1 Overview 3
2 Application Modules 5
3 Python API Change Log 177
Python Module Index 201
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 functionality can be accessed from Python scripts:

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

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
 >>> chunk = doc.chunk
 >>> chunk.matchPhotos(downscale=1, generic_preselection=True, reference_preselection=False)
 >>> chunk.alignCameras()
 >>> chunk.buildDepthMaps(downscale=4, filter_mode=Metashape.AggressiveFiltering)
 >>> chunk.buildDenseCloud()
 >>> chunk.buildModel(surface_type=Metashape.Arbitrary, interpolation=Metashape.EnabledInterpolation)
 >>> chunk.buildUV(mapping_mode=Metashape.GenericMapping)
 >>> chunk.buildTexture(blending_mode=Metashape.MosaicBlending, texture_size=4096)
 >>> doc.save()
```

class Metashape.Antenna
   GPS antenna position relative to camera.

   copy()
       Return a copy of the object.

       Returns  A copy of the object.

       Return type Antenna

fixed
       Fix antenna flag.

       Type  bool

location
       Antenna coordinates.

       Type  Vector

location_acc
       Antenna location accuracy.

       Type  Vector
location_covariance
    Antenna location covariance.
    Type Matrix

location_ref
    Antenna location reference.
    Type Vector

rotation
    Antenna rotation angles.
    Type Vector

rotation_acc
    Antenna rotation accuracy.
    Type Vector

rotation_covariance
    Antenna rotation covariance.
    Type Matrix

rotation_ref
    Antenna rotation reference.
    Type Vector

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

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

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

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

class ConsolePane
    ConsolePane class provides access to the console pane

    clear()
        Clear console pane.

    contents
        Console pane contents.
        Type string
class Application.ModelView
  ModelView class provides access to the model view

  class DenseCloudViewMode
    Dense cloud view mode in [DenseViewColor, DenseViewClasses, DenseViewConfidence]

  class Application.ModelView.ModelViewMode
    Model view mode in [ModelViewShaded, ModelViewSolid, ModelViewWireframe, ModelViewConfidence, ModelViewTextured]

  class Application.ModelView.PointCloudViewMode
    Point cloud view mode in [PointCloudViewColor, PointCloudViewVariance]

  class Application.ModelView.TiledModelViewMode
    Tiled model view mode in [TiledModelViewTextured, TiledModelViewSolid, TiledModelViewWireframe]

Application.ModelView.captureView(width, height, transparent, hide_items)
  Capture image from model view.
  Parameters
    • width (int) – Image width.
    • height (int) – Image height.
    • transparent (bool) – Sets transparent background.
    • hide_items (bool) – Hides all items.
  Returns Captured image.
  Return type Image

Application.ModelView.dense_cloud_view_mode
  Dense cloud view mode.
  Type DenseCloudViewMode

Application.ModelView.model_view_mode
  Model view mode.
  Type ModelViewMode

Application.ModelView.point_cloud_view_mode
  Point cloud view mode.
  Type PointCloudViewMode

Application.ModelView.texture_view_mode
  Texture view mode.
  Type TextureViewMode

Application.ModelView.tiled_model_view_mode
  Tiled model view mode.
  Type TiledModelViewMode

Application.ModelView.view_mode
  View mode.
  Type DataSource

Application.ModelView.viewpoint
  Viewpoint in the model view.
  Type Viewpoint

class Application.OrthoView
  OrthoView class provides access to the ortho view
captureView([width], [height], [transparent], [hide_items])
Capture image from ortho view.

Parameters
• width (int) – Image width.
• height (int) – Image height.
• transparent (bool) – Sets transparent background.
• hide_items (bool) – Hides all items.

Returns Captured image.
Return type Image

view_mode
View mode.
Type DataSource

class 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)

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.console_pane
    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

```python
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

```python
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

```python
Application.messageBox(message)
```

*Display message box to the user.*

**Parameters**

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

```python
Application.model_view
```

*Model view.*

**Type** ModelView

```python
Application.ortho_view
```

*Ortho view.*

**Type** OrthoView

```python
Application.photos-pane
```

*Photos pane.*

**Type** PhotosPane

```python
Application.quit()
```

*Exit application.*

```python
Application.releaseFreeMemory()
```

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

```python
Application.removeMenuItem(label)
```

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

**Parameters**

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

```python
Application.settings
```

*Application settings.*

**Type** Settings

```python
Application.title
```

*Application name.*
Type string

Application.update()
Update user interface during long operations.

Application.version
Metashape version.

Type string
class Metashape.BBox
Axis aligned bounding box
copy()
Return a copy of the object.

Returns A copy of the object.

Return type BBox

max
Maximum bounding box extent.

Type Vector

min
Minimum bounding box extent.

Type Vector

size
Bounding box dimension.

Type int
class Metashape.BlendingMode
Blending mode in [AverageBlending, MosaicBlending, MinBlending, MaxBlending, DisabledBlending]
class Metashape.Calibration
Calibration object contains camera calibration information including image size, focal length, principal point coordinates and distortion coefficients.
b1
Affinity.

Type float

b2
Non-orthogonality.

Type float
copy()
Return a copy of the object.

Returns A copy of the object.

Return type Calibration
covariance_matrix
Covariance matrix.

Type Matrix
covariance_params
Covariance matrix parameters.
Type  list of string

cx
   Principal point X coordinate.
   Type  float

cy
   Principal point Y coordinate.
   Type  float

error (point, proj)
   Return projection error.
   Parameters
      • point (Vector) – Coordinates of the point to be projected.
      • proj (Vector) – Pixel coordinates of the point.
   Returns  2D projection error.
   Return type  Vector

f
   Focal length.
   Type  float

height
   Image height.
   Type  int

k1
   Radial distortion coefficient K1.
   Type  float

k2
   Radial distortion coefficient K2.
   Type  float

k3
   Radial distortion coefficient K3.
   Type  float

k4
   Radial distortion coefficient K4.
   Type  float

load (path, format=CalibrationFormatXML)
   Loads calibration from file.
   Parameters
      • path (string) – path to calibration file
      • format (CalibrationFormat) – Calibration format.
   Returns  success of operation
   Return type  bool
p1
Decentering distortion coefficient P1.
Type float

p2
Decentering distortion coefficient P2.
Type float

p3
Decentering distortion coefficient P3.
Type float

p4
Decentering distortion coefficient P4.
Type float

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

rpc
RPC model.
Type RPCModel

save(path, format=CalibrationFormatXML[, label [, pixel_size [, focal_length ]], cx = 0, cy = 0])
Saves calibration to file.
Parameters
• path(string) – path to calibration file
• format(CalibrationFormat) – Calibration format.
• label(string) – Calibration label used in Australis, CalibCam and CalCam formats.
• pixel_size(Vector) – Pixel size in mm used to convert normalized calibration coefficients to Australis and CalibCam coefficients.
• focal_length(float) – Focal length (Grid calibration format only).
• cx(float) – X principal point coordinate (Grid calibration format only).
• cy(float) – Y principal point coordinate (Grid calibration format only).
Returns success of operation
Return type bool

type
Camera model.
Type Sensor.Type

unproject(point)
Return direction corresponding to the image point.
Parameters point(Vector) – Pixel coordinates of the point.
**Returns** 3D vector in the camera coordinate system.

**Return type** `Vector`

**width**
Image width.

**Type** `int`

class `Metashape.CalibrationFormat`

class `Metashape.Camera`
Camera instance

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

The following example describes how to create multispectral camera layout:

```python
>>> 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
   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
   
   Returns
   
   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
   
   CameraGroup.label
   Camera group label.
   Type string

   CameraGroup.selected
   Current selection state.
class Metashape.CameraTrack
Camera track.

chunk
Chunk the camera track belongs to.

Type Chunk

duration
Animation duration.

Type float

field_of_view
Vertical field of view in degrees.

Type float

keyframes
Camera track keyframes.

Type list of Camera

label
Animation label.

Type string

load(path[, projection])
Load camera track from file.

Parameters

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

meta
Camera track meta data.

Type MetaData

save(path[, projection])
Save camera track to file.

Parameters

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

class Metashape.CamerasFormat
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:

```python
>>> import Metashape
>>> chunk = Metashape.app.document.chunk
>>> for frame in chunk.frames:
...    frame.matchPhotos(downscale=1)
>>> chunk.alignCameras()
```

addCamera([sensor])
Add new camera to the chunk.

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

Returns: Created camera.

Return type: Camera

addCameraGroup()
Add new camera group to the chunk.

Returns: Created camera group.

Return type: CameraGroup

addCameraTrack()
Add new camera track to the chunk.

Returns: Created camera track.

Return type: CameraTrack

addDenseCloud()
Add new dense cloud to the chunk.

Returns: Created dense cloud.

Return type: DenseCloud

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

Returns: Created depth maps set.

Return type: DepthMaps
addElevation()
Add new elevation model to the chunk.

Returns Created elevation model.
Return type Elevation

addFrame()
Add new frame to the chunk.

