*

trackMarkers ($\big[$*start*$\big]\big[$, *end*$\big]\big[$, *progress*$\big]$)
>   Track marker projections through the frame sequence.
>
>>   **Parameters**
>>
>>   - **start** (*int*) – Starting frame index.
>>
>>   - **end** (*int*) – Ending frame index.
>>
>>   - **progress** (*Callable[[float], None]*) – Progress callback.

transform
>   4x4 matrix specifying chunk location in the world coordinate system.
>
>>   **Type** *ChunkTransform*

updateTransform()
>   Update chunk transformation based on reference data.

world_crs
>   Coordinate system used as world coordinate system.
>
>>   **Type** *CoordinateSystem*

class Metashape.**ChunkTransform**
>   Transformation between chunk and world coordinates systems.

matrix
>   Transformation matrix.
>
>>   **Type** *Matrix*

rotation
>   Rotation component.
>
>>   **Type** *Matrix*

scale
>   Scale component.
>
>>   **Type** float

translation
>   Translation component.
>
>>   **Type** *Vector*

class Metashape.**CirTransform**
>   CIR calibration matrix.

calibrate()
>   Calibrate CIR matrix based on orthomosaic histogram.

coeffs
>   Color matrix.
>
>>   **Type** *Matrix*

reset()
> Reset CIR calibration matrix.

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.loadReference("gcp.txt", Metashape.ReferenceFormatCSV)
>>> chunk.updateTransform()
```

> **addGeoid**(*path*)
> > Register geoid model.
> >
> > > **Parameters path** (*string*) – Path to geoid file.
>
> **authority**
> > Authority identifier of the coordinate system.
> >
> > > **Type** string
>
> **geoccs**
> > Base geocentric coordinate system.
> >
> > > **Type** *CoordinateSystem*
>
> **geogcs**
> > Base geographic coordinate system.
> >
> > > **Type** *CoordinateSystem*
>
> **geoid_height**
> > Fixed geoid height to be used instead of interpolated values.
> >
> > > **Type** float
>
> **init**(*crs*)
> > Initialize projection based on specified WKT definition or authority identifier.
> >
> > > **Parameters crs** (*string*) – WKT definition of coordinate system or authority identifier.
>
> **listBuiltinCRS**()
> > Returns a list of builtin coordinate systems.
>
> **localframe**(*point*)
> > Returns 4x4 transformation matrix to LSE coordinates at the given point.
> >
> > > **Parameters point** (*Vector*) – Coordinates of the origin in the geocentric coordinates.
> >
> > > **Returns** Transformation from geocentric coordinates to local coordinates.
> >
> > > **Return type** *Matrix*
>
> **name**
> > Name of the coordinate system.
> >
> > > **Type** string
>
> **proj4**
> > Coordinate system definition in PROJ.4 format.
> >
> > > **Type** string

project(*point*)

Projects point from geocentric coordinates to projected geographic coordinate system.

> **Parameters point** (*Vector*) – 3D point in geocentric coordinates.
>
> **Returns** 3D point in projected coordinates.
>
> **Return type** *Vector*

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

Transform point coordinates between coordinate systems.

> **Parameters**
>
> - **point** (2 or 3 component *Vector*) – Point coordinates.
> - **source** (*CoordinateSystem*) – Source coordinate system.
> - **target** (*CoordinateSystem*) – Target coordinate system.
>
> **Returns** Transformed point coordinates.
>
> **Return type** *Vector*

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

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

> **Parameters**
>
> - **point** (3 component *Vector*) – Point coordinates.
> - **source** (*CoordinateSystem*) – Source coordinate system.
> - **target** (*CoordinateSystem*) – Target coordinate system.
>
> **Returns** 4x4 transformation matrix.
>
> **Return type** *Matrix*

unproject(*point*)

Unprojects point from projected coordinates to geocentric coordinates.

> **Parameters point** (*Vector*) – 3D point in projected coordinate system.
>
> **Returns** 3D point in geocentric coordinates.
>
> **Return type** *Vector*

wkt

Coordinate system definition in WKT format.

> **Type** string

wkt2

Coordinate system definition in WKT format, version 2.

> **Type** string

class Metashape.**DataSource**

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

class Metashape.**DataType**

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

class Metashape.**DenseCloud**
    Dense point cloud data.

    assignClass(*target=0*[, *source*][, *progress*])
        Assign class to points.

        Parameters

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

                • **source** (*PointClass* or list of *PointClass*) – Classes of points to be replaced.

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

    assignClassToSelection(*target=0*[, *source*][, *progress*])
        Assign class to selected points.

        Parameters

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

                • **source** (*PointClass* or list of *PointClass*) – Classes of points to be replaced.

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

    bands
        List of color bands.

        Type list of string

    classifyGroundPoints(*max_angle=15.0*, *max_distance=1.0*, *cell_size=50.0*[, *source*][, *progress*])
        Classify points into ground and non ground classes.

        Parameters

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

                • **max_distance** (*float*) – Maximum distance (meters).

                • **cell_size** (*float*) – Cell size (meters).

                • **source** (*PointClass*) – Class of points to be re-classified.

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

    classifyPoints([*source*][, *target*], *confidence=0.0*[, *progress*])
        Multiclass classification of points.

        Parameters

                • **source** (*PointClass*) – Class of points to be re-classified.

                • **target** – Target point classes for classification.

                • **confidence** (*float*) – Required confidence level from 0.0 to 1.0.

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

    clear()
        Clears dense cloud data.

    compactPoints([*progress*])
        Permanently removes deleted points from dense cloud.

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

    copy()
        Create a copy of the dense cloud.

> **Returns** Copy of the dense cloud.
>
> **Return type** *[DenseCloud](#)*

cropSelectedPoints(*[point_classes][, progress]*)

> Crop selected points.
>
> > **Parameters**
> >
> > - **point_classes** (*[PointClass](#)* or list of *[PointClass](#)*) – Classes of points to be removed.
> >
> > - **progress** (*Callable[[float], None]*) – Progress callback.

crs

> Reference coordinate system.
>
> > **Type** *[CoordinateSystem](#)* or None

data_type

> Data type used to store color values.
>
> > **Type** *[DataType](#)*

key

> Dense cloud identifier.
>
> > **Type** int

label

> Dense cloud label.
>
> > **Type** string

meta

> Dense cloud meta data.
>
> > **Type** *[MetaData](#)*

modified

> Modified flag.
>
> > **Type** bool

pickPoint(*origin, target*)

> Returns ray intersection with the point cloud (point on the ray nearest to some point).
>
> > **Parameters**
> >
> > - **origin** (*[Vector](#)*) – Ray origin.
> >
> > - **target** (*[Vector](#)*) – Point on the ray.
> >
> > **Returns** Coordinates of the intersection point.
> >
> > **Return type** *[Vector](#)*

point_count

> Number of points in dense cloud.
>
> > **Type** int

removePoints(*point_classes[, progress]*)

> Remove points.
>
> > **Parameters**

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

removeSelectedPoints ([*point_classes*][, *progress*])
    Remove selected points.

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

restorePoints ([*point_classes*][, *progress*])
    Restore deleted points.

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

selectMaskedPoints (*cameras*, *softness=4*[, *progress*])
    Select dense points based on image masks.

>   Parameters
>
>   - **cameras** (list of *Camera*) – A list of cameras to use for selection.
>
>   - **softness** (*float*) – Mask edge softness.
>
>   - **progress** (*Callable[[float], None]*) – Progress callback.

selectPointsByColor (*color*, *tolerance=10*, *channels='RGB'*[, *progress*])
    Select dense points based on point colors.

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

transform
    4x4 dense cloud transformation matrix.

>   Type *Matrix*

updateStatistics ([*progress*])
    Updates dense cloud statistics.

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

class Metashape.**DepthMap**
    Depth map data.

calibration
    Depth map calibration.

>   Type *Calibration*

---

copy()
> Returns a copy of the depth map.

>> **Returns** Copy of the depth map.

>> **Return type** *[DepthMap](#)*

getCalibration(*level=0*)
> Returns calibration data.

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

>> **Returns** Calibration data.

>> **Return type** *[Calibration](#)*

image(*[level]*)
> Returns image data.

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

>> **Returns** Image data.

>> **Return type** *[Image](#)*

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

>> **Parameters**

>>> • **calibration** (*[Calibration](#)*) – Calibration data.

>>> • **level** (*int*) – Level index.

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

>> **Parameters**

>>> • **image** (*[Image](#)*) – Image object with depth map data.

>>> • **level** (*int*) – Level index.

class Metashape.**DepthMaps**
> A set of depth maps generated for a chunk frame.

> **clear**()
>> Clears depth maps data.

> **copy**()
>> Create a copy of the depth maps.

>>> **Returns** Copy of the depth maps.

>>> **Return type** *[DepthMaps](#)*

> **items**()
>> List of items.

> **key**
>> Depth maps identifier.

>>> **Type** int

> **keys**()
>> List of item keys.

> **label**
>> Depth maps label.

> **Type** string

meta
> Depth maps meta data.
>
> > **Type** *MetaData*

modified
> Modified flag.
>
> > **Type** bool

values()
> List of item values.

class Metashape.**Document**
> Metashape project.
>
> Contains list of chunks available in the project. Implements processing operations that work with multiple chunks. Supports saving/loading project files.
>
> The project currently opened in Metashape window can be accessed using Metashape.app.document attribute. Additional Document objects can be created as needed.
>
> The following example saves active chunk from the opened project in a separate project:

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

> **addChunk**()
> > Add new chunk to the document.
> >
> > > **Returns** Created chunk.
> > >
> > > **Return type** *Chunk*
>
> **alignChunks**(*chunks*, *reference*, *method='points'*, *fix_scale=False*, *accuracy=HighAccuracy*, *preselection=False*, *filter_mask=False*, *mask_tiepoints=False*, *point_limit=40000*[, *markers*][, *progress*])
> > Align specified set of chunks.
> >
> > > **Parameters**
> > >
> > > * **chunks** (*list*) – List of chunks to be aligned.
> > >
> > > * **reference** (*Chunk*) – Chunk to be used as a reference.
> > >
> > > * **method** (*string*) – Alignment method in ['points', 'markers', 'cameras'].
> > >
> > > * **fix_scale** (*bool*) – Fixes chunk scale during alignment.
> > >
> > > * **accuracy** (*Accuracy*) – Alignment accuracy.
> > >
> > > * **preselection** (*bool*) – Enables image pair preselection.
> > >
> > > * **filter_mask** (*bool*) – Filter points by mask.
> > >
> > > * **mask_tiepoints** (*bool*) – Apply mask filter to tie points.
> > >
> > > * **point_limit** (*int*) – Maximum number of points for each photo.
> > >
> > > * **markers** (list of *Marker*) – List of markers to be used for marker based alignment.
> > >
> > > * **progress** (*Callable[[float], None]*) – Progress callback.

append(*document*[, *chunks*][, *progress*])
>   Append the specified Document object to the current document.

>   **Parameters**

>   - **document** ([*Document*](#)) – Document object to be appended.

>   - **chunks** (list of [*Chunk*](#)) – List of chunks to append.

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

chunk
>   Active chunk.

>   **Type** [*Chunk*](#)

chunks
>   List of chunks in the document.

>   **Type** `Chunks`

clear()
>   Clear the contents of the Document object.

findChunk(*key*)
>   Find chunk by its key.

>   **Returns** Found chunk.

>   **Return type** [*Chunk*](#)

mergeChunks(*chunks*, *merge_dense_clouds=False*, *merge_models=False*, *merge_markers=False*[, *progress*])
>   Merge specified set of chunks.

>   **Parameters**

>   - **chunks** (`list`) – List of chunks to be merged.

>   - **merge_dense_clouds** (`bool`) – Enables/disables merging of dense clouds.

>   - **merge_models** (`bool`) – Enables/disables merging of polygonal models.

>   - **merge_markers** (`bool`) – Enables/disables merging of corresponding marker across the chunks.

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

meta
>   Document meta data.

>   **Type** [*MetaData*](#)

modified
>   Modified flag.

>   **Type** bool

open(*path*, *read_only=False*, *ignore_lock=False*)
>   Load document from the specified file.

>   **Parameters**

>   - **path** (`string`) – Path to the file.

>   - **read_only** (`bool`) – Open document in read-only mode.

>   - **ignore_lock** (`bool`) – Ignore lock state for project modifications.

path
    Path to the document file.

        Type string

read_only
    Read only status.

        Type bool

remove(*items*)
    Remove a set of items from the document.

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

save ($[path][, chunks], compression = 6, absolute\_paths = False[, version]$)
    Save document to the specified file.

        Parameters

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

class Metashape.**Elevation**
    Digital elevation model.

    altitude(*point*)
        Return elevation value at the specified point.

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

            Returns Elevation value.

            Return type float

    bottom
        Y coordinate of the bottom side.

            Type float

    clear()
        Clears elevation model data.

    copy()
        Create a copy of the elevation model.

            Returns Copy of the elevation model.

            Return type *Elevation*

    crs
        Coordinate system of elevation model.

            Type *CoordinateSystem*

    height
        Elevation model height.

            Type int

key
> Elevation model identifier.
>
> > **Type** int

label
> Elevation model label.
>
> > **Type** string

left
> X coordinate of the left side.
>
> > **Type** float

max
> Maximum elevation value.
>
> > **Type** float

meta
> Elevation model meta data.
>
> > **Type** *MetaData*

min
> Minimum elevation value.
>
> > **Type** float

modified
> Modified flag.
>
> > **Type** bool

palette
> Color palette.
>
> > **Type** dict

projection
> Projection of elevation model.
>
> > **Type** *OrthoProjection*

resolution
> DEM resolution in meters.
>
> > **Type** float

right
> X coordinate of the right side.
>
> > **Type** float

top
> Y coordinate of the top side.
>
> > **Type** float

width
> Elevation model width.
>
> > **Type** int

class Metashape.**EulerAngles**
> Euler angles in [EulerAnglesYPR, EulerAnglesOPK]

class Metashape.**FaceCount**
>   Face count in [LowFaceCount, MediumFaceCount, HighFaceCount, CustomFaceCount]

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

class Metashape.**Image**(*width*, *height*, *channels*, *datatype='U8'*)
>   n-channel image
>
>   > **Parameters**
>   >
>   > - **width** (*int*) – image width
>   >
>   > - **height** (*int*) – image height
>   >
>   > - **channels** (*string*) – color channel layout, e.g. 'RGB', 'RGBA', etc.
>   >
>   > - **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']
>
>   **channels**
>   >   Channel mapping for the image.
>   >
>   >   > **Type** string
>
>   **cn**
>   >   Number of color channels.
>   >
>   >   > **Type** int
>
>   **convert**(*channels*[, *datatype*])
>   >   Convert image to specified data type and channel layout.
>   >
>   >   > **Parameters**
>   >   >
>   >   > - **channels** (*string*) – color channels to be loaded, e.g. 'RGB', 'RGBA', etc.
>   >   >
>   >   > - **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']
>   >
>   >   > **Returns** Converted image.
>   >
>   >   > **Return type** *Image*
>
>   **copy**()
>   >   Return a copy of the image.
>   >
>   >   > **Returns** copy of the image
>   >
>   >   > **Return type** *Image*
>
>   **data_type**
>   >   Data type used to store pixel values.