Returns Created frame.
Return type Frame

addFrames([chunk],[frames], copy_depth_maps=True, copy_dense_cloud=True,
copy_model=True, copy_tiled_model=True, copy_elevation=True,
copy_orthomosaic=True[, progress])
Add frames from specified chunk.

Parameters
• chunk (int) – Chunk to copy frames from.
• frames (list of int) – List of frame keys to copy.
• copy_depth_maps (bool) – Copy depth maps.
• copy_dense_cloud (bool) – Copy dense cloud.
• copy_model (bool) – Copy model.
• copy_tiled_model (bool) – Copy tiled model.
• copy_elevation (bool) – Copy DEM.
• copy_orthomosaic (bool) – Copy orthomosaic.
• progress (Callable[[float], None]) – Progress callback.

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

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

Returns Created marker.
Return type Marker

addMarkerGroup()
Add new marker group to the chunk.

Returns Created marker group.
Return type MarkerGroup

addModel()
Add new model to the chunk.

Returns Created model.
Return type Model

addOrthomosaic()
Add new orthomosaic to the chunk.
Returns | Created orthomosaic.
Return type | Orthomosaic

```python
def addPhotos(filenames, filegroups, layout=UndefinedLayout, group=None, strip_extensions=True, load_reference=True, load_xmp_calibration=True, load_xmp_orientation=True, load_xmp_accuracy=False, load_xmp_antenna=True, load_rpc_txt=False, progress=None):
    # Add a list of photos to the chunk.
    # Parameters
    # • filenames (list of string) – List of files to add.
    # • filegroups (list of int) – List of file groups.
    # • layout (ImageLayout) – Image layout.
    # • group (int) – Camera group key.
    # • strip_extensions (bool) – Strip file extensions from camera labels.
    # • load_reference (bool) – Load reference coordinates.
    # • load_xmp_calibration (bool) – Load calibration from XMP meta data.
    # • load_xmp_orientation (bool) – Load orientation from XMP meta data.
    # • load_xmp_accuracy (bool) – Load accuracy from XMP meta data.
    # • load_xmp_antenna (bool) – Load GPS/INS offset from XMP meta data.
    # • load_rpc_txt (bool) – Load satellite RPC data from auxiliary TXT files.
    # • progress (Callable[[float, None]) – Progress callback.
```

```python
def addScalebar(point1, point2):
    # Add new scale bar to the chunk.
    # Parameters
    # • point1 (Marker or Camera) – First endpoint.
    # • point2 – Second endpoint.
    # Returns | Created scale bar.
    # Return type | Scalebar
```

```python
def addScalebarGroup():
    # Add new scale bar group to the chunk.
    # Returns | Created scale bar group.
    # Return type | ScalebarGroup
```

```python
def addSensor():
    # Add new sensor to the chunk.
    # Returns | Created sensor.
    # Return type | Sensor
```

```python
def addTiledModel():
    # Add new tiled model to the chunk.
    # Returns | Created tiled model.
    # Return type | TiledModel
alignCameras([cameras], min_image=2, adaptive_fitting=False, reset_alignment=False, subdivide_task=True, progress)

Perform photo alignment for the chunk.

Parameters

- cameras (list of int) – List of cameras to align.
- min_image (int) – Minimum number of point projections.
- adaptive_fitting (bool) – Enable adaptive fitting of distortion coefficients.
- reset_alignment (bool) – Reset current alignment.
- subdivide_task (bool) – Enable fine-level task subdivision.
- progress (Callable[[float], None]) – Progress callback.

analyzePhotos([cameras], filter_mask=False, progress)

Estimate image quality.

Parameters

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

buildContours(source_data=ElevationData, interval=1, min_value=-1e+10, max_value=1e+10, prevent_intersections=False, progress)

Build contours for the chunk.

Parameters

- source_data (DataSource) – Source data for contour generation.
- interval (float) – Contour interval.
- min_value (float) – Minimum value of contour range.
- max_value (float) – Maximum value of contour range.
- prevent_intersections (bool) – Prevent contour intersections.
- progress (Callable[[float], None]) – Progress callback.

buildDem(source_data=DenseCloudData, interpolation=EnabledInterpolation, projection, region, classes, flip_x=False, flip_y=False, flip_z=False, resolution=0, subdivide_task=True, workitem_size_tiles=10, max_workgroup_size=100, progress)

Build elevation model for the chunk.

Parameters

- source_data (DataSource) – Selects between dense point cloud and sparse point cloud.
- interpolation (Interpolation) – Interpolation mode.
- projection (OrthoProjection) – Output projection.
- region (BBox) – Region to be processed.
- classes (list of int) – List of dense point classes to be used for surface extraction.
- flip_x (bool) – Flip X axis direction.
- flip_y (bool) – Flip Y axis direction.
• **flip_z** (*bool*) – Flip Z axis direction.
• **resolution** (*float*) – Output resolution in meters.
• **subdivide_task** (*bool*) – Enable fine-level task subdivision.
• **workitem_size_tiles** (*int*) – Number of tiles in a workitem.
• **max_workgroup_size** (*int*) – Maximum workgroup size.
• **progress** (*Callable[[float], None]*) – Progress callback.

```python
buildDenseCloud(point_colors=True, point_confidence=False, keep_depth=True, max_neighbors=100, subdivide_task=True, workitem_size_cameras=20, max_workgroup_size=100, progress)
```

Generate dense cloud for the chunk.

**Parameters**

• **point_colors** (*bool*) – Enable point colors calculation.
• **point_confidence** (*bool*) – Enable point confidence calculation.
• **keep_depth** (*bool*) – Enable store depth maps option.
• **max_neighbors** (*int*) – Maximum number of neighbor images to use for depth map filtering.
• **subdivide_task** (*bool*) – Enable fine-level task subdivision.
• **workitem_size_cameras** (*int*) – Number of cameras in a workitem.
• **max_workgroup_size** (*int*) – Maximum workgroup size.
• **progress** (*Callable[[float], None]*) – Progress callback.

```python
buildDepthMaps(downscale=4, filter_mode=MildFiltering, cameras, reuse_depth=False, max_neighbors=-1, subdivide_task=True, workitem_size_cameras=20, max_workgroup_size=100, progress)
```

Generate depth maps for the chunk.

**Parameters**

• **downscale** (*int*) – Depth map quality.
• **filter_mode** (*FilterMode*) – Depth map filtering mode.
• **cameras** (*list of int*) – List of cameras to process.
• **reuse_depth** (*bool*) – Enable reuse depth maps option.
• **max_neighbors** (*int*) – Maximum number of neighbor images to use for depth map generation.
• **subdivide_task** (*bool*) – Enable fine-level task subdivision.
• **workitem_size_cameras** (*int*) – Number of cameras in a workitem.
• **max_workgroup_size** (*int*) – Maximum workgroup size.
• **progress** (*Callable[[float], None]*) – Progress callback.
buildModel(surface_type=Arbitrary, interpolation=EnabledInterpolation, face_count=HighFaceCount, face_count_custom=200000, source_data=DenseCloudData, vertex_colors=True, vertex_confidence=True, volumetric_masks=False, keep_depth=True, trimming_radius=10, cameras, classes, subdivide_task=True, workitem_size_cameras=20, max_workgroup_size=100, progress)

Generate model for the chunk frame.

Parameters

- **surface_type (SurfaceType)** – Type of object to be reconstructed.
- **interpolation (Interpolation)** – Interpolation mode.
- **face_count (FaceCount)** – Target face count.
- **face_count_custom (int)** – Custom face count.
- **source_data (DataSource)** – Selects between dense point cloud, sparse point cloud and depth maps.
- **vertex_colors (bool)** – Enable vertex colors calculation.
- **vertex_confidence (bool)** – Enable vertex confidence calculation.
- **volumetric_masks (bool)** – Enable strict volumetric masking.
- **keep_depth (bool)** – Enable store depth maps option.
- **trimming_radius (int)** – Trimming radius (no trimming if zero).
- **cameras (list of int)** – List of cameras to process.
- **classes (list of int)** – List of dense point classes to be used for surface extraction.
- **subdivide_task (bool)** – Enable fine-level task subdivision.
- **workitem_size_cameras (int)** – Number of cameras in a workitem.
- **max_workgroup_size (int)** – Maximum workgroup size.
- **progress (Callable[[float], None])** – Progress callback.

buildOrthomosaic(surface_data=ModelData, blending_mode=MosaicBlending, fill_holes=True, cull_faces=False, refine_seamlines=False, projection=device, region, resolution=0, resolution_x=0, resolution_y=0, flip_x=False, flip_y=False, flip_z=False, subdivide_task=True, workitem_size_cameras=20, workitem_size_tiles=10, max_workgroup_size=100, progress)

Build orthomosaic for the chunk.

Parameters

- **surface_data (DataSource)** – Orthorectification surface.
- **blending_mode (BlendingMode)** – Orthophoto blending mode.
- **fill_holes (bool)** – Enable hole filling.
- **cull_faces (bool)** – Enable back-face culling.
- **refine_seamlines (bool)** – Refine seamlines based on image content.
- **projection (OrthoProjection)** – Output projection.
- **region (BBox)** – Region to be processed.
- **resolution (float)** – Pixel size in meters.
• **resolution_x (float)** – Pixel size in the X dimension in projected units.

• **resolution_y (float)** – Pixel size in the Y dimension in projected units.

• **flip_x (bool)** – Flip X axis direction.

• **flip_y (bool)** – Flip Y axis direction.

• **flip_z (bool)** – Flip Z axis direction.

• **subdivide_task (bool)** – Enable fine-level task subdivision.

• **workitem_size_cameras (int)** – Number of cameras in a workitem.

• **workitem_size_tiles (int)** – Number of tiles in a workitem.

• **max_workgroup_size (int)** – Maximum workgroup size.

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

```python
buildSeamlines (epsilon=1.5[, progress])
```

Generate shapes for orthomosaic seamlines.

**Parameters**

• **epsilon (float)** – Contour simplification threshold.

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

```python
buildTexture (blending_mode=MosaicBlending, texture_size=4096, fill_holes=True, ghosting_filter=True, cameras [, source_model ], texture_type=DiffuseMap[, source_model ], transfer_texture=True[, progress])
```

Generate texture for the chunk.

**Parameters**

• **blending_mode (BlendingMode)** – Texture blending mode.

• **texture_size (int)** – Texture page size.

• **fill_holes (bool)** – Enable hole filling.

• **ghosting_filter (bool)** – Enable ghosting filter.

• **cameras (list of int)** – A list of cameras to be used for texturing.

• **texture_type (Model.TextureType)** – Texture type.

• **source_model (int)** – Source model.

• **transfer_texture (bool)** – Transfer texture.

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

```python
buildTiledModel (pixel_size=0, tile_size=256, source_data=DenseCloudData, face_count=4000, ghosting_filter=False, transfer_texture=False, keep_depth=True[, classes ], subdivide_task=True, workitem_size_cameras=20, max_workgroup_size=100[, progress])
```

Build tiled model for the chunk.

**Parameters**

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

• **tile_size (int)** – Size of tiles in pixels.

• **source_data (DataSource)** – Selects between dense point cloud and mesh.

• **face_count (int)** – Number of faces per megapixel of texture resolution.
- **ghosting_filter**(bool) – Enable ghosting filter.
- **transfer_texture**(bool) – Transfer source model texture to tiled model.
- **keep_depth**(bool) – Enable store depth maps option.
- **classes**(list of int) – List of dense point classes to be used for surface extraction.
- **subdivide_task**(bool) – Enable fine-level task subdivision.
- **workitem_size_cameras**(int) – Number of cameras in a workitem.
- **max_workgroup_size**(int) – Maximum workgroup size.
- **progress**(Callable[[float], None]) – Progress callback.

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

**Parameters**
- **mapping_mode**(MappingMode) – Texture mapping mode.
- **page_count**(int) – Number of texture pages to generate.
- **adaptive_resolution**(bool) – Enable adaptive face detialization.
- **camera**(int) – Camera to be used for texturing in MappingCamera mode.
- **progress**(Callable[[float], None]) – Progress callback.

`calibrateColors(source_data=ModelData, white_balance=False[, cameras ][, progress])`
Perform radiometric calibration.

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

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

**Parameters**
- **use_reflectance_panels**(bool) – Use calibrated reflectance panels.
- **use_sun_sensor**(bool) – Apply irradiance sensor measurements.
- **progress**(Callable[[float], None]) – Progress callback.

**camera_crs**  
Coordinate system used for camera reference data.

*Type* CoordinateSystem

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

*Type* list of CameraGroup

**camera_location_accuracy**  
Expected accuracy of camera coordinates in meters.

*Type* Vector
camera_rotation_accuracy
Expected accuracy of camera orientation angles in degrees.
    Type Vector

camera_track
Camera track.
    Type CameraTrack

camera_tracks
List of camera tracks in the chunk.
    Type list of CameraTrack

cameras
List of Regular and Keyframe cameras in the chunk.
    Type list of Camera

cir_transform
CIR calibration matrix.
    Type CirTransform

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

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

Returns Copy of the chunk.
    Return type Chunk

crs
Coordinate system used for reference data.
    Type CoordinateSystem
decimateModel(face_count=200000, asset, progress)
Decimate the model to the specified face count.

Parameters
• face_count (int) – Target face count.
• asset (int) – Model to process.
• 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 (target_type=CircularTarget12bit, tolerance=50, filter_mask=False, inverted=False, noparity=False, maximum_residual=5, minimum_size=0, minimum_dist=5[, cameras ][, frames ][, progress ])
Create markers from coded targets.

Parameters
• target_type (TargetType) – Type of targets.
• tolerance (int) – Detector tolerance (0 - 100).
• filter_mask (bool) – Ignore masked image regions.
• inverted (bool) – Detect markers on black background.
• noparity (bool) – Disable parity checking.
• maximum_residual (float) – Maximum residual for non-coded targets in pixels.
• minimum_size (int) – Minimum target radius in pixels to be detected (CrossTarget type only).
• minimum_dist (int) – Minimum distance between targets in pixels (CrossTarget type only).
• cameras (list of int) – List of cameras to process.
• frames (list of int) – List of frames to process.
• progress (Callable[[float], None]) – Progress callback.
elevation
Default elevation model for the current frame.
Type Elevation
elevations
List of elevation models for the current frame.
Type list of Elevation
enabled
Enables/disables the chunk.
Type bool
euler_angles
Euler angles triplet used for rotation reference.
Type EulerAngles
exportCameras(path='', format=CamerasFormatXML[, crs ], save_points=True, save_markers=False, use_labels=False, chan_rotation_order=RotationOrderXYZ, binary=False, bundler_save_list=True, bundler_path_list='list.txt', bingo_save_image=True, bingo_save ITERA=True, bingo_save_geoin=True, bingo_save_gps=False, bingo_path ITERA='itera.dat', bingo_path_image='image.dat', bingo_path_geoin='geoin.dat', bingo_path_gps='gps-imu.dat'[, progress ])

Export point cloud and/or camera positions.

Parameters

- **path (string)** – Path to output file.
- **format (CamerasFormat)** – Export format.
- **crs (CoordinateSystem)** – Output coordinate system.
- **save_points (bool)** – Enables/disables export of automatic tie points.
- **save_markers (bool)** – Enables/disables export of manual matching points.
- **use_labels (bool)** – Enables/disables label based item identifiers.
- **chan_rotation_order (RotationOrder)** – Rotation order (CHAN format only).
- **binary (bool)** – Enables/disables binary encoding for selected format (if applicable).
- **bundler_save_list (bool)** – Enables/disables export of Bundler image list file.
- **bundler_path_list (string)** – Path to Bundler image list file.
- **bingo_save_image (bool)** – Enables/disables export of BINGO IMAGE COORDINATE file.
- **bingo_save ITERA (bool)** – Enables/disables export of BINGO ITERA file.
- **bingo_save_geoin (bool)** – Enables/disables export of BINGO GEO INPUT file.
- **bingo_save_gps (bool)** – Enables/disables export of BINGO GPS/IMU data.
- **bingo_path ITERA (string)** – Path to BINGO ITERA file.
- **bingo_path_image (string)** – Path to BINGO IMAGE COORDINATE file.
- **bingo_path_geoin (string)** – Path to BINGO GEO INPUT file.
- **bingo_path_gps (string)** – Path to BINGO GPS/IMU file.
- **progress (Callable[[float], None])** – Progress callback.

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

Export markers.

Parameters

- **path (string)** – Path to output file.
- **crs (CoordinateSystem)** – Output coordinate system.
- **binary (bool)** – Enables/disables binary encoding for selected format (if applicable).
- **progress (Callable[[float], None])** – Progress callback.
exportModel(path='', binary=True, precision=6, texture_format=ImageFormatJPEG, save_texture=True, save_uv=True, save_normals=True, save_cameras=True, save_markers=True, save_udim=False, strip_extensions=False, raster_transform=RasterTransformNone, colors_rgb_8bit=True, comment='', save_comment=True, format=ModelFormatNone, crs[], shift[], viewpoint[], progress))

Export generated model for the chunk.

Parameters

- **path** (string) – Path to output model.
- **binary** (bool) – Enables/disables binary encoding (if supported by format).
- **precision** (int) – Number of digits after the decimal point (for text formats).
- **texture_format** (ImageFormat) – Texture format.
- **save_texture** (bool) – Enables/disables texture export.
- **save_uv** (bool) – Enables/disables uv coordinates export.
- **save_normals** (bool) – Enables/disables export of vertex normals.
- **save_colors** (bool) – Enables/disables export of vertex colors.
- **save_cameras** (bool) – Enables/disables camera export.
- **save_markers** (bool) – Enables/disables marker export.
- **save_udim** (bool) – Enables/disables UDIM texture layout.
- **save_alpha** (bool) – Enables/disables alpha channel export.
- **strip_extensions** (bool) – Strips camera label extensions during export.
- **raster_transform** (RasterTransformType) – Raster band transformation.
- **colors_rgb_8bit** (bool) – Convert colors to 8 bit RGB.
- **comment** (string) – Optional comment (if supported by selected format).
- **save_comment** (bool) – Enables/disables comment export.
- **format** (ModelFormat) – Export format.
- **crs** (CoordinateSystem) – Output coordinate system.
- **shift** (Vector) – Optional shift to be applied to vertex coordinates.
- **viewpoint** (Viewpoint) – Default view.
- **progress** (Callable[[float], None]) – Progress callback.

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

Export orthophotos for the chunk.

Parameters

- **path** (string) – Path to output orthophoto.
- **cameras** (list of int) – List of cameras to process.
- **raster_transform** (RasterTransformType) – Raster band transformation.
- **projection** (OrthoProjection) – Output projection.
Export point cloud.

**Parameters**

- **path** *(string)* – Path to output file.
- **source_data** *(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).
- **save_normals** *(bool)* – Enables/disables export of point normals.
- **save_colors** *(bool)* – Enables/disables export of point colors.
- **save_classes** *(bool)* – Enables/disables export of point classes.
- **save_confidence** *(bool)* – Enables/disables export of point confidence.
- **raster_transform** *(RasterTransformType)* – Raster band transformation.
- **colors_rgb_8bit** *(bool)* – Convert colors to 8 bit RGB.
- **comment** *(string)* – Optional comment (if supported by selected format).
- **save_comment** *(bool)* – Enable comment export.
- **format** *(PointsFormat)* – Export format.
- **image_format** *(ImageFormat)* – Image data format.
- **crs** *(CoordinateSystem)* – Output coordinate system.
- **shift** *(Vector)* – Optional shift to be applied to point coordinates.
- **region** *(BBox)* – Region to be exported.
- **block_width** *(float)* – Block width in meters.
- **block_height** *(float)* – Block height in meters.
• **split_in_blocks** (*bool*) – Enable tiled export.

• **classes** (*list of int*) – List of dense point classes to be exported.

• **save_images** (*bool*) – Enable image export.

• **viewpoint** (*Viewpoint*) – Default view.

• **subdivide_task** (*bool*) – Enable fine-level task subdivision.

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

---

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

Export generated raster 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*) – Output projection.

• **region** (*BBox*) – Region to be exported.

• **resolution** (*float*) – Output resolution in meters.

• **resolution_x** (*float*) – Pixel size in the X dimension in projected units.

• **resolution_y** (*float*) – Pixel size in the Y dimension in projected units.

• **block_width** (*int*) – Raster block width in pixels.

• **block_height** (*int*) – Raster block height in pixels.

• **split_in_blocks** (*bool*) – Split raster in blocks.

• **width** (*int*) – Raster width.

• **height** (*int*) – Raster height.

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

• **nodata_value** (*float*) – No-data value (DEM export only).

• **save_kml** (*bool*) – Enable kml file generation.

• **save_world** (*bool*) – Enable world file generation.

• **save_scheme** (*bool*) – Enable tile scheme files generation.

• **save_alpha** (*bool*) – Enable alpha channel generation.

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

• **image_compression** (*ImageCompression*) – Image compression parameters.
• **network_links** *(bool)* – Enable network links generation for KMZ format.

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

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

• **white_background** *(bool)* – Enable white background.

• **title** *(string)* – Export title.

• **description** *(string)* – Export description.

• **source_data** *(DataSource)* – Selects between DEM and orthomosaic.

• **north_up** *(bool)* – Use north-up orientation for export.

• **tile_width** *(int)* – Tile width in pixels.

• **tile_height** *(int)* – Tile height in pixels.

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

**exportReference** *(path='', format=ReferenceFormatNone, items='', columns='', delimiter=' ', progress=None)*

Export reference data to the specified file.

**Parameters**

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

• **format** *(ReferenceFormat)* – Export format.

• **items** *(ReferenceItems)* – Items to export in CSV format.

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

• **delimiter** *(string)* – Column delimiter in csv format.

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

**exportReport** *(path='', title='', description='', page_numbers=True, user_settings=[], progress=None)*

Export processing report in PDF format.

**Parameters**

• **path** *(string)* – Path to output report.

• **title** *(string)* – Report title.

• **description** *(string)* – Report description.

• **page_numbers** *(bool)* – Enable page numbers.

• **user_settings** *(list of (string, string) tuples)* – A list of user defined settings to include on the Processing Parameters page.

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