>   >
>   >   > **Type** string
>
>   **fromstring**(*data*, *width*, *height*, *channels*, *datatype='U8'*)
>   >   Create image from byte array.
>   >
>   >   > **Parameters**
>   >   >
>   >   > - **data** (*string*) – raw image data
>   >   >
>   >   > - **width** (*int*) – image width
>   >   >
>   >   > - **height** (*int*) – image height
>   >   >
>   >   > - **channels** (*string*) – color channel layout, e.g. 'RGB', 'RGBA', etc.
>   >   >
>   >   > - **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']

> **Returns** Created image.
>
> **Return type** *Image*

gaussianBlur(*radius*)
  Smooth image with a gaussian filter.

> **Parameters radius** (*float*) – smoothing radius.
>
> **Returns** Smoothed image.
>
> **Return type** *Image*

height
  Image height.

> **Type** int

open(*path*, *layer=0*, *datatype='U8'*[, *channels*][, *x*][, *y*][, *w*][, *h*])
  Load image from file.

> **Parameters**
>
> - **path** (*string*) – path to the image file
> - **layer** (*int*) – image layer in case of multipage file
> - **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']
> - **channels** (*string*) – color channels to be loaded, e.g. 'RGB', 'RGBA', etc.
> - **x** (*int*) – x offset of image region.
> - **y** (*int*) – y offset of image region.
> - **w** (*int*) – width of image region.
> - **h** (*int*) – height of image region.
>
> **Returns** Loaded image.
>
> **Return type** *Image*

resize(*width*, *height*)
  Resize image to specified dimensions.

> **Parameters**
>
> - **width** (*int*) – new image width
> - **height** (*int*) – new image height
>
> **Returns** resized image
>
> **Return type** *Image*

save(*path*)
  Save image to the file.

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

tostring()
  Convert image to byte array.

> **Returns** Raw image data.
>
> **Return type** string

undistort (*calib*, *center_principal_point = True*, *square_pixels = True*)
Undistort image using provided calibration.

> **Parameters**
>
> - **calib** (`Calibration`) – lens calibration
> - **center_principal_point** (`bool`) – moves principal point to the image center
> - **square_pixels** (`bool`) – create image with square pixels
>
> **Returns** undistorted image
>
> **Return type** `Image`

uniformNoise (*amplitude*)
Add uniform noise with specified amplitude.

> **Parameters amplitude** (`float`) – noise amplitude.
>
> **Returns** Image with added noise.
>
> **Return type** `Image`

warp (*calib0*, *trans0*, *calib1*, *trans1*)
Warp image by rotating virtual viewpoint.

> **Parameters**
>
> - **calib0** (`Calibration`) – initial calibration
> - **trans0** (`Matrix`) – initial camera orientation as 4x4 matrix
> - **calib1** (`Calibration`) – final calibration
> - **trans1** (`Matrix`) – final camera orientation as 4x4 matrix
>
> **Returns** warped image
>
> **Return type** `Image`

width
Image width.

> **Type** int

class Metashape.**ImageFormat**
Image format in [ImageFormatNone, ImageFormatJPEG, ImageFormatTIFF, ImageFormatPNG, ImageFormatBMP, ImageFormatEXR, ImageFormatPNM, ImageFormatSGI, ImageFormatCR2, ImageFormatSEQ, ImageFormatBIL, ImageFormatXYZ, ImageFormatARA, ImageFormatTGA, ImageFormatDDS, ImageFormatJP2]

class Metashape.**ImageLayout**
Image layout in [UndefinedLayout, FlatLayout, MultiframeLayout, MultiplaneLayout]

class Metashape.**Interpolation**
Interpolation mode in [DisabledInterpolation, EnabledInterpolation, Extrapolated]

class Metashape.**License**
License information.

activate (*license_key*)
Activate software online using a license key.

> **Parameters key** (`string`) – Activation key.

activateOffline(*license_key*)
Create a request for offline activation.

> **Parameters key** (*string*) – Activation key.
>
> **Returns** Activation request.
>
> **Return type** string

deactivate()
Deactivate software online.

deactivateOffline()
Create a request for offline deactivation.

> **Returns** Deactivation request.
>
> **Return type** string

valid
Metashape activation status.

> **Type** bool

class Metashape.**MappingMode**
UV mapping mode in [LegacyMapping, GenericMapping, OrthophotoMapping, AdaptiveOrthophotoMapping, SphericalMapping, CameraMapping]

class Metashape.**Marker**
Marker instance

class **Projection**
Marker data().

coord
Point coordinates in pixels.
> **Type** *Vector*

pinned
Pinned flag.
> **Type** bool

valid
Valid flag.
> **Type** bool

class Marker.**Projections**
Collection of projections specified for the marker

items()
List of items.

keys()
List of item keys.

values()
List of item values.

class Marker.**Reference**
Marker reference data.

accuracy
Marker location accuracy.
> **Type** *Vector*

enabled
> Enabled flag.
> > **Type** bool

location
> Marker coordinates.
> > **Type** *Vector*

class Marker.**Type**
> Marker type in [Regular, Vertex, Fiducial]

Marker.**chunk**
> Chunk the marker belongs to.
>
> > **Type** *Chunk*

Marker.**frames**
> Marker frames.
>
> > **Type** list of *Marker*

Marker.**group**
> Marker group.
>
> > **Type** *MarkerGroup*

Marker.**key**
> Marker identifier.
>
> > **Type** int

Marker.**label**
> Marker label.
>
> > **Type** string

Marker.**meta**
> Marker meta data.
>
> > **Type** *MetaData*

Marker.**position**
> Marker position in the current frame.
>
> > **Type** *Vector*

Marker.**projections**
> List of marker projections.
>
> > **Type** MarkerProjections

Marker.**reference**
> Marker reference data.
>
> > **Type** MarkerReference

Marker.**selected**
> Selects/deselects the marker.
>
> > **Type** bool

Marker.**sensor**
> Fiducial mark sensor.
>
> > **Type** *Sensor*

Marker.**type**
>   Marker type.

>   >   **Type** *Marker.Type*

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

>   **label**
>   >   Marker group label.

>   >   >   **Type** string

>   **selected**
>   >   Current selection state.

>   >   >   **Type** bool

class Metashape.**Mask**
>   Mask instance

>   **copy**()
>   >   Returns a copy of the mask.

>   >   >   **Returns** Copy of the mask.

>   >   >   **Return type** *Mask*

>   **image**()
>   >   Returns image data.

>   >   >   **Returns** Image data.

>   >   >   **Return type** *Image*

>   **invert**()
>   >   Create inverted copy of the mask.

>   >   >   **Returns** Inverted copy of the mask.

>   >   >   **Return type** *Mask*

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

>   >   >   **Parameters**

>   >   >   - **path** (*string*) – Path to the image file to be loaded.

>   >   >   - **layer** (*int*) – Optional layer index in case of multipage files.

>   **setImage**(*image*)

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

class Metashape.**MaskOperation**
>   Mask operation in [MaskOperationReplacement, MaskOperationUnion, MaskOperationIntersection, MaskOperationDifference]

class Metashape.**MaskSource**
>   Mask source in [MaskSourceAlpha, MaskSourceFile, MaskSourceBackground, MaskSourceModel]

class Metashape.**Masks**
>   A set of masks for a chunk frame.

items()
>   List of items.

keys()
>   List of item keys.

meta
>   Thumbnails meta data.
>
>> **Type** *MetaData*

modified
>   Modified flag.
>
>> **Type** bool

values()
>   List of item values.

class Metashape.**Matrix**
>   m-by-n matrix

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

Diag(*vector*)
>   Create a diagonal matrix.
>
>> **Parameters** **vector** (*Vector* or list of floats) – The vector of diagonal entries.
>>
>> **Returns** A diagonal matrix.
>>
>> **Return type** *Matrix*

Rotation(*matrix*)
>   Create a rotation matrix.
>
>> **Parameters** **matrix** (*Matrix*) – The 3x3 rotation matrix.
>>
>> **Returns** 4x4 matrix representing rotation.
>>
>> **Return type** *Matrix*

Scale(*scale*)
>   Create a scale matrix.
>
>> **Parameters** **scale** (*Vector*) – The scale vector.
>>
>> **Returns** A matrix representing scale.
>>
>> **Return type** *Matrix*

Translation(*vector*)
>   Create a translation matrix.
>
>> **Parameters** **vector** (*Vector*) – The translation vector.
>>
>> **Returns** A matrix representing translation.
>>
>> **Return type** *Matrix*

col(*index*)
    Returns column of the matrix.

        Returns matrix column.

        Return type *Vector*

copy()
    Returns a copy of this matrix.

        Returns an instance of itself

        Return type *Matrix*

det()
    Return the determinant of a matrix.

        Returns Return a the determinant of a matrix.

        Return type float

inv()
    Returns an inverted copy of the matrix.

        Returns inverted matrix.

        Return type *Matrix*

mulp(*point*)
    Transforms a point in homogeneous coordinates.

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

        Returns transformed point.

        Return type *Vector*

mulv(*vector*)
    Transforms vector in homogeneous coordinates.

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

        Returns transformed vector.

        Return type *Vector*

rotation()
    Returns rotation component of the 4x4 matrix.

        Returns rotation component

        Return type *Matrix*

row(*index*)
    Returns row of the matrix.

        Returns matrix row.

        Return type *Vector*

scale()
    Returns scale component of the 4x4 matrix.

        Returns scale component

        Return type float

size
Matrix dimensions.

> **Type** tuple

svd()
Returns singular value decomposition of the matrix.

> **Returns** u, s, v tuple where a = u * diag(s) * v

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

t()
Return a new, transposed matrix.

> **Returns** a transposed matrix

> **Return type** *Matrix*

translation()
Returns translation component of the 4x4 matrix.

> **Returns** translation component

> **Return type** *Vector*

zero()
Set all matrix elements to zero.

class Metashape.**MetaData**(*object*)
Collection of object properties

items()
List of items.

keys()
List of item keys.

values()
List of item values.

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

class Face
Triangular face of the model

hidden
Face visibility flag.
> **Type** bool

selected
Face selection flag.
> **Type** bool

tex_index
Texture page index.
> **Type** int

tex_vertices
Texture vertex indices.
> **Type** tuple of 3 int

vertices
Vertex indices.

> > **Type** tuple of 3 int

class Model.**Faces**
Collection of model faces

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

class Model.**Statistics**
Mesh statistics

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

> **degenerate_faces**
> Number of degenerate faces.
> > **Type** int

> **duplicate_faces**
> Number of duplicate faces.
> > **Type** int

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

> **flipped_normals**
> Number of edges with flipped normals.
> > **Type** int

> **free_vertices**
> Number of free vertices.
> > **Type** int

> **multiple_edges**
> Number of edges connecting more than 2 faces.
> > **Type** int

> **open_edges**
> Number of open edges.
> > **Type** int

> **out_of_range_indices**
> Number of out of range indices.
> > **Type** int

> **similar_vertices**
> Number of similar vertices.
> > **Type** int

> **vertices**
> Total number of vertices.
> > **Type** int

> **zero_faces**
> Number of zero faces.
> > **Type** int

class Model.**TexVertex**
Texture vertex of the model

> **coord**
> > Vertex coordinates.
> > > **Type** tuple of 2 float

class `Model.`**`TexVertices`**
> Collection of model texture vertices

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

class `Model.`**`TextureType`**
> Texture type in [DiffuseMap, NormalMap, OcclusionMap]

class `Model.`**`Vertex`**
> Vertex of the model

> **color**
> > Vertex color.
> > > **Type** tuple of 3 int

> **coord**
> > Vertex coordinates.
> > > **Type** *Vector*

class `Model.`**`Vertices`**
> Collection of model vertices

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

`Model.`**`area`**()
> Return area of the model surface.

> > **Returns** Model area.

> > **Return type** float

`Model.`**`bands`**
> List of color bands.

> > **Type** list of string

`Model.`**`clear`**()
> Clears model data.

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

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

`Model.`**`copy`**()
> Create a copy of the model.

> > **Returns** Copy of the model.

> > **Return type** *Model*

`Model.`**`cropSelection`**()
> Crop selected faces and free vertices from the mesh.

`Model.`**`data_type`**
> Data type used to store color values.

> > > **Type** *[DataType](#)*

`Model.`**`faces`**
    Collection of mesh faces.

> > **Type** MeshFaces

`Model.`**`fixTopology`**`()`
    Remove polygons causing topological problems.

`Model.`**`key`**
    Model identifier.

> > **Type** int

`Model.`**`label`**
    Model label.

> > **Type** string

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

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

`Model.`**`meta`**
    Model meta data.

> > **Type** *[MetaData](#)*

`Model.`**`modified`**
    Modified flag.

> > **Type** bool

`Model.`**`pickPoint`**(*origin*, *target*)
    Return ray intersection with mesh.

> > **Parameters**

> > > - **`origin`** (*[Vector](#)*) – Ray origin.

> > > - **`target`** (*[Vector](#)*) – Point on the ray.

> > **Returns** Coordinates of the intersection point.

> > **Return type** *[Vector](#)*

`Model.`**`removeComponents`**(*size*)
    Remove small connected components.

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

`Model.`**`removeSelection`**`()`
    Remove selected faces and free vertices from the mesh.

`Model.`**`renderDepth`**(*transform*, *calibration*)
    Render model depth image for specified viewpoint.

> > **Parameters**

> > > - **`transform`** (*[Matrix](#)*) – Camera location.

> > > - **`calibration`** (*[Calibration](#)*) – Camera calibration.

> > **Returns** Rendered image.

> > **Return type** *[Image](#)*

`Model.`**`renderImage`**(*transform*, *calibration*)
  Render model image for specified viewpoint.

> **Parameters**
>
> > • **transform** (*[Matrix](#)*) – Camera location.
> >
> > • **calibration** (*[Calibration](#)*) – Camera calibration.
>
> **Returns** Rendered image.
>
> **Return type** *[Image](#)*

`Model.`**`renderMask`**(*transform*, *calibration*)
  Render model mask image for specified viewpoint.

> **Parameters**
>
> > • **transform** (*[Matrix](#)*) – Camera location.
> >
> > • **calibration** (*[Calibration](#)*) – Camera calibration.
>
> **Returns** Rendered image.
>
> **Return type** *[Image](#)*

`Model.`**`renderNormalMap`**(*transform*, *calibration*)
  Render image with model normals for specified viewpoint.

> **Parameters**
>
> > • **transform** (*[Matrix](#)*) – Camera location.
> >
> > • **calibration** (*[Calibration](#)*) – Camera calibration.
>
> **Returns** Rendered image.
>
> **Return type** *[Image](#)*

`Model.`**`saveTexture`**(*path*)
  Save texture to the specified file.

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

`Model.`**`setTexture`**(*image*, *page=0*, *type=Model.DiffuseMap*)
  Initialize texture from image data.

> **Parameters**
>
> > • **image** (*[Image](#)*) – Texture image.
> >
> > • **page** (*int*) – Texture index for multitextured models.
> >
> > • **type** (*[Model.TextureType](#)*) – Texture type.

`Model.`**`statistics`**([*progress*])
  Return mesh statistics.

> **Parameters** **progress** (*Callable[[float], None]*) – Progress callback.