```python
exportShapes(path='', save_points=False, save_polylines=False, save_polygons=False, groups=[], format=ShapesFormatNone, crs=[], shift, polygons_as_polylines=False, save_labels=True, save_attributes=True, progress)
```

Export shapes layer to file.

**Parameters**

- **path** *(string)* – Path to shape file.
- **save_points** *(bool)* – Export points.
- **save_polylines** *(bool)* – Export polylines.
- **save_polygons** *(bool)* – Export polygons.
- **groups** *(list of int)* – A list of shape groups to export.
- **format** *(ShapesFormat)* – Export format.
- **crs** *(CoordinateSystem)* – Output coordinate system.
- **shift** *(Vector)* – Optional shift to be applied to vertex coordinates.
- **polygons_as_polylines** *(bool)* – Save polygons as polylines.
- **save_labels** *(bool)* – Export labels.
- **save_attributes** *(bool)* – Export attributes.
- **progress** *(Callable[[float], None]*) – Progress callback.

**exportTiledModel**

```python
exportTiledModel(path='', format=TiledModelFormatNone, model_format=ModelFormatCOLLADA, texture_format=ImageFormatJPEG, raster_transform=RasterTransformNone, image_compression, crs, progress)
```

Export generated tiled model for the chunk.

**Parameters**

- **path** *(string)* – Path to output model.
- **format** *(TiledModelFormat)* – Export format.
- **model_format** *(ModelFormat)* – Model format for zip export.
- **texture_format** *(ImageFormat)* – Texture format.
- **raster_transform** *(RasterTransformType)* – Raster band transformation.
- **image_compression** *(ImageCompression)* – Image compression parameters.
- **crs** *(CoordinateSystem)* – Output coordinate system.
- **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.
**image_brightness**
Image brightness as percentage.
**Type** float

**image_contrast**
Image contrast as percentage.
**Type** float

**importCameras**
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).
- **load_image_list** (bool) – Enable Bundler image list import.
- **progress** (Callable[[float], None]) – Progress callback.

**importMarkers**
Import markers.

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

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

**importModel**
Import model from file.

**Parameters**
- **path** (string) – Path to model.
- **format** (ModelFormat) – Model format.
- **crs** (CoordinateSystem) – Model coordinate system.
- **shift** (Vector) – Optional shift to be applied to vertex coordinates.
- **progress** (Callable[[float], None]) – Progress callback.
importPoints(path='', format=PointsFormatNone, calculate_normals=True, crs=None, shift=None, progress=None)

Import point cloud from file.

Parameters

- **path** *(string)* - Path to point cloud.
- **format** *(PointsFormat)* - Point cloud format.
- **calculate_normals** *(bool)* - Calculate point normals.
- **crs** *(CoordinateSystem)* - Point cloud coordinate system.
- **shift** *(Vector)* - Optional shift to be applied to point coordinates.
- **progress** *(Callable[[float], None])* - Progress callback.

importRaster(path='', crs=None, raster_type=ElevationData, progress=None)

Import raster layer from file.

Parameters

- **path** *(string)* - Path to elevation model in GeoTIFF format.
- **crs** *(CoordinateSystem)* - Default coordinate system if not specified in GeoTIFF file.
- **raster_type** *(DataSource)* - Type of raster layer to import.
- **progress** *(Callable[[float], None])* - Progress callback.

importReference(path='', format=ReferenceFormatCSV, columns='', delimiter='', group_delimiters=False, skip_rows=0, items=None, crs=None, ignore_labels=False, create_markers=False, threshold=0.1, shutter_lag=0, progress=None)

Import reference data from the specified file.

Parameters

- **path** *(string)* - Path to the file with reference data.
- **format** *(ReferenceFormat)* - File format.
- **columns** *(string)* - Column order in csv format (n - label, o - enabled flag, x/y/z - coordinates, X/Y/Z - coordinate accuracy, a/b/c - rotation angles, A/B/C - rotation angle accuracy, [ ] - group of multiple values, | - column separator within group).
- **delimiter** *(string)* - Column delimiter in csv format.
- **group_delimiters** *(bool)* - Combine consecutive delimiters in csv format.
- **skip_rows** *(int)* - Number of rows to skip in (csv format only).
- **items** *(ReferenceItems)* - List of items to load reference for (csv format only).
- **crs** *(CoordinateSystem)* - Reference data coordinate system (csv format only).
- **ignore_labels** *(bool)* - Matches reference data based on coordinates alone (csv format only).
- **create_markers** *(bool)* - Create markers for missing entries (csv format only).
- **threshold** *(float)* - Error threshold in meters used when ignore_labels is set (csv format only).
- **shutter_lag** *(float)* - Shutter lag in seconds (APM format only).
- **progress** *(Callable[[float], None])* - Progress callback.
importShapes(path='', replace=False, boundary_type=NoBoundary, format=ShapesFormatNone, columns='nxyzd', delimiter=', ', group_delimiters=False, skip_rows=0, crs=[], progress)

Import shapes layer from file.

Parameters

- **path (string)** – Path to shape file.
- **replace (bool)** – Replace current shapes with new data.
- **boundary_type (Shape.BoundaryType)** – Boundary type to be applied to imported shapes.
- **format (ShapesFormat)** – Shapes format.
- **columns (string)** – Column order in csv format (n - label, x/y/z - coordinates, d - description, [] - group of multiple values, | - column separator within group).
- **delimiter (string)** – Column delimiter in csv format.
- **group_delimiters (bool)** – Combine consecutive delimiters in csv format.
- **skip_rows (int)** – Number of rows to skip in (csv format only).
- **crs (CoordinateSystem)** – Reference data coordinate system (csv format only).
- **progress (Callable[[float], None])** – Progress callback.

importTiledModel(path='', progress)

Import tiled model from file.

Parameters

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

key

Chunk identifier.

Type int

label

Chunk label.

Type string

loadReferenceExif(load_rotation=False, load_accuracy=False)

Import camera locations from EXIF meta data.

Parameters

- **load_rotation (bool)** – Load yaw, pitch and roll orientation angles.
- **load_accuracy (bool)** – Load camera location accuracy.

loadReflectancePanelCalibration(path[, cameras])

Load reflectance panel calibration from CSV file.

Parameters

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

locateReflectancePanels([progress])

Locate reflectance panels based on QR-codes.
Parameters `progress (Callable[[float], None])` – Progress callback.

`marker_crs`  
Coordinate system used for marker reference data.  
Type `CoordinateSystem`

`marker_groups`  
List of marker groups in the chunk.  
Type list of `MarkerGroup`

`marker_location_accuracy`  
Expected accuracy of marker coordinates in meters.  
Type `Vector`

`marker_projection_accuracy`  
Expected accuracy of marker projections in pixels.  
Type `float`

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

`masks`  
Image masks.  
Type `Masks`

`matchPhotos (downscale=1, generic_preselection=True, reference_preselection=True, reference_preselection_mode=ReferencePreselectionSource, filter_mask=False, mask_tiepoints=True, keypoint_limit=40000, tiepoint_limit=4000, keep_keypoints=False, pairs [], cameras [], guided_matching=False, reset_matches=False, subdivide_task=True, workitem_size_cameras=20, workitem_size_pairs=80, max_workgroup_size=100, progress )`  
Perform image matching for the chunk frame.

Parameters

- `downscale (int)` – Image alignment accuracy.
- `generic_preselection (bool)` – Enable generic preselection.
- `reference_preselection (bool)` – Enable reference preselection.
- `reference_preselection_mode (ReferencePreselectionMode)` – Reference preselection mode.
- `filter_mask (bool)` – Filter points by mask.
- `mask_tiepoints (bool)` – Apply mask filter to tie points.
- `keypoint_limit (int)` – Key point limit.
- `tiepoint_limit (int)` – Tie point limit.
- `keep_keypoints (bool)` – Store keypoints in the project.
- `pairs (list of (int, int) tuples)` – User defined list of camera pairs to match.
- `cameras (list of int)` – List of cameras to match.
- `guided_matching (bool)` – Enable guided image matching.
• **reset_matches**(bool) – Reset current matches.
• **subdivide_task**(bool) – Enable fine-level task subdivision.
• **workitem_size_cameras**(int) – Number of cameras in a workitem.
• **workitem_size_pairs**(int) – Number of image pairs in a workitem.
• **max_workgroup_size**(int) – Maximum workgroup size.
• **progress**(Callable[[float], None]) – Progress callback.

**meta**

Chunk meta data.

Type **MetaData**

**model**

Default model for the current frame.

Type **Model**

**models**

List of models for the current frame.

Type **list of** **Model**

**modified**

Modified flag.

Type **bool**

**optimizeCameras**(fit_f=True, fit_cx=True, fit_cy=True, fit_b1=False, fit_b2=False, fit_k1=True, fit_k2=True, fit_k3=True, fit_k4=False, fit_p1=True, fit_p2=True, fit_corrections=False, adaptive_fitting=False, tiepoint_covariance=False[, progress])

Perform optimization of point cloud / camera parameters.

**Parameters**

• **fit_f**(bool) – Enable optimization of focal length coefficient.
• **fit_cx**(bool) – Enable optimization of X principal point coordinates.
• **fit_cy**(bool) – Enable optimization of Y principal point coordinates.
• **fit_b1**(bool) – Enable optimization of aspect ratio.
• **fit_b2**(bool) – Enable optimization of skew coefficient.
• **fit_k1**(bool) – Enable optimization of k1 radial distortion coefficient.
• **fit_k2**(bool) – Enable optimization of k2 radial distortion coefficient.
• **fit_k3**(bool) – Enable optimization of k3 radial distortion coefficient.
• **fit_k4**(bool) – Enable optimization of k3 radial distortion coefficient.
• **fit_p1**(bool) – Enable optimization of p1 tangential distortion coefficient.
• **fit_p2**(bool) – Enable optimization of p2 tangential distortion coefficient.
• **fit_corrections**(bool) – Enable optimization of additional corrections.
• **adaptive_fitting**(bool) – Enable adaptive fitting of distortion coefficients.
• **tiepoint_covariance**(bool) – Estimate tie point covariance matrices.
• **progress**(Callable[[float], None]) – Progress callback.
orthomosaic
Default orthomosaic for the current frame.

Type Orthomosaic

orthomosaics
List of orthomosaics for the current frame.

Type list of Orthomosaic

point_cloud
Generated sparse point cloud.

Type PointCloud

primary_channel
Primary channel index (-1 for default).

Type int

raster_transform
Raster transform.

Type RasterTransform

reduceOverlap (overlap=30, max_cameras=0, use_selection=False[, progress ])
Disable redundant cameras.

Parameters
• overlap (int) – Overlap level.
• max_cameras (int) – Maximum cameras to use.
• use_selection (bool) – Focus on model selection.
• progress (Callable[[float], None]) – Progress callback.

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

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

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

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

region
Reconstruction volume selection.

Type Region
remove(items)
Remove items from the chunk.

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

removeLighting(color_mode=False, internal_blur=1.5, mesh_noise_suppression=1, ambient_occlusion_path='', ambient_occlusion_multiplier=1.5, progress)
Generate model for the chunk frame.

Parameters
- **color_mode** (bool) – Enable multi-color processing mode.
- **internal_blur** (float) – Internal blur. Should be in range [0, 4].
- **mesh_noise_suppression** (float) – Mesh normals noise suppression strength. Should be in range [0, 4].
- **ambient_occlusion_path** (string) – Path to ambient occlusion texture atlas. Can be empty.
- **ambient_occlusion_multiplier** (float) – Ambient occlusion multiplier. Should be in range [0.25, 4].
- **progress** (Callable[[float], None]) – Progress callback.

renderPreview(transform)

resetRegion()
Reset reconstruction volume selector to default position.

scalebar_accuracy
Expected scale bar accuracy in meters.

Type float

scalebar_groups
List of scale bar groups in the chunk.

Type list of ScalebarGroup

scalebars
List of scale bars in the chunk.

Type list of Scalebar

selected
Selects/deselects the chunk.

Type bool

sensors
List of sensors in the chunk.

Type list of Sensor

shapes
Shapes for the current frame.

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

Parameters

• strength (float) – Smoothing strength.
• apply_to_selection (bool) – Apply to selected faces.
• fix_borders (bool) – Fix borders.
• progress (Callable[[float], None]) – Progress callback.

sortCameras ()
Sorts cameras by their labels.

sortMarkers ()
Sorts markers by their labels.

sortScalebars ()
Sorts scalebars by their labels.

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

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

thumbnails
Image thumbnails.

Type Thumbnails

tiepoint_accuracy
Expected tie point accuracy in pixels.

Type float

tiled_model
Default tiled model for the current frame.

Type TiledModel

tiled_models
List of tiled models for the current frame.

Type list of TiledModel

trackMarkers (first_frame=0, last_frame=0[, progress])
Track marker projections through the frame sequence.

Parameters

• first_frame (int) – Starting frame index.
• last_frame (int) – Ending frame index.
• progress (Callable[[float], None]) – Progress callback.

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

Type ChunkTransform

triangulatePoints (max_error=10, min_image=2[, progress])
Rebuild point cloud for the chunk.

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

```python
updateTransform()
```
Update chunk transformation based on reference data.