>
> **Returns** Mesh statistics.
>
> **Return type** *[Model.Statistics](#)*

`Model.`**`tex_vertices`**
  Collection of mesh texture vertices.

> **Type** MeshTexVertices

Model.**texture**(*page=0*, *type=Model.DiffuseMap*)
    Return texture image.

>   **Parameters**
>
>   • **page** (*int*) – Texture index for multitextured models.
>
>   • **type** (*Model.TextureType*) – Texture type.
>
>   **Returns** Texture image.
>
>   **Return type** *Image*

Model.**vertices**
    Collection of mesh vertices.

>   **Type** MeshVertices

Model.**volume**()
    Return volume of the closed model surface.

>   **Returns** Model volume.
>
>   **Return type** float

class Metashape.**ModelFormat**
    Model format in [ModelFormatNone, ModelFormatOBJ, ModelFormat3DS, ModelFormatVRML, ModelFormatPLY, ModelFormatCOLLADA, ModelFormatU3D, ModelFormatPDF, ModelFormatDXF, ModelFormatFBX, ModelFormatKMZ, ModelFormatCTM, ModelFormatSTL, ModelFormatDXF_3DF, ModelFormatTLS, ModelFormatABC, ModelFormatOSGB, ModelFormatGLTF, ModelFormatX3D]

class Metashape.**ModelViewMode**
    Model view mode in [ShadedModelView, SolidModelView, WireframeModelView, TexturedModelView]

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

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

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

abortBatch(*batch_id*)
    Abort batch.

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

abortNode(*node_id*)
    Abort node.

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

batchList(*revision=0*)
    Get list of batches.

>   **Parameters** **revision** (*int*) – First revision to get.
>
>   **Returns** List of batches.
>
>   **Return type** dict

batchStatus(*batch_id*, *revision=0*)
    Get batch status.

> **Parameters**
>
> - **batch_id** (*int*) – Batch id.
>
> - **revision** (*int*) – First revision to get.
>
> **Returns** Batch status.
>
> **Return type** dict

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

Connect to the server.

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

createBatch (*path*, *tasks*)

Create new batch.

> **Parameters**
>
> - **path** (*string*) – Project path relative to root folder.
>
> - **tasks** (list of *NetworkTask*) – Project path relative to root folder.
>
> **Returns** Batch id.
>
> **Return type** int

disconnect ()

Disconnect from the server.

dumpBatches ($\big[$*batch_ids*$\big]$)

Dump current state of batches.

> **Parameters batch_ids** (*list of int*) – First revision to get.
>
> **Returns** Batches data.
>
> **Return type** string

findBatch (*path*)

Get batch id based on project path.

> **Parameters path** (*string*) – Project path relative to root folder.
>
> **Returns** Batch id.
>
> **Return type** int

loadBatches (*data*)

Load batches from dump.

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

nodeList (*revision=0*)

Get list of nodes.

> **Parameters revision** (*int*) – First revision to get.
>
> **Returns** List of nodes.
>
> **Return type** dict

nodeStatus (*node_id*, *revision=0*)

Get node status.

> Parameters
>
> - **node_id** (*int*) – Node id.
>
> - **revision** (*int*) – First revision to get.
>
> Returns Node status.
>
> Return type dict

pauseBatch(*batch_id*)

Pause batch.

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

pauseNode(*node_id*)

Pause node.

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

quitNode(*node_id*)

Quit node.

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

resumeBatch(*batch_id*)

Resume batch.

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

resumeNode(*node_id*)

Resume node.

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

serverInfo()

Get server information.

> Returns Server information.
>
> Return type dict

setBatchNodeLimit(*batch_id*, *node_limit*)

Set node limit of the batch.

> Parameters
>
> - **batch_id** (*int*) – Batch id.
>
> - **node_limit** (*int*) – Node limit of the batch (0 - unlimited).

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).

setNodeCPUEnable(*node_id*, *cpu_enable*)

Set node CPU enable flag.

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

> - **cpu_enable** (*bool*) – CPU enable flag.

setNodeCapability(*node_id*, *capability*)
> Set node capability.

> > **Parameters**

> > - **node_id** (*int*) – Node id.

> > - **capability** (*int*) – Node capability (1 - CPU, 2 - GPU, 3 - Any).

setNodeGPUMask(*node_id*, *gpu_mask*)
> Set node GPU mask.

> > **Parameters**

> > - **node_id** (*int*) – Node id.

> > - **gpu_mask** (*int*) – GPU device mask.

setNodePriority(*node_id*, *priority*)
> Set node priority.

> > **Parameters**

> > - **node_id** (*int*) – Node id.

> > - **priority** (*int*) – Node priority (2 - Highest, 1 - High, 0 - Normal, -1 - Low, -2 - Lowest).

class Metashape.**NetworkTask**
> NetworkTask class contains information about network task and its parameters.

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

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

> **chunks**
> > List of chunks.

> > **Type** list

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

> **frames**
> > List of frames.

> > **Type** list

> **name**
> > Task name.

> > **Type** string

> **params**
> > Task parameters.

> > **Type** dict

supports_gpu
> GPU support flag.
>
> > **Type** bool

class Metashape.**OrthoProjection**
> Orthographic projection.
>
> **class Type**
> > Projection type in [Planar, Cylindrical]
>
> OrthoProjection.**crs**
> > Base coordinate system.
> >
> > > **Type** *CoordinateSystem*
>
> OrthoProjection.**matrix**
> > Ortho transformation matrix.
> >
> > > **Type** *Matrix*
>
> OrthoProjection.**radius**
> > Cylindrical projection radius.
> >
> > > **Type** float
>
> OrthoProjection.**transform**(*point*, *source*, *target*)
> > Transform point coordinates between coordinate systems.
> >
> > > **Parameters**
> > >
> > > - **point** (2 or 3 component *Vector*) – Point coordinates.
> > > - **source** (*CoordinateSystem*) – Source coordinate system.
> > > - **target** (*CoordinateSystem*) – Target coordinate system.
> > >
> > > **Returns** Transformed point coordinates.
> > >
> > > **Return type** *Vector*
>
> OrthoProjection.**type**
> > Projection type.
> >
> > > **Type** *OrthoProjection.Type*

class Metashape.**Orthomosaic**
> Orthomosaic data.
>
> The following sample assigns to the first shape in the chunk the image from the first camera for the orthomosaic patch and updates the mosaic:

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

> **class Patch**
> > Orthomosaic patch.

copy()
> Returns a copy of the patch.
> > **Returns** Copy of the patch.
> > **Return type** *Orthomosaic.Patch*

excluded
> Excluded flag.
> > **Type** bool

image_keys
> Image keys.
> > **Type** list of int

class Orthomosaic.**Patches**
> A set of orthomosaic patches.

> **items**()
> > List of items.

> **keys**()
> > List of item keys.

> **values**()
> > List of item values.

Orthomosaic.**bands**
> List of color bands.

> > **Type** list of string

Orthomosaic.**bottom**
> Y coordinate of the bottom side.

> > **Type** float

Orthomosaic.**clear**()
> Clears orthomosaic data.

Orthomosaic.**copy**()
> Create a copy of the orthomosaic.

> > **Returns** Copy of the orthomosaic.

> > **Return type** *Orthomosaic*

Orthomosaic.**crs**
> Coordinate system of orthomosaic.

> > **Type** *CoordinateSystem*

Orthomosaic.**data_type**
> Data type used to store color values.

> > **Type** *DataType*

Orthomosaic.**height**
> Orthomosaic height.

> > **Type** int

Orthomosaic.**key**
> Orthomosaic identifier.

> > **Type** int

Orthomosaic.**label**
    Orthomosaic label.

        Type string

Orthomosaic.**left**
    X coordinate of the left side.

        Type float

Orthomosaic.**meta**
    Orthomosaic meta data.

        Type *MetaData*

Orthomosaic.**modified**
    Modified flag.

        Type bool

Orthomosaic.**patches**
    Orthomosaic patches.

        Type *Orthomosaic.Patches*

Orthomosaic.**projection**
    Orthomosaic projection.

        Type *OrthoProjection*

Orthomosaic.**removeOrthophotos**()
    Remove orthorectified images from orthomosaic.

Orthomosaic.**reset**($\big[$*progress*$\big]$)
    Reset all edits to orthomosaic.

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

Orthomosaic.**resolution**
    Orthomosaic resolution in meters.

        Type float

Orthomosaic.**right**
    X coordinate of the right side.

        Type float

Orthomosaic.**top**
    Y coordinate of the top side.

        Type float

Orthomosaic.**update**($\big[$*progress*$\big]$)
    Apply edits to orthomosaic.

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

Orthomosaic.**width**
    Orthomosaic width.

        Type int

class Metashape.**Photo**
    Photo instance

alpha()
>   Returns alpha channel data.
>
>>   **Returns** Alpha channel data.
>>
>>   **Return type** *Image*

copy()
>   Returns a copy of the photo.
>
>>   **Returns** Copy of the photo.
>>
>>   **Return type** *Photo*

image([*channels*][, *datatype*])
>   Returns image data.
>
>>   **Parameters**
>>
>>> • **datatype** (*string*) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']
>>>
>>> • **channels** (*string*) – color channels to be loaded, e.g. 'RGB', 'RGBA', etc.
>>
>>   **Returns** Image data.
>>
>>   **Return type** *Image*

imageMeta()
>   Returns image meta data.
>
>>   **Returns** Image meta data.
>>
>>   **Return type** *MetaData*

layer
>   Layer index in the image file.
>
>>   **Type** int

meta
>   Frame meta data.
>
>>   **Type** *MetaData*

open(*path*[, *layer*])
>   Loads specified image file.
>
>>   **Parameters**
>>
>>> • **path** (*string*) – Path to the image file to be loaded.
>>>
>>> • **layer** (*int*) – Optional layer index in case of multipage files.

path
>   Path to the image file.
>
>>   **Type** string

thumbnail(*width=192*, *height=192*)
>   Creates new thumbnail with specified dimensions.
>
>>   **Returns** Thumbnail data.
>>
>>   **Return type** *Thumbnail*

class Metashape.**PointClass**
>   Point class in [Created, Unclassified, Ground, LowVegetation, MediumVegetation, HighVegetation, Building,

LowPoint, ModelKeyPoint, Water, Rail, RoadSurface, OverlapPoints, WireGuard, WireConductor, TransmissionTower, WireConnector, BridgeDeck, HighNoise, Car, Manmade]

class Metashape.**PointCloud**
    Sparse point cloud instance

> **class Cameras**
>   Collection of *PointCloud.Projections* objects indexed by corresponding cameras
>
> **class** PointCloud.**Filter**
>   Sparse point cloud filter
>
>   The following example selects all points of the sparse cloud from the active chunk that have reprojection error higher than defined threshold:
>
> ```
> >>> chunk = Metashape.app.document.chunk # active chunk
> >>> threshold = 0.5
> >>> f = Metashape.PointCloud.Filter()
> >>> f.init(chunk, criterion = Metashape.PointCloud.Filter.ReprojectionError)
> >>> f.selectPoints(threshold)
> ```
>
> > **class Criterion**
> >   Point filtering criterion in [ReprojectionError, ReconstructionUncertainty, ImageCount, ProjectionAccuracy]
> >
> > PointCloud.Filter.**init**(*points*, *criterion*, *progress*)
> >   Initialize point cloud filter based on specified criterion.
> >   **Parameters**
> >   • **points** (*PointCloud* or *Chunk*) – Point cloud to filter.
> >   • **criterion** (*PointCloud.Filter.Criterion*) – Point filter criterion.
> >   • **progress** (*Callable[[float], None]*) – Progress callback.
> >
> > PointCloud.Filter.**max_value**
> >   Maximum value.
> >   **Type** int or double
> >
> > PointCloud.Filter.**min_value**
> >   Minimum value.
> >   **Type** int or double
> >
> > PointCloud.Filter.**removePoints**(*threshold*)
> >   Remove points based on specified threshold.
> >   **Parameters threshold**(*float*) – Criterion threshold.
> >
> > PointCloud.Filter.**resetSelection**()
> >   Reset previously made selection.
> >
> > PointCloud.Filter.**selectPoints**(*threshold*)
> >   Select points based on specified threshold.
> >   **Parameters threshold**(*float*) – Criterion threshold.
> >
> > PointCloud.Filter.**values**
> >   List of values.
> >   **Type** list of int or list of double
>
> **class** PointCloud.**Point**
>   3D point in the point cloud
>
> > **coord**
> >   Point coordinates.
> >   **Type** *Vector*

> **cov**
>> Point coordinates covariance matrix.
>>> **Type** *Matrix*

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

> **track_id**
>> Track index.
>>> **Type** int

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

class `PointCloud.`**`Points`**
> Collection of 3D points in the point cloud

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

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

class `PointCloud.`**`Projection`**
> Projection of the 3D point on the photo

> **coord**
>> Projection coordinates.
>>> **Type** tuple of 2 float

> **size**
>> Point size.
>>> **Type** float

> **track_id**
>> Track index.
>>> **Type** int

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

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

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

class `PointCloud.`**`Track`**
> Track in the point cloud

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

class `PointCloud.`**`Tracks`**
    Collection of tracks in the point cloud

    `copy``()`
        Returns a copy of tracks buffer.
            Returns Copy of tracks buffer.
            Return type *PointCloud.Tracks*

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

`PointCloud.`**`bands`**
    List of color bands.

        Type list of string

`PointCloud.`**`copy`**`(keypoints=True)`
    Returns a copy of the point cloud.

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

        Returns Copy of the point cloud.

        Return type *PointCloud*

`PointCloud.`**`cropSelectedPoints`**`()`
    Crop selected points.

`PointCloud.`**`cropSelectedTracks`**`()`
    Crop selected tie points.

`PointCloud.`**`data_type`**
    Data type used to store color values.

        Type *DataType*

`PointCloud.`**`export`**`(path, format='obj'[, projection])`
    Export point cloud.

        Parameters

            • **`path`** (*string*) – Path to output file.

            • **`format`** (*string*) – Export format in ['obj', 'ply'].

            • **`projection`** (*Matrix* or *CoordinateSystem*) – Sets output projection.

`PointCloud.`**`meta`**
    Point cloud meta data.

        Type *MetaData*

`PointCloud.`**`modified`**
    Modified flag.

        Type bool

`PointCloud.`**`pickPoint`**`(origin, target)`
    Returns ray intersection with the point cloud (point on the ray nearest to some point).

        Parameters

            • **`origin`** (*Vector*) – Ray origin.

            • **`target`** (*Vector*) – Point on the ray.

> **Returns** Coordinates of the intersection point.
>
> **Return type** *Vector*

PointCloud.**points**
    List of points.

> **Type** *PointCloud.Points*

PointCloud.**projections**
    Point projections for each photo.

> **Type** *PointCloud.Projections*

PointCloud.**removeKeypoints**()
    Remove keypoints from point cloud.

PointCloud.**removeSelectedPoints**()
    Remove selected points.

PointCloud.**removeSelectedTracks**()
    Remove selected tie points.

PointCloud.**tracks**
    List of tracks.

> **Type** *PointCloud.Tracks*

class Metashape.**PointsFormat**
    Point cloud format in [PointsFormatNone, PointsFormatOBJ, PointsFormatPLY, PointsFormatXYZ, Points-FormatLAS, PointsFormatExpe, PointsFormatU3D, PointsFormatPDF, PointsFormatE57, PointsFormatOC3, PointsFormatPotree, PointsFormatLAZ, PointsFormatCL3, PointsFormatPTS, PointsFormatDXF, PointsFormatCesium]

class Metashape.**Preselection**
    Image pair preselection in [NoPreselection, GenericPreselection, ReferencePreselection]

class Metashape.**Quality**
    Dense point cloud quality in [UltraQuality, HighQuality, MediumQuality, LowQuality, LowestQuality]

class Metashape.**RasterFormat**
    Raster format in [RasterFormatNone, RasterFormatTiles, RasterFormatKMZ, RasterFormatXYZ, RasterFormatMBTiles, RasterFormatWW, RasterFormatTMS]

class Metashape.**RasterTransform**
    Raster transform definition.

calibrateRange()
    Auto detect range based on orthomosaic histogram.

enabled
    Enable flag.

> **Type** bool

false_color
    False color channels.

> **Type** list

formula
    Raster calculator expression.

> **Type** string

interpolation
>   Interpolation enable flag.

>   >   **Type** bool

palette
>   Color palette.

>   >   **Type** dict

range
>   Palette mapping range.

>   >   **Type** tuple

reset()
>   Reset raster transform.

class Metashape.**RasterTransformType**
>   Raster transformation type in [RasterTransformNone, RasterTransformValue, RasterTransformPalette]

class Metashape.**ReferenceFormat**
>   Reference format in [ReferenceFormatNone, ReferenceFormatXML, ReferenceFormatTEL, ReferenceFor-matCSV, ReferenceFormatMavinci, ReferenceFormatBramor, ReferenceFormatAPM]

class Metashape.**ReferenceItems**
>   Reference items in [ReferenceItemsCameras, ReferenceItemsMarkers, ReferenceItemsScalebars]

class Metashape.**Region**
>   Region parameters

>   **center**
>   >   Region center coordinates.

>   >   >   **Type** *Vector*

>   **rot**
>   >   Region rotation matrix.

>   >   >   **Type** *Matrix*

>   **size**
>   >   Region size.

>   >   >   **Type** *Vector*

class Metashape.**RotationOrder**
>   Rotation order in [RotationOrderXYZ, RotationOrderXZY, RotationOrderYXZ, RotationOrderYZX, Rota-tionOrderZXY, 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

Scalebar.chunk
> Chunk the scalebar belongs to.
>> **Type** *Chunk*

Scalebar.frames
> Scale bar frames.
>> **Type** list of *Scalebar*

Scalebar.group
> Scale bar group.
>> **Type** *ScalebarGroup*

Scalebar.key
> Scale bar identifier.
>> **Type** int

Scalebar.label
> Scale bar label.
>> **Type** string

Scalebar.meta
> Scale bar meta data.
>> **Type** *MetaData*

Scalebar.point0
> Start of the scale bar.
>> **Type** *Marker*

Scalebar.point1
> End of the scale bar.
>> **Type** *Marker*

Scalebar.reference
> Scale bar reference data.
>> **Type** ScalebarReference

Scalebar.selected
> Selects/deselects the scale bar.
>> **Type** bool

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

> **label**
>> Scale bar group label.
>>> **Type** string

> **selected**
>> Current selection state.

> **Type** bool

class Metashape.**Sensor**
> Sensor instance

> **class Reference**
> > Sensor reference data.

> > **accuracy**
> > > Sensor location accuracy.
> > > > **Type** *Vector*

> > **enabled**
> > > Enabled flag.
> > > > **Type** bool

> > **location**
> > > Sensor coordinates.
> > > > **Type** *Vector*

> > **location_accuracy**
> > > Sensor location accuracy.
> > > > **Type** *Vector*

> > **rotation**
> > > Sensor rotation angles.
> > > > **Type** *Vector*

> > **rotation_accuracy**
> > > Sensor rotation accuracy.
> > > > **Type** *Vector*

> **class** Sensor.**Type**
> > Sensor type in [Frame, Fisheye, Spherical]

> Sensor.**antenna**
> > GPS antenna correction.

> > > **Type** *Antenna*

> Sensor.**bands**
> > List of color bands.

> > > **Type** list of string

> Sensor.**black_level**
> > Black level for each band.

> > > **Type** list of float

> Sensor.**calibrateFiducials**(*resolution=0.014*)
> > Fit fiducial coordinates to image measurements.

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

> Sensor.**calibration**
> > Adjusted calibration of the photo.

> > > **Type** *Calibration*

> Sensor.**chunk**
> > Chunk the sensor belongs to.

> > > **Type** *Chunk*

Sensor.**data_type**
    Data type used to store color values.

> **Type** *DataType*

Sensor.**fiducials**
    Fiducial marks.

> **Type** list of *Marker*

Sensor.**film_camera**
    Film camera flag.

> **Type** bool

Sensor.**fixed**
    Fix calibration flag.

> **Type** bool

Sensor.**fixed_calibration**
    Fix calibration flag.

> **Type** bool

Sensor.**fixed_location**
    Fix location flag.

> **Type** bool

Sensor.**fixed_params**
    List of fixed calibration parameters.

> **Type** list of string

Sensor.**fixed_rotation**
    Fix rotation flag.

> **Type** bool

Sensor.**focal_length**
    Focal length in mm.

> **Type** float

Sensor.**height**
    Image height.

> **Type** int

Sensor.**key**
    Sensor identifier.

> **Type** int

Sensor.**label**
    Sensor label.

> **Type** string

Sensor.**layer_index**
    Sensor layer index.

> **Type** int

Sensor.**location**
    Sensor plane location.

> **Type** *Vector*

Sensor.**location_covariance**
  Sensor plane location covariance.

> **Type** *Matrix*

Sensor.**master**
  Master sensor.

> **Type** *Sensor*

Sensor.**normalize_sensitivity**
  Enable sensitivity normalization.

> **Type** bool

Sensor.**normalize_to_float**
  Convert pixel values to floating point after normalization.

> **Type** bool

Sensor.**photo_params**
  List of photo-invariant calibration parameters.

> **Type** list of string

Sensor.**pixel_height**
  Pixel height in mm.

> **Type** float

Sensor.**pixel_size**
  Pixel size in mm.

> **Type** *Vector*

Sensor.**pixel_width**
  Pixel width in mm.

> **Type** float

Sensor.**planes**
  Sensor planes.

> **Type** list of *Sensor*

Sensor.**reference**
  Sensor reference data.

> **Type** SensorReference

Sensor.**rolling_shutter**
  Enable rolling shutter compensation.

> **Type** bool

Sensor.**rotation**
  Sensor plane rotation.

> **Type** *Matrix*

Sensor.**rotation_covariance**
  Sensor plane rotation covariance.

> **Type** *Matrix*

Sensor.**sensitivity**
Sensitivity for each band.

>   **Type** list of float

Sensor.**type**
Sensor projection model.

>   **Type** *Sensor.Type*

Sensor.**user_calib**
Custom calibration used as initial calibration during photo alignment.

>   **Type** *Calibration*

Sensor.**vignetting**
Vignetting for each band.

>   **Type** list of *Vignetting*

Sensor.**width**
Image width.

>   **Type** int

class Metashape.**Shape**
Shape data.

>   **class BoundaryType**
>   Shape boundary type in [NoBoundary, OuterBoundary, InnerBoundary]
>
>   **class** Shape.**Type**
>   Shape type in [Point, Polyline, Polygon]
>
>   **class** Shape.**Vertices**
>   Collection of shape vertices
>
>   Shape.**area**()
>   Return area of the shape on DEM.
>
>   >   **Returns** Shape area.
>   >
>   >   **Return type** float
>
>   Shape.**attributes**
>   Shape attributes.
>
>   >   **Type** *MetaData*
>
>   Shape.**boundary_type**
>   Shape boundary type.
>
>   >   **Type** *Shape.BoundaryType*
>
>   Shape.**group**
>   Shape group.
>
>   >   **Type** *ShapeGroup*
>
>   Shape.**has_z**
>   Z enable flag.
>
>   >   **Type** bool
>
>   Shape.**key**
>   Shape identifier.

> > > **Type** int

Shape.**label**
> Shape label.

> > **Type** string

Shape.**perimeter2D**()
> Return perimeter of the shape on DEM.

> > **Returns** Shape perimeter.

> > **Return type** float

Shape.**perimeter3D**()
> Return perimeter of the shape.

> > **Returns** Shape perimeter.

> > **Return type** float

Shape.**selected**
> Selects/deselects the shape.

> > **Type** bool

Shape.**type**
> Shape type.

> > **Type** *Shape.Type*

Shape.**vertex_ids**
> List of shape vertex ids.

> > **Type** ShapeVertices

Shape.**vertices**
> List of shape vertices.

> > **Type** ShapeVertices

Shape.**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 Shape-
> Group instance to the Shape.group attribute of participating shapes.

> **color**
> > Shape group color.

> > > **Type** tuple of 3 int

> **enabled**
> > Enable flag.

> > > **Type** bool

> **key**
> > Shape group identifier.

Type int

label
Shape group label.

Type string

selected
Current selection state.

Type bool

show_labels
Shape labels visibility flag.

Type bool

class `Metashape.`**`Shapes`**
A set of shapes for a chunk frame.

addGroup()
Add new shape group to the set of shapes.

Returns Created shape group.

Return type *ShapeGroup*

addShape()
Add new shape to the set of shapes.

Returns Created shape.

Return type *Shape*

crs
Shapes coordinate system.

Type *CoordinateSystem*

groups
List of shape groups.

Type list of *ShapeGroup*

items()
List of items.

meta
Shapes meta data.

Type *MetaData*

modified
Modified flag.

Type bool

projection
Shapes projection.

Type *OrthoProjection*

remove(*items*)
Remove items from the shape layer.

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

shapes
>   List of shapes.

>> **Type** list of [`Shape`](#)

updateAltitudes(*items*[, *progress*])
>   Update altitudes for items.

>> **Parameters**

>>>   • **items** (list of [`Shape`](#) or [`ShapeGroup`](#)) – A list of items to be updated.

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

class Metashape.**ShapesFormat**
>   Shapes format in [ShapesFormatNone, ShapesFormatSHP, ShapesFormatKML, ShapesFormatDXF]

class Metashape.**Shutter**
>   Shutter object contains estimated parameters of the rolling shutter correction model.

>   **rotation**
>>   Rotation matrix of the rolling shutter model.

>>>   **Type** [`Matrix`](#)

>   **translation**
>>   Translation vector of the rolling shutter model.

>>>   **Type** [`Vector`](#)

class Metashape.**SurfaceType**
>   Surface type in [Arbitrary, HeightField]

class Metashape.**Target**
>   Target parameters

>   **code**
>>   Target code.

>>>   **Type** int

>   **coord**
>>   Target location.

>>>   **Type** [`Vector`](#)

>   **radius**
>>   Target radius.

>>>   **Type** float

class Metashape.**TargetType**
>   Target type in [CircularTarget12bit, CircularTarget14bit, CircularTarget16bit, CircularTarget20bit, CircularTarget, CrossTarget]

class Metashape.**Tasks**
>   Task classes.

>   **class AddFrames**
>>   Task class containing processing parameters.

>>   **apply**(*object*[, *workitem*][, *progress*])
>>>   Apply task to specified object.
>>>   **Parameters**
>>>   • **object** ([`Chunk`](#) or [`Document`](#)) – Chunk or Document object to be processed.

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

chunk
> Chunk to copy frames from.
> > **Type** int

copy_dense_cloud
> Copy dense cloud.
> > **Type** bool

copy_depth_maps
> Copy depth maps.
> > **Type** bool

copy_elevation
> Copy DEM.
> > **Type** bool

copy_model
> Copy model.
> > **Type** bool

copy_orthomosaic
> Copy orthomosaic.
> > **Type** bool

copy_tiled_model
> Copy tiled model.
> > **Type** bool

decode (*dict*)
> Initializaes task parameters with a dictionary.

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

encode ()
> Create a dictionary with task parameters.

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

frames
> List of frame keys to copy.
> > **Type** list of int

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

supports_gpu
> GPU support flag.
> > **Type** bool

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

workitem_count
> Work item count.
> > **Type** int

class `Tasks.`**`AddPhotos`**

> Task class containing processing parameters.
>
> **`apply`**(*object*[, *workitem*][, *progress*])
>
> > Apply task to specified object.
> >
> > > **Parameters**
> > >
> > > - **`object`** (*Chunk* or *Document*) – Chunk or Document object to be processed.
> > > - **`workitem`** (*int*) – Workitem index.
> > > - **`progress`** (*Callable[[float], None]*) – Progress callback.
>
> **`decode`**(*dict*)
>
> > Initializaes task parameters with a dictionary.
>
> **`decodeJSON`**(*json*)
>
> > Initializaes task parameters from a JSON string.
>
> **`encode`**()
>
> > Create a dictionary with task parameters.
>
> **`encodeJSON`**()
>
> > Create a JSON string with task parameters.
>
> **`filegroups`**
>
> > List of file groups.
> >
> > > **Type** list of int
>
> **`filenames`**
>
> > List of files to add.
> >
> > > **Type** list of string
>
> **`group`**
>
> > Camera group key.
> >
> > > **Type** int
>
> **`layout`**
>
> > Image layout.
> >
> > > **Type** *ImageLayout*
>
> **`load_reference`**
>
> > Load reference coordinates.
> >
> > > **Type** bool
>
> **`load_xmp_accuracy`**
>
> > Load accuracy from XMP meta data.
> >
> > > **Type** bool
>
> **`load_xmp_antenna`**
>
> > Load GPS/INS offset from XMP meta data.
> >
> > > **Type** bool
>
> **`load_xmp_calibration`**
>
> > Load calibration from XMP meta data.
> >
> > > **Type** bool
>
> **`load_xmp_orientation`**
>
> > Load orientation from XMP meta data.
> >
> > > **Type** bool
>
> **`name`**
>
> > Task name.
> >
> > > **Type** string

strip_extensions
> Strip file extensions from camera labels.
> > **Type** bool

supports_gpu
> GPU support flag.
> > **Type** bool

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

workitem_count
> Work item count.
> > **Type** int

class `Tasks.`**`AlignCameras`**
> Task class containing processing parameters.

adaptive_fitting
> Enable adaptive fitting of distortion coefficients.
> > **Type** bool

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

cameras
> List of cameras to align.
> > **Type** list of int

decode (*dict*)
> Initializaes task parameters with a dictionary.

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

encode ()
> Create a dictionary with task parameters.

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

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

network_distribute
> Enable distributed processing.
> > **Type** bool

reset_alignment
> Reset current alignment.
> > **Type** bool

supports_gpu
> GPU support flag.
> > **Type** bool

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

workitem_count
>   Work item count.
>   >   **Type** int

class `Tasks.`**`AlignChunks`**
>   Task class containing processing parameters.

align_method
>   Alignment method.
>   >   **Type** int

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

chunks
>   List of chunks to be aligned.
>   >   **Type** list of int

decode(*dict*)
>   Initializaes task parameters with a dictionary.

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

encode()
>   Create a dictionary with task parameters.