```python
world_crs
```
Coordinate system used as world coordinate system.

```
Type CoordinateSystem
```

```python
class Metashape.ChunkTransform
```
Transformation between chunk and world coordinates systems.

```python
copy()
```
Return a copy of the object.

```
Returns A copy of the object.
```

```
Return type ChunkTransform
```

```python
matrix
```
Transformation matrix.

```
Type Matrix
```

```python
rotation
```
Rotation component.

```
Type Matrix
```

```python
scale
```
Scale component.

```
Type float
```

```python
translation
```
Translation component.

```
Type Vector
```

```python
class Metashape.CirTransform
```
CIR calibration matrix.

```python
calibrate()
```
Calibrate CIR matrix based on orthomosaic histogram.

```python
coeffs
```
Color matrix.

```
Type Matrix
```

```python
copy()
```
Return a copy of the object.

```
Returns A copy of the object.
```

```
Return type CirTransform
```

```python
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:

```python
>>> import Metashape

>>> chunk = Metashape.app.document.chunk

>>> chunk.crs = Metashape.CoordinateSystem("EPSG::32641")

>>> chunk.importReference("gcp.txt", Metashape.ReferenceFormatCSV)

>>> chunk.updateTransform()
```

addGeoid(path)

Register geoid model.

Parameters path (string) – Path to geoid file.

authority

Authority identifier of the coordinate system.

Type string

copy()

Return a copy of the object.

Returns A copy of the object.

Return type CoordinateSystem

geoccs

Base geocentric coordinate system.

Type CoordinateSystem

geogcs

Base geographic coordinate system.

Type CoordinateSystem

good

Fixed geoid height to be used instead of interpolated values.

Type float

init(crs)

Initialize projection based on specified WKT definition or authority identifier.

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

listBuiltInCRS()

Returns a list of builtin coordinate systems.

localframe(point)

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

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

Returns Transformation from geocentric coordinates to local coordinates.

Return type Matrix

name

Name of the coordinate system.

Type string
proj4
Coordinate system definition in PROJ.4 format.

Type string

project (point)
Projects point from geocentric coordinates to projected geographic coordinate system.

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

Returns 3D point in projected coordinates.

Return type Vector

towgs84
TOWGS84 transformation parameters (dx, dy, dz, rx, ry, rz, scale).

Type list of float

transform (point, source, target)
Transform point coordinates between coordinate systems.

Parameters

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

Returns Transformed point coordinates.

Return type Vector

transformationMatrix (point, source, target)
Local approximation of coordinate transformation from source to target coordinate system at the given point.

Parameters

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

Returns 4x4 transformation matrix.

Return type Matrix

unproject (point)
Unprojects point from projected coordinates to geocentric coordinates.

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

Returns 3D point in geocentric coordinates.

Return type Vector

wkt
Coordinate system definition in WKT format.

Type string

wkt2
Coordinate system definition in WKT format, version 2.

Type string
class Metashape.DataSource
Data source in [PointCloudData, DenseCloudData, DepthMapsData, ModelData, TiledModelData, ElevationData, OrthomosaicData, ImagesData]

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

class Metashape.DenseCloud
Dense point cloud data.

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

Parameters

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

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

Parameters

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

bands
List of color bands.

  Type  list of string

classifyGroundPoints(max_angle=15.0, max_distance=1.0, cell_size=50.0[, 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 (list of PointClass) – Target point classes for classification.
  • confidence (float) – Required confidence level from 0.0 to 1.0.
  • progress (Callable[[float], None]) – Progress callback.
clear()
Clears dense cloud data.

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

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

copy()
Create a copy of the dense cloud.

Returns Copy of the dense cloud.

Return type DenseCloud
cropSelectedPoints([point_classes][, progress])
Crop selected points.

Parameters

• point_classes(PointClass or list of PointClass) – Classes of points to be removed.

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

crs
Reference coordinate system.

Type CoordinateSystem or None
data_type
Data type used to store color values.

Type DataType
key
Dense cloud identifier.

Type int
label
Dense cloud label.

Type string
meta
Dense cloud meta data.

Type MetaData
modified
Modified flag.

Type bool
pickPoint(origin, target, endpoints=1)
Returns ray intersection with the point cloud (point on the ray nearest to some point).

Parameters

• origin(Vector) – Ray origin.

• target(Vector) – Point on the ray.

• endpoints(int) – Number of endpoints to check for (0 - line, 1 - ray, 2 - segment).

Returns Coordinates of the intersection point.
Return type **Vector**

**point_count**
Number of points in dense cloud.

**Type** int

**removePoints** (*point_classes*, *progress*)
Remove points.

**Parameters**

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

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

**Parameters**

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

**renderDepth** (*transform*, *calibration*, *point_size=1*, *resolution=1*, *cull_points=False*, *add_alpha=True*)
Render dense cloud depth image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point_size** (*int*) – Point size.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull_points** (*bool*) – Enable normal based culling.
- **add_alpha** (*bool*) – Generate image with alpha channel.

**Returns** Rendered image.

**Return type** **Image**

**renderImage** (*transform*, *calibration*, *point_size=1*, *resolution=1*, *cull_points=False*, *add_alpha=True*, *raster_transform=RasterTransformNone*)
Render dense cloud image for specified viewpoint.

**Parameters**

- **transform** (*Matrix*) – Camera location.
- **calibration** (*Calibration*) – Camera calibration.
- **point_size** (*int*) – Point size.
- **resolution** (*float*) – Level of detail resolution in screen pixels.
- **cull_points** (*bool*) – Enable normal based culling.
- **add_alpha** (*bool*) – Generate image with alpha channel.
- **raster_transform** (*RasterTransformType*) – Raster band transformation.
Returns  Rendered image.
Return type  Image

`renderMask(transform, calibration, point_size=1, resolution=1, cull_points=False)`
Render dense cloud mask image for specified viewpoint.

Parameters

• `transform (Matrix)` – Camera location.
• `calibration (Calibration)` – Camera calibration.
• `point_size (int)` – Point size.
• `resolution (float)` – Level of detail resolution in screen pixels.
• `cull_points (bool)` – Enable normal based culling.

Returns  Rendered image.
Return type  Image

`renderNormalMap(transform, calibration, point_size=1, resolution=1, cull_points=False, add_alpha=True)`
Render image with dense cloud normals for specified viewpoint.

Parameters

• `transform (Matrix)` – Camera location.
• `calibration (Calibration)` – Camera calibration.
• `point_size (int)` – Point size.
• `resolution (float)` – Level of detail resolution in screen pixels.
• `cull_points (bool)` – Enable normal based culling.
• `add_alpha (bool)` – Generate image with alpha channel.

Returns  Rendered image.
Return type  Image

`resetFilters()`
Reset filters.

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

Parameters

• `point_classes (PointClass or list of PointClass)` – Classes of points to be restored.
• `progress (Callable[[float], None])` – Progress callback.

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

Parameters

• `cameras (list of Camera)` – A list of cameras to use for selection.
• `softness (float)` – Mask edge softness.
• `progress (Callable[[float], None])` – Progress callback.
selectPointsByColor\(\text{color, tolerance=10, channels='RGB', progress}\)  
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.

setClassesFilter\(\text{point_classes}\)  
Set filter by point classes.

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

setConfidenceFilter\(\text{min_confidence, max_confidence}\)  
Set filter by confidence.

**Parameters**

- **min_confidence** (int) – Minimum confidence value.
- **max_confidence** (int) – Maximum confidence value.

setSelectionFilter()  
Set filter by selection.

transform  
4x4 dense cloud transformation matrix.

**Type** Matrix

updateStatistics\(\text{progress}\)  
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

calibration\(\text{level=0}\)  
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:

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

addChunk()

Add new chunk to the document.

Returns Created chunk.

Return type Chunk

alignChunks ([chunks][, reference ], method=0, fit_scale=True, downscale=1, generic_preselection=False, filter_mask=False, mask_tiepoints=False, keypoint_limit=40000[, markers ][, progress ])

Align specified set of chunks.

Parameters

- chunks (list of int) – List of chunks to be aligned.
- reference (int) – Chunk to be used as a reference.
- method (int) – Alignment method.
- fit_scale (bool) – Fit chunk scale during alignment.
- downscale (int) – Alignment accuracy.
- generic_preselection (bool) – Enables image pair preselection.
- filter_mask (bool) – Filter points by mask.
- mask_tiepoints (bool) – Apply mask filter to tie points.
- keypoint_limit (int) – Maximum number of points for each photo.
- markers (list of int) – List of markers to be used for marker based alignment.
- progress (Callable[[float], None]) – Progress callback.

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

Append the specified Document object to the current document.

Parameters

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

chunk

Active chunk.

Type Chunk
chunks
   List of chunks in the document.
   Type Chunk

clear()
   Clear the contents of the Document object.

copy()
   Return a copy of the document.
   Returns A copy of the document.
   Return type Document

findChunk(key)
   Find chunk by its key.
   Returns Found chunk.
   Return type Chunk

mergeChunks(merge_markers=False, merge_tiepoints=False, merge_depth_maps=False,
          merge_dense_clouds=True, merge_models=False, merge_elevations=False,
          merge_orthomosaics=False, chunks=[], progress)
   Merge specified set of chunks.
   Parameters
   • merge_markers (bool) – Merge markers.
   • merge_tiepoints (bool) – Merge tie points.
   • merge_depth_maps (bool) – Merge depth maps.
   • merge_dense_clouds (bool) – Merge dense clouds.
   • merge_models (bool) – Merge models.
   • merge_elevations (bool) – Merge DEMs.
   • merge_orthomosaics (bool) – Merge orthomosaics.
   • chunks (list of int) – List of chunks to process.
   • progress (Callable[[float], None]) – Progress callback.

meta
   Document meta data.
   Type Metadata

modified
   Modified flag.
   Type bool

open(path, read_only=False, ignore_lock=False)
   Load document from the specified file.
   Parameters
   • path (string) – Path to the file.
   • read_only (bool) – Open document in read-only mode.
   • ignore_lock (bool) – Ignore lock state for project modifications.
path
Path to the document file.

Type string

read_only
Read only status.

Type bool

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

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

save([path][, chunks], 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
<table>
<thead>
<tr>
<th>key</th>
<th>Elevation model identifier.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>label</td>
<td>Elevation model label.</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>left</td>
<td>X coordinate of the left side.</td>
</tr>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>max</td>
<td>Maximum elevation value.</td>
</tr>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>meta</td>
<td>Elevation model meta data.</td>
</tr>
<tr>
<td>Type</td>
<td>MetaData</td>
</tr>
<tr>
<td>min</td>
<td>Minimum elevation value.</td>
</tr>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>modified</td>
<td>Modified flag.</td>
</tr>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>palette</td>
<td>Color palette.</td>
</tr>
<tr>
<td>Type</td>
<td>dict</td>
</tr>
<tr>
<td>projection</td>
<td>Projection of elevation model.</td>
</tr>
<tr>
<td>Type</td>
<td>OrthoProjection</td>
</tr>
<tr>
<td>resolution</td>
<td>DEM resolution in meters.</td>
</tr>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>right</td>
<td>X coordinate of the right side.</td>
</tr>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>top</td>
<td>Y coordinate of the top side.</td>
</tr>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>width</td>
<td>Elevation model width.</td>
</tr>
<tr>
<td>Type</td>
<td>int</td>
</tr>
</tbody>
</table>

**class Metashape.EulerAngles**

Euler angles in [EulerAnglesYPR, EulerAnglesOPK, EulerAnglesPOK, EulerAnglesANK]
class Metashape.FaceCount
    Face count in [LowFaceCount, MediumFaceCount, HighFaceCount, CustomFaceCount]

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

class Metashape.Image(width, height, channels, datatype='U8')
    n-channel image

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

    channels
    Channel mapping for the image.
      Type string

cn
    Number of color channels.
      Type int

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

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

    Returns Converted image.
      Return type Image

copy ()
    Return a copy of the image.

    Returns copy of the image
      Return type Image

data_type
    Data type used to store pixel values.
      Type string

    fromstring (data, width, height, channels, datatype='U8')
    Create image from byte array.

    Parameters
    • data (string) – raw image data
    • width (int) – image width
    • height (int) – image height
    • channels (string) – color channel layout, e.g. ‘RGB’, ‘RGBA’, etc.
    • datatype (string) – pixel data type in ['U8', 'U16', 'U32', 'F16', 'F32', 'F64']
Returns
Created image.

Return type
Image

gaussianBlur(radius)
Smooth image with a gaussian filter.

Parameters
radius(float) – smoothing radius.

Returns
Smoothed image.

Return type
Image

height
Image height.

Type
int

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

Parameters

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

Returns
Loaded image.

Return type
Image

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

Parameters

• width(int) – new image width
• height(int) – new image height

Returns
resized image

Return type
Image

save(path[, compression])
Save image to the file.

Parameters

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

tostring()
Convert image to byte array.

Returns
Raw image data.
Return type string

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

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

**Returns** undistorted image

Return type **Image**

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

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

**Returns** Image with added noise.

**Return type** **Image**

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

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

**Returns** warped image

**Return type** **Image**

width
Image width.

**Type** int

**class** Metashape.ImageCompression
Image compression parameters

**class** TiffCompression
Tiff compression in [TiffCompressionNone, TiffCompressionLZW, TiffCompressionJPEG, TiffCompressionPackbits, TiffCompressionDeflate]

ImageCompression.copy()
Return a copy of the object.

**Returns** A copy of the object.

**Return type** **Viewpoint**

**ImageCompression.jpeg_quality**
JPEG quality.

**Type** int
ImageCompression.tiff_big
   Enable BigTIFF compression for TIFF files.
   Type bool

ImageCompression.tiff_compression
   Tiff compression.
   Type int

ImageCompression.tiff_overviews
   Enable image pyramid generation for TIFF files.
   Type bool

ImageCompression.tiff_tiled
   Export tiled TIFF.
   Type bool

class Metashape.ImageFormat

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**


**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.enabled**

Enables/disables the marker.

- Type `bool`
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.position_covariance
    Marker position covariance.
    Type Matrix

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

**copy()**

Return a copy of the object.

**Returns** A copy of the object.

**Return type** *MetaData*

**items()**

List of items.

**keys()**

List of item keys.

**values()**

List of item values.

**class Metashape.Model**

Triangular mesh model instance

**class Face**

Triangular face of the model

**hidden**

Face visibility flag.

**Type** bool

**selected**

Face selection flag.

**Type** bool

**tex_index**

Texture page index.

**Type** int

**tex_vertices**

Texture vertex indices.

**Type** tuple of 3 int

**vertices**

Vertex indices.

**Type** tuple of 3 int

**class Metashape.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.Texture*

Model texture.

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

Return texture image.

**Parameters**

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

**Returns**

Texture image.

**Return type**

*Image*

**label**

Animation label.

**Type**

*string*

**meta**

Camera track meta data.

**Type**

*MetaData*

**model**

Model the texture belongs to.

**Type**

*Model*

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

Initialize texture from image data.

**Parameters**

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

**type**

Texture type.

**Type**

*Model.TextureType*

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

Add new texture to the model.

**Parameters**

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

**Returns**

Created texture.
Return type `Model.Texture`

`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.getActiveTexture(type=Model.DiffuseMap)`
Return active texture.

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

Returns Texture image.

Return type `Image`

`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, endpoints=1)*
   Return ray intersection with mesh.

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

   Returns  Coordinates of the intersection point.
      Return type **Vector**

Model.remove *(items)*
   Remove textures from the model.

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

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, cull_faces=True, add_alpha=True)*
   Render model depth image for specified viewpoint.

   Parameters
      • **transform** *(Matrix)* – Camera location.
      • **calibration** *(Calibration)* – Camera calibration.
      • **cull_faces** *(bool)* – Enable back-face culling.
      • **add_alpha** *(bool)* – Generate image with alpha channel.

   Returns  Rendered image.
      Return type **Image**

Model.renderImage *(transform, calibration, cull_faces=True, add_alpha=True, raster_transform=RasterTransformNone)*
   Render model image for specified viewpoint.

   Parameters
      • **transform** *(Matrix)* – Camera location.
      • **calibration** *(Calibration)* – Camera calibration.
      • **cull_faces** *(bool)* – Enable back-face culling.
• **add_alpha** *(bool)* – Generate image with alpha channel.

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

  Returns Rendered image.

  Return type *Image*

**Model.renderMask**(transform, calibration, cull_faces=True)

Render model mask image for specified viewpoint.

  Parameters

  • **transform** *(Matrix)* – Camera location.
  
  • **calibration** *(Calibration)* – Camera calibration.
  
  • **cull_faces** *(bool)* – Enable back-face culling.

  Returns Rendered image.

  Return type *Image*

**Model.renderNormalMap**(transform, calibration, cull_faces=True, add_alpha=True)

Render image with model normals for specified viewpoint.

  Parameters

  • **transform** *(Matrix)* – Camera location.
  
  • **calibration** *(Calibration)* – Camera calibration.
  
  • **cull_faces** *(bool)* – Enable back-face culling.
  
  • **add_alpha** *(bool)* – Generate image with alpha channel.

  Returns Rendered image.

  Return type *Image*

**Model.saveTexture**(path)

Save texture to the specified file.

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

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

Set active texture.

  Parameters

  • **texture** *(Model.Texture)* – Texture to set.
  
  • **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.textures
List of model textures.

Type list of Model.Texture

Model.transform(transform)
Transform vertex coordinates.

Parameters transform(Matrix) – 4x4 transformation matrix.