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

fit_scale
>   Fit chunk scale during alignment.
>   >   **Type** bool

markers
>   List of markers to be used for marker based alignment.
>   >   **Type** list of int

match_downscale
>   Alignment accuracy.
>   >   **Type** int

match_filter_mask
>   Filter points by mask.
>   >   **Type** bool

match_mask_tiepoints
>   Apply mask filter to tie points.
>   >   **Type** bool

match_point_limit
>   Maximum number of points for each photo.
>   >   **Type** int

match_select_pairs
> Enables image pair preselection.
>> **Type** bool

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

reference
> Chunk to be used as a reference.
>> **Type** int

supports_gpu
> GPU support flag.
>> **Type** bool

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

workitem_count
> Work item count.
>> **Type** int

class `Tasks.`**`AnalyzePhotos`**
> Task class containing processing parameters.

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

cameras
> List of cameras to be analyzed.
>> **Type** list of int

decode (*dict*)
> Initializaes task parameters with a dictionary.

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

encode ()
> Create a dictionary with task parameters.

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

filter_mask
> Constrain analyzed image region by mask.
>> **Type** bool

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

supports_gpu
> GPU support flag.
>> **Type** bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

class Tasks.**BuildContours**
    Task class containing processing parameters.

apply (*object* [, *workitem*] [, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

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

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

interval
    Contour interval.
        Type float

max_value
    Maximum value of contour range.
        Type float

min_value
    Minimum value of contour range.
        Type float

name
    Task name.
        Type string

prevent_intersections
    Prevent contour intersections.
        Type bool

source_data
    Source data for contour generation.
        Type *DataSource*

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

class `Tasks.`**`BuildDem`**
    Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

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

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

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

flip_x
    Flip X axis direction.
        Type bool

flip_y
    Flip Y axis direction.
        Type bool

flip_z
    Flip Z axis direction.
        Type bool

interpolation
    Interpolation mode.
        Type *Interpolation*

max_workgroup_size
    Maximum workgroup size.
        Type int

name
    Task name.
        Type string

network_distribute
    Enable distributed processing.
        Type bool

projection
    Output projection.
        Type *OrthoProjection*

region
> Region to be exported in the (x0, y0, x1, y1) format.
>> **Type** list of 4 floats

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

source_data
> Selects between dense point cloud and sparse point cloud.
>> **Type** *[DataSource](#)*

supports_gpu
> GPU support flag.
>> **Type** bool

target
> Task target.
>> **Type** *[Tasks.TargetType](#)*

workitem_count
> Work item count.
>> **Type** int

workitem_size_tiles
> Number of tiles in a workitem.
>> **Type** int

class `Tasks.`**`BuildDenseCloud`**
> Task class containing processing parameters.

> **apply**(*object*[, *workitem*][, *progress*])
>> Apply task to specified object.
>>> **Parameters**
>>> - **object** (*[Chunk](#)* or *[Document](#)*) – Chunk or Document object to be processed.
>>> - **workitem** (*int*) – Workitem index.
>>> - **progress** (*Callable[[float], None]*) – Progress callback.

> **decode**(*dict*)
>> Initializaes task parameters with a dictionary.

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

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

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

> **max_neighbors**
>> Maximum number of neighbor images to use for depth map filtering.
>>> **Type** int

> **max_workgroup_size**
>> Maximum workgroup size.
>>> **Type** int

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

network_distribute
>   Enable distributed processing.
>   > **Type** bool

point_colors
>   Enable point colors calculation.
>   > **Type** bool

store_depth
>   Enable store depth maps option.
>   > **Type** bool

supports_gpu
>   GPU support flag.
>   > **Type** bool

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

workitem_count
>   Work item count.
>   > **Type** int

workitem_size_cameras
>   Number of cameras in a workitem.
>   > **Type** int

class `Tasks.`**BuildDepthMaps**
>   Task class containing processing parameters.

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

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

decode (*dict*)
>   Initializaes task parameters with a dictionary.

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

downscale
>   Depth map quality.
>   > **Type** int

encode ()
>   Create a dictionary with task parameters.

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

filter_mode
>   Depth map filtering mode.
>   > **Type** *FilterMode*

max_neighbors
> Maximum number of neighbor images to use for depth map generation.
> > **Type** int

max_workgroup_size
> Maximum workgroup size.
> > **Type** int

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

network_distribute
> Enable distributed processing.
> > **Type** bool

reuse_depth
> Enable reuse depth maps option.
> > **Type** bool

supports_gpu
> GPU support flag.
> > **Type** bool

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

workitem_count
> Work item count.
> > **Type** int

workitem_size_cameras
> Number of cameras in a workitem.
> > **Type** int

class `Tasks.`**BuildModel**
> Task class containing processing parameters.

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

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

> **classes**
> > List of dense point classes to be used for surface extraction.
> > > **Type** list of int

> **decode** (*dict*)
> > Initializaes task parameters with a dictionary.

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

encode()
Create a dictionary with task parameters.

encodeJSON()
Create a JSON string with task parameters.

face_count
Target face count.
> **Type** *FaceCount*

face_count_custom
Custom face count.
> **Type** int

interpolation
Interpolation mode.
> **Type** *Interpolation*

name
Task name.
> **Type** string

source_data
Selects between dense point cloud, sparse point cloud and depth maps.
> **Type** *DataSource*

store_depth
Enable store depth maps option.
> **Type** bool

supports_gpu
GPU support flag.
> **Type** bool

surface_type
Type of object to be reconstructed.
> **Type** *SurfaceType*

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

trimming_radius
Trimming radius (no trimming if zero).
> **Type** int

vertex_colors
Enable vertex colors calculation.
> **Type** bool

volumetric_masks
Enable strict volumetric masking.
> **Type** bool

workitem_count
Work item count.
> **Type** int

class Tasks.**BuildOrthomosaic**
Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
>   Apply task to specified object.
>   >   **Parameters**
>   >   >   •   **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
>   >   >   •   **workitem** (*int*) – Workitem index.
>   >   >   •   **progress** (*Callable[[float], None]*) – Progress callback.

blending_mode
>   Orthophoto blending mode.
>   >   **Type** *BlendingMode*

cull_faces
>   Enable back-face culling.
>   >   **Type** bool

decode(*dict*)
>   Initializaes task parameters with a dictionary.

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

encode()
>   Create a dictionary with task parameters.

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

fill_holes
>   Enable hole filling.
>   >   **Type** bool

flip_x
>   Flip X axis direction.
>   >   **Type** bool

flip_y
>   Flip Y axis direction.
>   >   **Type** bool

flip_z
>   Flip Z axis direction.
>   >   **Type** bool

max_workgroup_size
>   Maximum workgroup size.
>   >   **Type** int

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

network_distribute
>   Enable distributed processing.
>   >   **Type** bool

ortho_surface
>   Orthorectification surface.
>   >   **Type** *DataSource*

projection
>   Output projection.

Type *OrthoProjection*

refine_seamlines
>   Refine seamlines based on image content.
>   >   **Type** bool

region
>   Region to be exported in the (x0, y0, x1, y1) format.
>   >   **Type** list of 4 floats

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

supports_gpu
>   GPU support flag.
>   >   **Type** bool

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

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 `Tasks.`**BuildSeamlines**
>   Task class containing processing parameters.

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

>   **decode**(*dict*)
>   >   Initializaes task parameters with a dictionary.

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

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

encodeJSON()
    Create a JSON string with task parameters.

epsilon
    Contour simplificaion threshold.
        Type float

name
    Task name.
        Type string

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

class `Tasks.`**BuildTexture**
    Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

blending_mode
    Texture blending mode.
        Type *BlendingMode*

cameras
    A list of cameras to be used for texturing.
        Type list of int

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

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

fill_holes
    Enable hole filling.
        Type bool

ghosting_filter
    Enable ghosting filter.
        Type bool

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

supports_gpu
>   GPU support flag.
>> **Type** bool

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

texture_size
>   Texture size.
>> **Type** int

workitem_count
>   Work item count.
>> **Type** int

class `Tasks.`**`BuildTiledModel`**
>   Task class containing processing parameters.

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

>   **classes**
>>   List of dense point classes to be used for surface extraction.
>>>   **Type** list of int

>   **decode** (*dict*)
>>   Initializaes task parameters with a dictionary.

>   **decodeJSON** (*json*)
>>   Initializaes 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

>   **max_workgroup_size**
>>   Maximum workgroup size.
>>>   **Type** int

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

>   **network_distribute**
>>   Enable distributed processing.
>>>   **Type** bool

pixel_size
: Target model resolution in meters.
    Type float

source_data
: Selects between dense point cloud and mesh.
    Type *[DataSource](#)*

store_depth
: Enable store depth maps option.
    Type bool

supports_gpu
: GPU support flag.
    Type bool

target
: Task target.
    Type *[Tasks.TargetType](#)*

tile_size
: Size of tiles in pixels.
    Type int

workitem_count
: Work item count.
    Type int

workitem_size_cameras
: Number of cameras in a workitem.
    Type int

class Tasks.**BuildUV**
: Task class containing processing parameters.

    apply(*object*[, *workitem*][, *progress*])
    : Apply task to specified object.
        Parameters
            • **object** (*[Chunk](#)* or *[Document](#)*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

    camera
    : Camera to be used for texturing in MappingCamera mode.
        Type int

    decode(*dict*)
    : Initializaes task parameters with a dictionary.

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

    encode()
    : Create a dictionary with task parameters.

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

    mapping_mode
    : Texture mapping mode.
        Type *[MappingMode](#)*

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

supports_gpu
>> GPU support flag.
>>> **Type** bool

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

texture_count
>> Texture count.
>>> **Type** int

workitem_count
>> Work item count.
>>> **Type** int

class `Tasks.`**`CalibrateColors`**
> Task class containing processing parameters.

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

> **calibrate_color_balance**
>> Turn on color balance compensation.
>>> **Type** bool

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

> **data_source**
>> Source data for calibration.
>>> **Type** *DataSource*

> **decode** (*dict*)
>> Initializaes task parameters with a dictionary.

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

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

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

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

> **supports_gpu**
>> GPU support flag.
>>> **Type** bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

class `Tasks.CalibrateLens`
    Task class containing processing parameters.

apply (*object*[, *workitem*][, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

border
    Border size to ignore.
        Type int

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

decodeJSON (*json*)
    Initializaes 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

fit_p3
> Enable optimization of p3 tangential distortion coefficient.
> > **Type** bool

fit_p4
> Enable optimization of p4 tangential distortion coefficient.
> > **Type** bool

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

supports_gpu
> GPU support flag.
> > **Type** bool

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

workitem_count
> Work item count.
> > **Type** int

class `Tasks.`**CalibrateReflectance**
> Task class containing processing parameters.

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

> **decode**(*dict*)
> > Initializaes task parameters with a dictionary.

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

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

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

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

supports_gpu
GPU support flag.
> **Type** bool

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

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 `Tasks.`**ClassifyGroundPoints**
Task class containing processing parameters.

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

cell_size
Cell size (meters).
> **Type** float

cls_from
Class of points to be re-classified.
> **Type** int

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

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

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

max_angle
Maximum angle (degrees).
> **Type** float

max_distance
Maximum distance (meters).
> **Type** float

name
Task name.
> **Type** string

supports_gpu
> GPU support flag.
> > **Type** bool

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

workitem_count
> Work item count.
> > **Type** int

class `Tasks.`**ClassifyPoints**
> Task class containing processing parameters.

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

cls_from
> Class of points to be re-classified.
> > **Type** int

cls_to
> Target point classes for classification.
> > **Type** list of int

confidence
> Required confidence level
> > **Type** float

decode (*dict*)
> Initializaes task parameters with a dictionary.

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

encode ()
> Create a dictionary with task parameters.

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

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

network_distribute
> Enable distributed processing.
> > **Type** bool

supports_gpu
> GPU support flag.
> > **Type** bool

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

> **workitem_count**
>> Work item count.
>>> **Type** int

class `Tasks.CloseHoles`
> Task class containing processing parameters.

> **apply**(*object*[, *workitem*][, *progress*])
>> Apply task to specified object.
>>> **Parameters**
>>> - **object** ([*Chunk*](#) or [*Document*](#)) – Chunk or Document object to be processed.
>>> - **workitem** (*int*) – Workitem index.
>>> - **progress** (*Callable[[float], None]*) – Progress callback.

> **decode**(*dict*)
>> Initializaes task parameters with a dictionary.

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

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

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

> **level**
>> Hole size threshold in percents.
>>> **Type** int

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

> **supports_gpu**
>> GPU support flag.
>>> **Type** bool

> **target**
>> Task target.
>>> **Type** [*Tasks.TargetType*](#)

> **workitem_count**
>> Work item count.
>>> **Type** int

class `Tasks.ColorizeDenseCloud`
> Task class containing processing parameters.

> **apply**(*object*[, *workitem*][, *progress*])
>> Apply task to specified object.
>>> **Parameters**
>>> - **object** ([*Chunk*](#) or [*Document*](#)) – Chunk or Document object to be processed.
>>> - **workitem** (*int*) – Workitem index.
>>> - **progress** (*Callable[[float], None]*) – Progress callback.

> **decode**(*dict*)
>> Initializaes task parameters with a dictionary.

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

name
    Task name.
        Type string

source_data
    Source data to extract colors from.
        Type *DataSource*

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

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

apply(*object*[, *workitem*][, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

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

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

name
    Task name.
        Type string

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.

> **Type** int

class Tasks.**DecimateModel**
Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
> Apply task to specified object.
> > **Parameters**
> > - **object** ([*Chunk*](#) or [*Document*](#)) – Chunk or Document object to be processed.
> > - **workitem** (*int*) – Workitem index.
> > - **progress** (*Callable[[float], None]*) – Progress callback.

decode(*dict*)
> Initializaes task parameters with a dictionary.

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

encode()
> Create a dictionary with task parameters.

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

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

supports_gpu
> GPU support flag.
> > **Type** bool

target
> Task target.
> > **Type** [*Tasks.TargetType*](#)

target_face_count
> Target face count.
> > **Type** int

workitem_count
> Work item count.
> > **Type** int

class Tasks.**DetectFiducials**
Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
> Apply task to specified object.
> > **Parameters**
> > - **object** ([*Chunk*](#) or [*Document*](#)) – Chunk or Document object to be processed.
> > - **workitem** (*int*) – Workitem index.
> > - **progress** (*Callable[[float], None]*) – Progress callback.

decode(*dict*)
> Initializaes task parameters with a dictionary.

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

encode()
> Create a dictionary with task parameters.

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

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

supports_gpu
>   GPU support flag.
>   >   **Type** bool

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

workitem_count
>   Work item count.
>   >   **Type** int

class `Tasks.`**DetectMarkers**
>   Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
>   Apply task to specified object.
>   >   **Parameters**
>   >   >   • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
>   >   >   • **workitem** (*int*) – Workitem index.
>   >   >   • **progress** (*Callable[[float], None]*) – Progress callback.

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

decode(*dict*)
>   Initializaes task parameters with a dictionary.

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

encode()
>   Create a dictionary with task parameters.

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

filter_mask
>   Ignore masked image regions.
>   >   **Type** bool

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

inverted
>   Detect markers on black background.
>   >   **Type** bool

maximum_residual
>   Maximum residual for non-coded targets in pixels.
>   >   **Type** float

minimum_dist
>   Minimum distance between targets in pixels (CrossTarget type only).
>   > **Type** int

minimum_size
>   Minimum target radius in pixels to be detected (CrossTarget type only).
>   > **Type** int

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

noparity
>   Disable parity checking.
>   > **Type** bool

supports_gpu
>   GPU support flag.
>   > **Type** bool

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

target_type
>   Type of targets.
>   > **Type** *TargetType*

tolerance
>   Detector tolerance (0 - 100).
>   > **Type** int

workitem_count
>   Work item count.
>   > **Type** int

class `Tasks.`**DuplicateChunk**
>   Task class containing processing parameters.

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

>   **chunk**
>   >   Chunk to copy.
>   >   > **Type** int

>   **copy_dense_cloud**
>   >   Copy dense cloud.
>   >   > **Type** bool

>   **copy_depth_maps**
>   >   Copy depth maps.
>   >   > **Type** bool

>   **copy_elevation**
>   >   Copy DEM.
>   >   > **Type** bool

copy_keypoints
Copy keypoints.
> **Type** bool

copy_model
Copy model.
> **Type** bool

copy_orthomosaic
Copy orthomosaic.
> **Type** bool

copy_tiled_model
Copy tiled model.
> **Type** bool

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

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

encode()
Create a dictionary with task parameters.

encodeJSON()
Create a JSON string with task parameters.

frames
List of frame keys to copy.
> **Type** list of int

label
New chunk label.
> **Type** string

name
Task name.
> **Type** string

supports_gpu
GPU support flag.
> **Type** bool

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

workitem_count
Work item count.
> **Type** int

class Tasks.**ExportCameras**
Task class containing processing parameters.

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

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

bingo_export_geoin
  Enables/disables export of BINGO GEO INPUT file.
    Type bool

bingo_export_gps
  Enables/disables export of BINGO GPS/IMU data.
    Type bool

bingo_export_image
  Enables/disables export of BINGO IMAGE COORDINATE file.
    Type bool

bingo_export_itera
  Enables/disables export of BINGO ITERA file.
    Type bool

bingo_path_geoin
  Path to BINGO GEO INPUT file.
    Type string

bingo_path_gps
  Path to BINGO GPS/IMU file.
    Type string

bingo_path_image
  Path to BINGO IMAGE COORDINATE file.
    Type string

bingo_path_itera
  Path to BINGO ITERA file.
    Type string

bundler_export_list
  Enables/disables export of Bundler image list file.
    Type bool

bundler_path_list
  Path to Bundler image list file.
    Type string

chan_order_rotate
  Rotation order (CHAN format only).
    Type *RotationOrder*

coordinates
  Output coordinate system.
    Type *CoordinateSystem*

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

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

encode()
  Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

export_markers
    Enables/disables export of manual matching points.
        Type bool

export_points
    Enables/disables export of automatic tie points.
        Type bool

format
    Export format.
        Type *CamerasFormat*

name
    Task name.
        Type string

path
    Path to output file.
        Type string

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

use_labels
    Enables/disables label based item identifiers.
        Type bool

workitem_count
    Work item count.
        Type int

class `Tasks.`**ExportDepth**
    Task class containing processing parameters.

    apply (*object* [, *workitem*] [, *progress*])
        Apply task to specified object.
            Parameters
                • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
                • **workitem** (*int*) – Workitem index.
                • **progress** (*Callable[[float], None]*) – Progress callback.

    cameras
        List of cameras to process.
            Type list of int

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

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

    encode ()
        Create a dictionary with task parameters.

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

export_depth
> Enable export of depth map.
>> **Type** bool

export_diffuse
> Enable export of diffuse map.
>> **Type** bool

export_normals
> Enable export of normal map.
>> **Type** bool

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

path_depth
> Path to depth map.
>> **Type** string

path_diffuse
> Path to diffuse map.
>> **Type** string

path_normals
> Path to normal map.
>> **Type** string

supports_gpu
> GPU support flag.
>> **Type** bool

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

workitem_count
> Work item count.
>> **Type** int

class Tasks.**ExportMarkers**
> Task class containing processing parameters.

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

binary
> Enables/disables binary encoding for selected format (if applicable).
>> **Type** bool

coordinates
> Output coordinate system.
>> **Type** *CoordinateSystem*

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

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

name
    Task name.
        Type string

path
    Path to output file.
        Type string

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

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

apply (*object* [, *workitem*] [, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

cameras
    List of cameras to process.
        Type list of int

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

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

name
    Task name.
        Type string

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

supports_gpu
>   GPU support flag.
>>   **Type** bool

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

workitem_count
>   Work item count.
>>   **Type** int

class Tasks.**ExportModel**
>   Task class containing processing parameters.

>   **apply** (*object* [, *workitem*] [, *progress*])
>>   Apply task to specified object.
>>   **Parameters**
>>   • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
>>   • **workitem** (*int*) – Workitem index.
>>   • **progress** (*Callable[[float], None]*) – Progress callback.

>   **binary**
>>   Enables/disables binary encoding (if supported by format).
>>   **Type** bool

>   **colors_rgb_8bit**
>>   Convert colors to 8 bit RGB.
>>   **Type** bool

>   **comment**
>>   Optional comment (if supported by selected format).
>>   **Type** string

>   **coordinates**
>>   Output coordinate system.
>>   **Type** *CoordinateSystem*

>   **decode** (*dict*)
>>   Initializaes task parameters with a dictionary.

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

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

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

>   **export_alpha**
>>   Enables/disables alpha channel export.
>>   **Type** bool

>   **export_cameras**
>>   Enables/disables camera export.
>>   **Type** bool

export_colors
　　Enables/disables export of vertex colors.
　　　　Type bool

export_comment
　　Enables/disables comment export.
　　　　Type bool

export_markers
　　Enables/disables marker export.
　　　　Type bool

export_normals
　　Enables/disables export of vertex normals.
　　　　Type bool

export_texture
　　Enables/disables texture export.
　　　　Type bool

export_udim
　　Enables/disables UDIM texture layout.
　　　　Type bool

export_uv
　　Enables/disables uv coordinates export.
　　　　Type bool

format
　　Export format.
　　　　Type *ModelFormat*

name
　　Task name.
　　　　Type string

path
　　Path to output model.
　　　　Type string

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

raster_transform
　　Raster band transformation.
　　　　Type *RasterTransformType*

shift
　　Optional shift to be applied to vertex coordinates.
　　　　Type 3-element vector

strip_camera_ext
　　Strips camera label extensions during export.
　　　　Type bool

supports_gpu
　　GPU support flag.
　　　　Type bool

target
    Task target.
        Type *Tasks.TargetType*

texture_format
    Texture format.
        Type *ImageFormat*

viewpoint
    Default view.
        Type *Viewpoint*

workitem_count
    Work item count.
        Type int

class `Tasks.`**ExportOrthophotos**
    Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

cameras
    List of cameras to process.
        Type list of int

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

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

jpeg_quality
    JPEG quality.
        Type int

name
    Task name.
        Type string

north_up
    Use north-up orientation for export.
        Type bool

path
    Path to output orthophoto.
        Type string

projection
    Output projection.
        Type *OrthoProjection*

raster_transform
>   Raster band transformation.
>   **Type** *RasterTransformType*

region
>   Region to be exported in the (x0, y0, x1, y1) format.
>   **Type** list of 4 floats

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

supports_gpu
>   GPU support flag.
>   **Type** bool

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

tiff_big
>   Enable BigTIFF compression for TIFF files.
>   **Type** bool

tiff_compression
>   Tiff compression.
>   **Type** int

tiff_overviews
>   Enable image pyramid deneration for TIFF files.
>   **Type** bool

tiff_tiled
>   Export tiled TIFF.
>   **Type** bool

workitem_count
>   Work item count.
>   **Type** int

write_alpha
>   Enable alpha channel generation.
>   **Type** bool

write_kml
>   Enable kml file generation.
>   **Type** bool

write_world
>   Enable world file generation.
>   **Type** bool

class `Tasks.`**`ExportPanorama`**
>   Task class containing processing parameters.

>   **`apply`**(*object*[, *workitem*][, *progress*])
>   >   Apply task to specified object.
>   >   >   **Parameters**
>   >   >   - **`object`** (*`Chunk`* or *`Document`*) – Chunk or Document object to be processed.
>   >   >   - **`workitem`** (*`int`*) – Workitem index.
>   >   >   - **`progress`** (*`Callable[[float], None]`*) – Progress callback.

>   **`camera_groups`**
>   >   List of camera groups to process.
>   >   >   **Type** list of int

>   **`decode`**(*dict*)
>   >   Initializaes task parameters with a dictionary.

>   **`decodeJSON`**(*json*)
>   >   Initializaes task parameters from a JSON string.

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

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

>   **`height`**
>   >   Height of output panorama.
>   >   >   **Type** int

>   **`name`**
>   >   Task name.
>   >   >   **Type** string

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

>   **`region`**
>   >   Region to be exported in the (x0, y0, x1, y1) format.
>   >   >   **Type** list of 4 floats

>   **`rotation`**
>   >   Panorama 3x3 orientation matrix.
>   >   >   **Type** *`Matrix`*

>   **`supports_gpu`**
>   >   GPU support flag.
>   >   >   **Type** bool

>   **`target`**
>   >   Task target.
>   >   >   **Type** *`Tasks.TargetType`*

>   **`width`**
>   >   Width of output panorama.
>   >   >   **Type** int

>   **`workitem_count`**
>   >   Work item count.
>   >   >   **Type** int

class `Tasks.`**`ExportPoints`**
    Task class containing processing parameters.

    `apply`(*object*[, *workitem*][, *progress*])
        Apply task to specified object.
            Parameters
                • **`object`** (*Chunk* or *Document*) – Chunk or Document object to be processed.
                • **`workitem`** (*int*) – Workitem index.
                • **`progress`** (*Callable[[float], None]*) – Progress callback.

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

    `classes`
        List of dense point classes to be exported.
            Type list of int

    `colors_rgb_8bit`
        Convert colors to 8 bit RGB.
            Type bool

    `comment`
        Optional comment (if supported by selected format).
            Type string

    `coordinates`
        Output coordinate system.
            Type *CoordinateSystem*

    `data_source`
        Selects between dense point cloud and sparse point cloud. If not specified, uses dense cloud if available.
            Type *DataSource*

    `decode`(*dict*)
        Initializaes task parameters with a dictionary.

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

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

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

    `export_colors`
        Enables/disables export of point colors.
            Type bool

    `export_comment`
        Enable comment export.
            Type bool

    `export_images`
        Enable image export.
            Type bool

    `export_normals`
        Enables/disables export of point normals.

> **Type** bool

format
> Export format.
> > **Type** *PointsFormat*

image_format
> Image data format.
> > **Type** *ImageFormat*

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

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

precision
> Number of digits after the decimal point (for text formats).
> > **Type** int

raster_transform
> Raster band transformation.
> > **Type** *RasterTransformType*

shift
> Optional shift to be applied to vertex coordinates.
> > **Type** 3-element vector

supports_gpu
> GPU support flag.
> > **Type** bool

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

tile_height
> Tile height in meters.
> > **Type** float

tile_width
> Tile width in meters.
> > **Type** float

viewpoint
> Default view.
> > **Type** *Viewpoint*

workitem_count
> Work item count.
> > **Type** int

write_tiles
> Enable tiled export.
> > **Type** bool

class Tasks.**ExportRaster**
> Task class containing processing parameters.

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

data_source
Selects between DEM and orthomosaic.
> **Type** *DataSource*

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

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

description
Export description.
> **Type** string

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

format
Export format.
> **Type** *RasterFormat*

height
Raster height.
> **Type** int

image_description
Optional description to be added to image files.
> **Type** string

image_format
Tile format.
> **Type** *ImageFormat*

jpeg_quality
JPEG quality.
> **Type** int

kmz_section_enable
Enable network links generation for KMZ format.
> **Type** bool

name
Task name.
> **Type** string

nodata_value
No-data value (DEM export only).
> **Type** float

north_up
Use north-up orientation for export.

   Type bool

path
 Path to output orthomosaic.
  Type string

projection
 Output projection.
  Type *OrthoProjection*

raster_transform
 Raster band transformation.
  Type *RasterTransformType*

region
 Region to be exported in the (x0, y0, x1, y1) format.
  Type list of 4 floats

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

supports_gpu
 GPU support flag.
  Type bool

target
 Task target.
  Type *Tasks.TargetType*

tiff_big
 Enable BigTIFF compression for TIFF files.
  Type bool

tiff_compression
 Tiff compression.
  Type int

tiff_overviews
 Enable image pyramid deneration for TIFF files.
  Type bool

tiff_tiled
 Export tiled TIFF.
  Type bool

tile_height
 Specifies block height of the orthomosaic in pixels.
  Type int

tile_width
 Specifies block width of the orthomosaic in pixels.
  Type int

title
> Export title.
>> **Type** string

white_background
> Enable white background.
>> **Type** bool

width
> Raster width.
>> **Type** int

workitem_count
> Work item count.
>> **Type** int

world_transform
> 2x3 raster-to-world transformation matrix.
>> **Type** *Matrix*

write_alpha
> Enable alpha channel generation.
>> **Type** bool

write_kml
> Enable kml file generation.
>> **Type** bool

write_scheme
> Enable tile scheme files generation.
>> **Type** bool

write_tiles
> Enable tiled export.
>> **Type** bool

write_world
> Enable world file generation.
>> **Type** bool

xyz_level_max
> Maximum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).
>> **Type** int

xyz_level_min
> Minimum zoom level (Google Map Tiles, MBTiles and World Wind Tiles formats only).
>> **Type** int

class `Tasks.`**ExportReference**
> Task class containing processing parameters.

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

columns
> Column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, u/v/w - estimated coordinates,

U/V/W - coordinate errors, d/e/f - estimated orientation angles, D/E/F - orientation errors, [] - group of multiple values, | - column separator within group).

> **Type** string

decode(*dict*)
> Initializaes task parameters with a dictionary.

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

delimiter
> Column delimiter in csv format.
>> **Type** string

encode()
> Create a dictionary with task parameters.

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

format
> Export format.
>> **Type** *ReferenceFormat*

items
> Items to export in CSV format.
>> **Type** *ReferenceItems*

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

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

supports_gpu
> GPU support flag.
>> **Type** bool

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

workitem_count
> Work item count.
>> **Type** int

class Tasks.**ExportReport**
> Task class containing processing parameters.

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

decode(*dict*)
> Initializaes task parameters with a dictionary.

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

description
> Report description.
>> **Type** string

encode ()
> Create a dictionary with task parameters.

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

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

page_numbers
> Enable page numbers.
>> **Type** bool

path
> Path to output report.
>> **Type** string

supports_gpu
> GPU support flag.
>> **Type** bool

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

title
> Report title.
>> **Type** string

workitem_count
> Work item count.
>> **Type** int

class Tasks.**ExportShapes**
> Task class containing processing parameters.

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

coordinates
> Output coordinate system.
>> **Type** *CoordinateSystem*

decode (*dict*)
> Initializaes task parameters with a dictionary.

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

export_attributes
    Export attributes.
        Type bool

export_labels
    Export labels.
        Type bool

export_points
    Export points.
        Type bool

export_polygons
    Export polygons.
        Type bool

export_polylines
    Export polylines.
        Type bool

format
    Export format.
        Type *ShapesFormat*

groups
    A list of shape groups to export.
        Type list of int

name
    Task name.
        Type string

path
    Path to shape file.
        Type string

polygons_as_polylines
    Save polygons as polylines.
        Type bool

shift
    Optional shift to be applied to vertex coordinates.
        Type 3-element vector

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

class `Tasks.`**`ExportTexture`**
    Task class containing processing parameters.

    `apply`(*object*[, *workitem*][, *progress*])
        Apply task to specified object.
            Parameters
                • **`object`** (*Chunk* or *Document*) – Chunk or Document object to be processed.
                • **`workitem`** (*int*) – Workitem index.
                • **`progress`** (*Callable[[float], None]*) – Progress callback.

    `decode`(*dict*)
        Initializaes task parameters with a dictionary.

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

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

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

    `name`
        Task name.
            Type string

    `path`
        Path to output file.
            Type string

    `supports_gpu`
        GPU support flag.
            Type bool

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

    `workitem_count`
        Work item count.
            Type int

    `write_alpha`
        Enable alpha channel export.
            Type bool

class `Tasks.`**`ExportTiledModel`**
    Task class containing processing parameters.

    `apply`(*object*[, *workitem*][, *progress*])
        Apply task to specified object.
             Parameters
                • **`object`** (*Chunk* or *Document*) – Chunk or Document object to be processed.
                • **`workitem`** (*int*) – Workitem index.
                • **`progress`** (*Callable[[float], None]*) – Progress callback.

    `decode`(*dict*)
        Initializaes task parameters with a dictionary.

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

encode()
>   Create a dictionary with task parameters.

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

format
>   Export format.
>   **Type** *TiledModelFormat*

mesh_format
>   Mesh format for zip export.
>   **Type** *ModelFormat*

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

path
>   Path to output model.
>   **Type** string

raster_transform
>   Raster band transformation.
>   **Type** *RasterTransformType*

supports_gpu
>   GPU support flag.
>   **Type** bool

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

workitem_count
>   Work item count.
>   **Type** int

class Tasks.**ImportCameras**
>   Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
>   Apply task to specified object.
>   **Parameters**
>   • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
>   • **workitem** (*int*) – Workitem index.
>   • **progress** (*Callable[[float], None]*) – Progress callback.

decode(*dict*)
>   Initializaes task parameters with a dictionary.

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

encode()
>   Create a dictionary with task parameters.

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

format
>   File format.

> > > **Type** *CamerasFormat*

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

path
> Path to the file.
> > **Type** string

supports_gpu
> GPU support flag.
> > **Type** bool

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

workitem_count
> Work item count.
> > **Type** int

class `Tasks.`**`ImportDem`**
> Task class containing processing parameters.

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

coordinates
> Default coordinate system if not specified in GeoTIFF file.
> > **Type** *CoordinateSystem*

decode (*dict*)
> Initializaes task parameters with a dictionary.

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

encode ()
> Create a dictionary with task parameters.

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

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

path
> Path to elevation model in GeoTIFF format.
> > **Type** string

supports_gpu
> GPU support flag.
> > **Type** bool

target
> Task target.

> **Type** [*Tasks.TargetType*](#)

workitem_count
> Work item count.
> > **Type** int

class `Tasks.`**`ImportMarkers`**
> Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
> Apply task to specified object.
> > **Parameters**
> > - **object** ([*Chunk*](#) or [*Document*](#)) – Chunk or Document object to be processed.
> > - **workitem** (*int*) – Workitem index.
> > - **progress** (*Callable[[float], None]*) – Progress callback.

decode(*dict*)
> Initializaes task parameters with a dictionary.

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

encode()
> Create a dictionary with task parameters.

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

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

path
> Path to the file.
> > **Type** string

supports_gpu
> GPU support flag.
> > **Type** bool

target
> Task target.
> > **Type** [*Tasks.TargetType*](#)

workitem_count
> Work item count.
> > **Type** int

class `Tasks.`**`ImportMasks`**
> Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
> Apply task to specified object.
> > **Parameters**
> > - **object** ([*Chunk*](#) or [*Document*](#)) – Chunk or Document object to be processed.
> > - **workitem** (*int*) – Workitem index.
> > - **progress** (*Callable[[float], None]*) – Progress callback.

cameras
> Optional list of cameras to be processed.
> > **Type** list of int

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

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

method
    Mask source.
        Type *MaskSource*

name
    Task name.
        Type string

operation
    Mask operation.
        Type *MaskOperation*

path
    Mask file name template.
        Type string

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

tolerance
    Background masking tolerance.
        Type int

workitem_count
    Work item count.
        Type int

class Tasks.**ImportModel**
    Task class containing processing parameters.

apply (*object* [, *workitem*] [, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

coordinates
    Model coordinate system.
        Type *CoordinateSystem*

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

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

encode ()
> Create a dictionary with task parameters.

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

format
> Model format.
> > **Type** *ModelFormat*

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

path
> Path to model.
> > **Type** string

shift
> Optional shift to be applied to vertex coordinates.
> > **Type** 3-element vector

supports_gpu
> GPU support flag.
> > **Type** bool

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

workitem_count
> Work item count.
> > **Type** int

class Tasks.**ImportPoints**
> Task class containing processing parameters.

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

coordinates
> Point cloud coordinate system.
> > **Type** *CoordinateSystem*

decode (*dict*)
> Initializaes task parameters with a dictionary.

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

encode ()
> Create a dictionary with task parameters.

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

format
   Point cloud format.
      Type *PointsFormat*

name
   Task name.
      Type string

path
   Path to point cloud.
      Type string

shift
   Optional shift to be applied to point coordinates.
      Type 3-element vector

supports_gpu
   GPU support flag.
      Type bool

target
   Task target.
      Type *Tasks.TargetType*

workitem_count
   Work item count.
      Type int

class `Tasks.`**ImportReference**
   Task class containing processing parameters.

   apply (*object* [, *workitem*] [, *progress*])
      Apply task to specified object.
         Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

   columns
      Column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, [] - group of multiple values, | - column separator within group).
         Type string

   coordinates
      Reference data coordinate system (csv format only).
         Type *CoordinateSystem*

   create_markers
      Create markers for missing entries (csv format only).
         Type bool

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

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

   delimiter
      Column delimiter in csv format.
         Type string

encode()
Create a dictionary with task parameters.

encodeJSON()
Create a JSON string with task parameters.

format
File format.
> **Type** *ReferenceFormat*

group_delimiters
Combine consequitive delimiters in csv format.
> **Type** bool

ignore_labels
Matches reference data based on coordinates alone (csv format only).
> **Type** bool

items
List of items to load reference for (csv format only).
> **Type** *ReferenceItems*

name
Task name.
> **Type** string

path
Path to the file with reference data.
> **Type** string

shutter_lag
Shutter lag in seconds (APM format only).
> **Type** float

skip_rows
Number of rows to skip in (csv format only).
> **Type** int

supports_gpu
GPU support flag.
> **Type** bool

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

threshold
Error threshold in meters used when ignore_labels is set (csv format only).
> **Type** float

workitem_count
Work item count.
> **Type** int

class Tasks.**ImportShapes**
Task class containing processing parameters.

apply(*object*[, *workitem*][, *progress*])
Apply task to specified object.
> **Parameters**
> - **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.

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

boundary_type
　　Boundary type to be applied to imported shapes.
　　　　Type *Shape.BoundaryType*

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

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

encode ()
　　Create a dictionary with task parameters.

encodeJSON ()
　　Create a JSON string with task parameters.

format
　　Shapes format.
　　　　Type *ShapesFormat*

name
　　Task name.
　　　　Type string

path
　　Path to shape file.
　　　　Type string

replace
　　Replace current shapes with new data.
　　　　Type bool

supports_gpu
　　GPU support flag.
　　　　Type bool

target
　　Task target.
　　　　Type *Tasks.TargetType*

workitem_count
　　Work item count.
　　　　Type int

class Tasks.**InvertMasks**
　　Task class containing processing parameters.

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

cameras
　　List of cameras to process.
　　　　Type list of int

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

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

encode()
Create a dictionary with task parameters.

encodeJSON()
Create a JSON string with task parameters.

name
Task name.
> **Type** string

supports_gpu
GPU support flag.
> **Type** bool

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

workitem_count
Work item count.
> **Type** int

class Tasks.**LoadProject**
Task class containing processing parameters.

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

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

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

encode()
Create a dictionary with task parameters.

encodeJSON()
Create a JSON string with task parameters.

name
Task name.
> **Type** string

path
Path to project file.
> **Type** string

read_only
Open project in read only mode.
> **Type** bool

supports_gpu
>   GPU support flag.
>> **Type**   bool

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

workitem_count
>   Work item count.
>> **Type**   int

class `Tasks.`**MatchPhotos**
>   Task class containing processing parameters.

apply (*object* [*, workitem*] [*, progress*])
>   Apply task to specified object.
>> **Parameters**
>>> • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
>>> • **workitem** (*int*) – Workitem index.
>>> • **progress** (*Callable[[float], None]*) – Progress callback.

cameras
>   List of cameras to match.
>> **Type**   list of int

decode (*dict*)
>   Initializaes task parameters with a dictionary.

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

downscale
>   Image alignment accuracy.
>> **Type**   int

encode ()
>   Create a dictionary with task parameters.

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

filter_mask
>   Filter points by mask.
>> **Type**   bool

keypoint_limit
>   Key point limit.
>> **Type**   int

mask_tiepoints
>   Apply mask filter to tie points.
>> **Type**   bool

max_workgroup_size
>   Maximum workgroup size.
>> **Type**   int

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

network_distribute
> Enable distributed processing.
> > **Type** bool

pairs
> User defined list of camera pairs to match.
> > **Type** list of int

preselection_generic
> Enable generic preselection.
> > **Type** bool

preselection_reference
> Enable reference preselection.
> > **Type** bool

reset_matches
> Reset current matches.
> > **Type** bool

store_keypoints
> Store keypoints in the project.
> > **Type** bool

supports_gpu
> GPU support flag.
> > **Type** bool

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

tiepoint_limit
> Tie point limit.
> > **Type** int

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 `Tasks.`**MergeAssets**
> Task class containing processing parameters.

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

assets
> List of assets to process.
> > **Type** list of int

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

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

name
    Task name.
        Type string

source_data
    Asset type.
        Type *DataSource*

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

class `Tasks.`**MergeChunks**
    Task class containing processing parameters.

    apply(*object*[, *workitem*][, *progress*])
        Apply task to specified object.
            Parameters
                • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
                • **workitem** (*int*) – Workitem index.
                • **progress** (*Callable[[float], None]*) – Progress callback.

    chunks
        List of chunks to process.
            Type list of int

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

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

    encode()
        Create a dictionary with task parameters.

    encodeJSON()
        Create a JSON string with task parameters.

    merge_dense_clouds
        Merge dense clouds.
            Type bool

merge_markers
Merge markers.
> **Type** bool

merge_models
Merge models.
> **Type** bool

merge_tiepoints
Merge tie points.
> **Type** bool

name
Task name.
> **Type** string

supports_gpu
GPU support flag.
> **Type** bool

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

workitem_count
Work item count.
> **Type** int

class Tasks.**OptimizeCameras**
Task class containing processing parameters.

adaptive_fitting
Enable adaptive fitting of distortion coefficients.
> **Type** bool

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

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

decodeJSON(*json*)
Initializaes 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_cx
> Enable optimization of X principal point coordinates.
>> **Type** bool

fit_cy
> Enable optimization of Y principal point coordinates.
>> **Type** bool

fit_f
> Enable optimization of focal length coefficient.
>> **Type** bool

fit_k1
> Enable optimization of k1 radial distortion coefficient.
>> **Type** bool

fit_k2
> Enable optimization of k2 radial distortion coefficient.
>> **Type** bool

fit_k3
> Enable optimization of k3 radial distortion coefficient.
>> **Type** bool

fit_k4
> Enable optimization of k3 radial distortion coefficient.
>> **Type** bool

fit_p1
> Enable optimization of p1 tangential distortion coefficient.
>> **Type** bool

fit_p2
> Enable optimization of p2 tangential distortion coefficient.
>> **Type** bool

fit_p3
> Enable optimization of p3 tangential distortion coefficient.
>> **Type** bool

fit_p4
> Enable optimization of p4 tangential distortion coefficient.
>> **Type** bool

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

supports_gpu
> GPU support flag.
>> **Type** bool

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

tiepoint_covariance
> Estimate tie point covariance matrices.
>> **Type** bool

workitem_count
    Work item count.
        Type int

class `Tasks.`**`PlanMission`**
    Task class containing processing parameters.

    apply(*object*[, *workitem*][, *progress*])
        Apply task to specified object.
            Parameters
                • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
                • **workitem** (*int*) – Workitem index.
                • **progress** (*Callable[[float], None]*) – Progress callback.

    capture_distance
        Image capture distance (m).
            Type float

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

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

    encode()
        Create a dictionary with task parameters.

    encodeJSON()
        Create a JSON string with task parameters.

    max_cameras
        Maximum cameras to use.
            Type int

    max_waypoints
        Maximum waypoints per flight.
            Type int

    min_altitude
        Minimum altitude (m).
            Type float

    min_waypoint_spacing
        Minimum waypoint spacing (m).
            Type float

    name
        Task name.
            Type string

    overlap
        Overlap level.
            Type int

    safety_distance
        Safety distance (m).
            Type float

    sensor
        Sensor key.
            Type int

> **supports_gpu**
> > GPU support flag.
> > > **Type** bool

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

> **use_cameras**
> > Complete current survey.
> > > **Type** bool

> **use_selection**
> > Focus on model selection.
> > > **Type** bool

> **workitem_count**
> > Work item count.
> > > **Type** int

class `Tasks.`**`ReduceOverlap`**
> Task class containing processing parameters.

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

> **capture_distance**
> > Image capture distance (m).
> > > **Type** float

> **decode** (*dict*)
> > Initializaes task parameters with a dictionary.

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

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

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

> **max_cameras**
> > Maximum cameras to use.
> > > **Type** int

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

> **overlap**
> > Overlap level.
> > > **Type** int

> **supports_gpu**
> > GPU support flag.
> > > **Type** bool

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

use_selection
>   Focus on model selection.
>   >   **Type** bool

workitem_count
>   Work item count.
>   >   **Type** int

class `Tasks.`**`RefineMesh`**
>   Task class containing processing parameters.

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

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

>   **decode**(*dict*)
>   >   Initializaes task parameters with a dictionary.

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

>   **downscale**
>   >   Refinement quality.
>   >   >   **Type** int

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

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

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

>   **niterations**
>   >   Number of refinement iterations.
>   >   >   **Type** int

>   **smoothness**
>   >   Smoothing strength. Should be in range [0, 1].
>   >   >   **Type** float

>   **supports_gpu**
>   >   GPU support flag.
>   >   >   **Type** bool

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

workitem_count
> Work item count.
>> **Type** int

class `Tasks.RemoveLighting`
> Task class containing processing parameters.

ambient_occlusion_multiplier
> Ambient occlusion multiplier. Should be in range [0.25, 4].
>> **Type** float

ambient_occlusion_path
> Path to ambient occlusion texture atlas. Can be empty.
>> **Type** string

apply (*object* [, *workitem*] [, *progress*])
> Apply task to specified object.
>> **Parameters**
>>> • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
>>> • **workitem** (*int*) – Workitem index.
>>> • **progress** (*Callable[[float], None]*) – Progress callback.

color_mode
> Enable multi-color processing mode.
>> **Type** bool

decode (*dict*)
> Initializaes task parameters with a dictionary.

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

encode ()
> Create a dictionary with task parameters.

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

internal_blur
> Internal blur. Should be in range [0, 4].
>> **Type** float

mesh_noise_suppression
> Mesh normals noise suppression strength. Should be in range [0, 4].
>> **Type** float

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

supports_gpu
> GPU support flag.
>> **Type** bool

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

workitem_count
> Work item count.
>> **Type** int

class `Tasks.`**`ResetMasks`**
Task class containing processing parameters.

`apply`(*object*[, *workitem*][, *progress*])
Apply task to specified object.
> **Parameters**
> - **`object`** (*Chunk* or *Document*) – Chunk or Document object to be processed.
> - **`workitem`** (*int*) – Workitem index.
> - **`progress`** (*Callable[[float], None]*) – Progress callback.

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

`decode`(*dict*)
Initializaes task parameters with a dictionary.

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

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

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

`name`
Task name.
> **Type** string

`supports_gpu`
GPU support flag.
> **Type** bool

`target`
Task target.
> **Type** *Tasks.TargetType*

`workitem_count`
Work item count.
> **Type** int

class `Tasks.`**`RunScript`**
Task class containing processing parameters.

`apply`(*object*[, *workitem*][, *progress*])
Apply task to specified object.
> **Parameters**
> - **`object`** (*Chunk* or *Document*) – Chunk or Document object to be processed.
> - **`workitem`** (*int*) – Workitem index.
> - **`progress`** (*Callable[[float], None]*) – Progress callback.

`args`
Script arguments.
> **Type** string

`code`
Script code.
> **Type** string

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

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

name
    Task name.
        Type string

path
    Script path.
        Type string

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

workitem_count
    Work item count.
        Type int

class Tasks.**SaveProject**
    Task class containing processing parameters.

absolute_paths
    Store absolute image paths.
        Type bool

apply(*object*[, *workitem*][, *progress*])
    Apply task to specified object.
        Parameters
            • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
            • **workitem** (*int*) – Workitem index.
            • **progress** (*Callable[[float], None]*) – Progress callback.

chunks
    List of chunks to be saved.
        Type list of int

compression
    Project compression level.
        Type int

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

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

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

name
    Task name.
        Type string

path
    Path to project.
        Type string

supports_gpu
    GPU support flag.
        Type bool

target
    Task target.
        Type *Tasks.TargetType*

version
    Project version to save.
        Type string

workitem_count
    Work item count.
        Type int

class Tasks.**SmoothModel**
    Task class containing processing parameters.

    apply(*object*[, *workitem*][, *progress*])
        Apply task to specified object.
            Parameters
                • **object** (*Chunk* or *Document*) – Chunk or Document object to be processed.
                • **workitem** (*int*) – Workitem index.
                • **progress** (*Callable[[float], None]*) – Progress callback.

    apply_to_selected
        Apply to selected faces.
            Type bool

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

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

    encode()
        Create a dictionary with task parameters.

    encodeJSON()
        Create a JSON string with task parameters.

    fix_borders
        Fix borders.
            Type bool

    name
        Task name.
            Type string

strength
> Smoothing strength.
>> **Type** float

supports_gpu
> GPU support flag.
>> **Type** bool

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

workitem_count
> Work item count.
>> **Type** int

class Tasks.**TargetType**
> Task target type in [DocumentTarget, ChunkTarget, FrameTarget]

class Tasks.**TrackMarkers**
> Task class containing processing parameters.

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

> **decode** (*dict*)
>> Initializaes task parameters with a dictionary.

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

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

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

> **frame_end**
>> Ending frame index.
>>> **Type** int

> **frame_start**
>> Starting frame index.
>>> **Type** int

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

> **supports_gpu**
>> GPU support flag.
>>> **Type** bool

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

workitem_count
> Work item count.
>> **Type** int

class `Tasks.`**`TriangulatePoints`**
> Task class containing processing parameters.

> **apply**(*object*[, *workitem*][, *progress*])
>> Apply task to specified object.
>>> **Parameters**
>>>> • **object** ([*Chunk*](#) or [*Document*](#)) – Chunk or Document object to be processed.
>>>> • **workitem** (*int*) – Workitem index.
>>>> • **progress** (*Callable[[float], None]*) – Progress callback.

> **decode**(*dict*)
>> Initializaes task parameters with a dictionary.

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

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

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

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

> **supports_gpu**
>> GPU support flag.
>>> **Type** bool

> **target**
>> Task target.
>>> **Type** [*Tasks.TargetType*](#)

> **workitem_count**
>> Work item count.
>>> **Type** int

class `Tasks.`**`UndistortPhotos`**
> Task class containing processing parameters.

> **apply**(*object*[, *workitem*][, *progress*])
>> Apply task to specified object.
>>> **Parameters**
>>>> • **object** ([*Chunk*](#) or [*Document*](#)) – Chunk or Document object to be processed.
>>>> • **workitem** (*int*) – Workitem index.
>>>> • **progress** (*Callable[[float], None]*) – Progress callback.

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

> **color_correction**
>> Apply color correction.
>>> **Type** bool

decode(*dict*)
> Initializaes task parameters with a dictionary.

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

encode()
> Create a dictionary with task parameters.

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

fix_pixel_aspect
> Fix pixel aspect.
>> **Type** bool

fix_principal_point
> Fix principal point.
>> **Type** bool

jpeg_quality
> JPEG quality.
>> **Type** int

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

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

remove_distortions
> Remove distortions.
>> **Type** bool

supports_gpu
> GPU support flag.
>> **Type** bool

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

tiff_compression
> Tiff compression.
>> **Type** int

update_gps_tags
> Update GPS tags.
>> **Type** bool

workitem_count
> Work item count.
>> **Type** int

Tasks.**createTask**(*name*)
> Create task object by its name.

>> **Parameters** **name** (*string*) – Task name.

>> **Returns** Task object.

Return type object

class Metashape.**Thumbnail**
> Thumbnail instance

> **copy**()
> > Returns a copy of thumbnail.

> > **Returns** Copy of thumbnail.

> > **Return type** *[Thumbnail](#)*

> **image**()
> > Returns image data.

> > **Returns** Image data.

> > **Return type** *[Image](#)*

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

> > **Parameters**

> > > • **path** (*string*) – Path to the image file to be loaded.

> > > • **layer** (*int*) – Optional layer index in case of multipage files.

> **setImage**(*image*)

> > **Parameters** **image** (*[Image](#)*) – Image object with thumbnail data.

class Metashape.**Thumbnails**
> A set of thumbnails generated for a chunk frame.

> **items**()
> > List of items.

> **keys**()
> > List of item keys.

> **meta**
> > Thumbnails meta data.

> > **Type** *[MetaData](#)*

> **modified**
> > Modified flag.

> > **Type** bool

> **values**()
> > List of item values.

class Metashape.**TiffCompression**
> Tiff compression in [TiffCompressionNone, TiffCompressionLZW, TiffCompressionJPEG, TiffCompression-Packbits, TiffCompressionDeflate]

class Metashape.**TiledModel**
> Tiled model data.

> **bands**
> > List of color bands.

> > **Type** list of string

clear()
> Clears tiled model data.

copy()
> Create a copy of the tiled model.

> > **Returns** Copy of the tiled model.

> > **Return type** *TiledModel*

data_type
> Data type used to store color values.

> > **Type** *DataType*

key
> Tiled model identifier.

> > **Type** int

label
> Tiled model label.

> > **Type** string

meta
> Tiled model meta data.

> > **Type** *MetaData*

modified
> Modified flag.

> > **Type** bool

pickPoint(*origin*, *target*)
> Returns ray intersection with the tiled model.

> > **Parameters**

> > > • **origin** (*Vector*) – Ray origin.

> > > • **target** (*Vector*) – Point on the ray.

> > **Returns** Coordinates of the intersection point.

> > **Return type** *Vector*

class Metashape.**TiledModelFormat**
> Tiled model format in [TiledModelFormatNone, TiledModelFormatTLS, TiledModelFormatLOD, TiledModelFormatZIP, TiledModelFormatCesium, TiledModelFormatSLPK, TiledModelFormatOSGB]

class Metashape.**Utils**
> Utility functions.

createChessboardImage(*calib*, *cell_size=150*, *max_tilt=30*)
> Synthesizes photo of a chessboard.

> > **Parameters**

> > > • **calib** (*Calibration*) – Camera calibration.

> > > • **cell_size** (*float*) – Chessboard cell size.

> > > • **max_tilt** (*float*) – Maximum camera tilt in degrees.

> > **Returns** Resulting image.

Return type *[Image](#)*

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

  Parameters

  - **image** (*[Image](#)*) – Image to be masked.

  - **background** (*[Image](#)* or color tuple) – Background image or color value.

  - **tolerance** (*int*) – Tolerance value.

  - **fit_colors** (*bool*) – Enables white balance correction.

  Returns Resulting mask.

  Return type *[Image](#)*

createMarkers (*chunk*, *projections*)
  Creates markers from a list of non coded projections.

  Parameters

  - **chunk** (*[Chunk](#)*) – Chunk to create markers in.

  - **projections** (list of (*[Camera](#)*, *[Target](#)*) tuples) – List of marker projections.

detectTargets (*image*, *type=TargetCircular12bit*, *tolerance=50*, *inverted=False*, *noparity=False*[, *minimum_size*][, *minimum_dist*])
  Detect targets on the image.

  Parameters

  - **image** (*[Image](#)*) – Image to process.

  - **type** (*[TargetType](#)*) – Type of targets.

  - **tolerance** (*int*) – Detector tolerance (0 - 100).

  - **inverted** (*bool*) – Detect markers on black background.

  - **noparity** (*bool*) – Disable parity checking.

  - **minimum_size** (*int*) – Minimum target radius in pixels to be detected (CrossTarget type only).

  - **minimum_dist** (*int*) – Minimum distance between targets in pixels (CrossTarget type only).

  Returns List of detected targets.

  Return type list of *[Target](#)*

estimateImageQuality (*image*[, *mask*])
  Estimate image sharpness.

  Parameters

  - **image** (*[Image](#)*) – Image to be analyzed.

  - **mask** (*[Image](#)*) – Mask of the analyzed image region.

  Returns Quality metric.

  Return type float

mat2opk (*R*)
  Calculate omega, phi, kappa from camera to world rotation matrix.

> **Parameters R** (*Matrix*) – Rotation matrix.

> **Returns** Omega, phi, kappa angles in degrees.

> **Return type** *Vector*

mat2ypr(*R*)

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

> **Parameters R** (*Matrix*) – Rotation matrix.

> **Returns** Yaw, pitch roll angles in degrees.

> **Return type** *Vector*

opk2mat(*angles*)

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

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

> **Returns** Rotation matrix.

> **Return type** *Matrix*

ypr2mat(*angles*)

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

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

> **Returns** Rotation matrix.

> **Return type** *Matrix*

class Metashape.**Vector**

n-component vector

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

copy()

Return a copy of the vector.

> **Returns** A copy of the vector.

> **Return type** *Vector*

cross(*a*, *b*)

Cross product of 2 vectors.

> **Parameters**
>
> - **a** (*Vector*) – First vector.
>
> - **b** (*Vector*) – Second vector.

> **Returns** Cross product.

> **Return type** *Vector*

norm()
> Return norm of the vector.

norm2()
> Return squared norm of the vector.

normalize()
> Normalize vector to the unit length.

normalized()
> Return a new, normalized vector.
>
> > **Returns** a normalized copy of the vector
> >
> > **Return type** *Vector*

size
> Vector dimensions.
>
> > **Type** int

w
> Vector W component.
>
> > **Type** float

x
> Vector X component.
>
> > **Type** float

y
> Vector Y component.
>
> > **Type** float

z
> Vector Z component.
>
> > **Type** float

zero()
> Set all elements to zero.

class Metashape.**Version**
> Version object contains application version numbers.

build
> Build number.
>
> > **Type** int

major
> Major version number.
>
> > **Type** int

micro
> Micro version number.
>
> > **Type** int

minor
> Minor version number.
>
> > **Type** int

class `Metashape.`**`Viewpoint`**(*app*)
   Represents viewpoint in the model view

   center
      Camera center.

         Type *Vector*

   coo
      Center of orbit.

         Type *Vector*

   fov
      Camera vertical field of view in degrees.

         Type float

   height
      OpenGL window height.

         Type int

   mag
      Camera magnification defined by distance to the center of rotation.

         Type float

   rot
      Camera rotation matrix.

         Type *Matrix*

   width
      OpenGL window width.

         Type int

class `Metashape.`**`Vignetting`**
   Vignetting polynomial

PYTHON API CHANGE LOG

3.1 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

3.2 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

3.3 Metashape version 1.5.1

- Added License class

- Added Tasks.MergeAssets class

- Added Metashape.license attribute

- Renamed Tasks.OptimizeCoverage class to Tasks.PlanMotion

3.4 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 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

3.5 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

3.6 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

3.7 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

3.8 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

3.9 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

3.10 PhotoScan version 1.3.5

No Python API changes

3.11 PhotoScan version 1.3.4

No Python API changes

3.12 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

3.13 PhotoScan version 1.3.2

- Added vertex_colors argument to Chunk.buildModel() method
- Added Shape.vertex_ids attribute

3.14 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(), Dense-Cloud.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

3.15 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() mehod
- 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 Reference-Format

- 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(), Network-Client.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

3.16 PhotoScan version 1.2.6

No Python API changes

3.17 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

3.18 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

3.19 PhotoScan version 1.2.3

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

3.20 PhotoScan version 1.2.2

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

3.21 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

3.22 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.addMenuItem() 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

3.23 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

3.24 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

3.25 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.addMenuItem() method

- Added absolute_paths parameter to Document.save() method

- Added fit_f, fit_cxcy, fit_k1k2k3 and fit_k4 parameters to Chunk.optimizePhotos() method

- 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

3.26 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.addMenuItem() method

- Added Model.closeHoles() and Model.fixTopology() methods

3.27 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)

3.28 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

3.29 PhotoScan version 0.8.4

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

3.30 PhotoScan version 0.8.3

Initial version of PhotoScan Python API

m