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.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:

```python
>>> 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

def connect(host, port=5840)
    Connect to the server.

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

def 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

def disconnect()
    Disconnect from the server.

def dumpBatches(batch_ids)
    Dump current state of batches.

    Parameters
    - batch_ids (list of int) – List of batch ids to dump.

    Returns  Batches data.
Return type  string

def findBatch(path)
    Get batch id based on project path.

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

    Returns  Batch id.
Return type  int

def loadBatches(data)
    Load batches from dump.

    Parameters
    - data (string) – Batches data.


def nodeList(revision=0)
    Get list of nodes.

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

    Returns  List of nodes.
Return type  dict

def 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).

setMasterServer ([host])
  Set or reset master server.
  Parameters  host (string) – Master server hostname.

setNodeCPUEnable (node_id, cpu_enable)
  Set node CPU enable flag.
  Parameters
    • node_id (int) – Node id.
    • cpu_enable (bool) – CPU enable flag.
**setNodeCapability** *(node_id, capability)*

Set node capability.

**Parameters**

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

**setNodeGPUMask** *(node_id, gpu_mask)*

Set node GPU mask.

**Parameters**

- `node_id` *(int)* – Node id.
- `gpu_mask` *(int)* – GPU device mask.

**setNodePriority** *(node_id, priority)*

Set node priority.

**Parameters**

- `node_id` *(int)* – Node id.

**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:

```python
>>> 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.copy()
        Return a copy of the object.

        Returns  A copy of the object.

        Return type  OrthoProjection

    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 (OrthoProjection) – Source coordinate system.
            •  target (OrthoProjection) – 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]
```python
>>> 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.
Orthomosaic

**key**
Orthomosaic identifier.

**label**
Orthomosaic label.

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

**meta**
Orthomosaic meta data.

**modified**
Modified flag.

**patches**
Orthomosaic patches.

**projection**
Orthomosaic projection.

**removeOrthophotos**
Remove orthorectified images from orthomosaic.

**reset**
Reset all edits to orthomosaic.

**resolution**
Orthomosaic resolution in meters.

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

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

**update**
Apply edits to orthomosaic.

**width**
Orthomosaic width.

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

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:

```python
>>> 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.cleanup([progress])
   Remove points with insufficient number of projections.
   Parameters  progress (Callable[[float, None]] – Progress callback.

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, endpoints=1)
    Returns ray intersection with the point cloud (point on the ray nearest to some point).

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

    Returns Coordinates of the intersection point.

    Return type Vector

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.renderDepth(transform, calibration, point_size=1, cull_points=False, add_alpha=True)
    Render point cloud depth image for specified viewpoint.

    Parameters
    • transform (Matrix) – Camera location.
    • calibration (Calibration) – Camera calibration.
    • point_size (int) – Point size.
    • cull_points (bool) – Enable normal based culling.
    • add_alpha (bool) – Generate image with alpha channel.

    Returns Rendered image.

    Return type Image

PointCloud.renderImage(transform, calibration, point_size=1, cull_points=False, add_alpha=True, raster_transform=RasterTransformNone)
    Render point cloud image for specified viewpoint.

    Parameters
    • transform (Matrix) – Camera location.
    • calibration (Calibration) – Camera calibration.
    • point_size (int) – Point size.
    • cull_points (bool) – Enable normal based culling.
    • add_alpha (bool) – Generate image with alpha channel.
- `raster_transform(RasterTransformType)` – Raster band transformation.
  Returns  Rendered image.
  Return type  `Image`

`PointCloud.renderMask(transform, calibration, point_size=1, cull_points=False)`
Render point cloud mask image for specified viewpoint.

Parameters
- `transform(Matrix)` – Camera location.
- `calibration(Calibration)` – Camera calibration.
- `point_size(int)` – Point size.
- `cull_points(bool)` – Enable normal based culling.

Returns  Rendered image.
Return type  `Image`

`PointCloud.renderNormalMap(transform, calibration, point_size=1, cull_points=False, add_alpha=True)`
Render image with point cloud normals for specified viewpoint.

Parameters
- `transform(Matrix)` – Camera location.
- `calibration(Calibration)` – Camera calibration.
- `point_size(int)` – Point size.
- `cull_points(bool)` – Enable normal based culling.
- `add_alpha(bool)` – Generate image with alpha channel.

Returns  Rendered image.
Return type  `Image`

`PointCloud.tracks`
List of tracks.

Type  `PointCloud.Tracks`

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

### class Metashape.Preselection
Image pair preselection in [NoPreselection, GenericPreselection, ReferencePreselection]

### class Metashape.RPCModel
Rational polynomial model.

#### copy()
Return a copy of the object.

Returns  A copy of the object.

Return type  `RPCModel`
error \( (\text{point, proj}) \)
Returns projection error.

Parameters

- \textbf{point} \( (\text{Vector}) \) – Coordinates of the point to be projected.
- \textbf{proj} \( (\text{Vector}) \) – Pixel coordinates of the point.

Returns 2D projection error.

Return type \text{Vector}

\textit{image_offset}
Image coordinate offset.

Type \text{Vector}

\textit{image_scale}
Image coordinate scale.

Type \text{Vector}

\textit{line_den_coeff}
Line denominator.

Type \text{Vector}

\textit{line_num_coeff}
Line numerator.

Type \text{Vector}

load \( (\text{path}) \)
Load RPC model from file.

Parameters \textbf{path} \( (\text{string}) \) – path to RPC model file

\textit{object_offset}
Object coordinate offset.

Type \text{Vector}

\textit{object_scale}
Object coordinate scale.

Type \text{Vector}

\textit{project} \( (\text{point}) \)
Returns projected pixel coordinates of the point.

Parameters \textbf{point} \( (\text{Vector}) \) – Coordinates of the point to be projected.

Returns 2D projected point coordinates.

Return type \text{Vector}

\textit{samp_den_coeff}
Sample denominator.

Type \text{Vector}

\textit{samp_num_coeff}
Sample numerator.

Type \text{Vector}
**save** *(path)*
Save RPC model to file.

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

**unproject** *(point)*
Returns direction corresponding to the image point.

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

**Returns**
3D vector in the camera coordinate system.

**Return type**
Vector

class **Metashape.RasterFormat**
Raster format in [RasterFormatNone, RasterFormatTiles, RasterFormatKMZ, RasterFormatXYZ, RasterFormatMBTiles, RasterFormatWW, RasterFormatTMS]

class **Metashape.RasterTransform**
Raster transform definition.

**calibrateRange** ()
Auto detect range based on orthomosaic histogram.

**copy** ()
Return a copy of the object.

**Returns**
A copy of the object.

**Return type**
RasterTransform

**enabled**
Enable flag.

**Type**
bool

**false_color**
False color channels.

**Type**
list

**formula**
Raster calculator expression.

**Type**
string

**interpolation**
Interpolation enable flag.

**Type**
bool

**palette**
Color palette.

**Type**
dict

**range**
Palette mapping range.

**Type**
tuple

**reset** ()
Reset raster transform.

class **Metashape.RasterTransformType**
Raster transformation type in [RasterTransformNone, RasterTransformValue, RasterTransformPalette]
class Metashape.ReferenceFormat

class Metashape.ReferenceItems
    Reference items in [ReferenceItemsCameras, ReferenceItemsMarkers, ReferenceItemsScalebars]

class Metashape.ReferencePreselectionMode
    Reference preselection mode in [ReferencePreselectionSource, ReferencePreselectionEstimated, ReferencePreselectionSequential]

class Metashape.Region
    Region parameters

    center
        Region center coordinates.
        Type Vector

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

    rot
        Region rotation matrix.
        Type Matrix

    size
        Region size.
        Type Vector

class Metashape.RotationOrder
    Rotation order in [RotationOrderXYZ, RotationOrderXZY, RotationOrderYXZ, RotationOrderYZX, RotationOrderZXY, RotationOrderZYX]

class Metashape.Scalebar
    Scale bar instance

class Reference
    Scale bar reference data

    accuracy
        Scale bar length accuracy.
        Type float

    distance
        Scale bar length.
        Type float

    enabled
        Enabled flag.
        Type bool

    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

disabled
   Location enabled flag.
   Type bool

location
   Sensor coordinates.
   Type Vector

location_accuracy
   Sensor location accuracy.
   Type Vector

location_enabled
   Location enabled flag.
   Type bool

rotation
   Sensor rotation angles.
   Type Vector

rotation_accuracy
   Sensor rotation accuracy.
   Type Vector

rotation_enabled
   Rotation enabled flag.
   Type bool

class Sensor:
   Sensor type in [Frame, Fisheye, Spherical, RPC]

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 image-variant 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 ShapeGroup 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, ShapesFormatGeoJSON, ShapesFormatCSV]

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

copy()
Return a copy of the object.

Returns A copy of the object.

Return type Shutter

rotation
Rotation matrix of the rolling shutter model.

Type Matrix

translación
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

cord
Target location.

Type Vector

copy()
Return a copy of the object.

Returns A copy of the object.

Return type Target

radius
Target radius.

Type float
class Metashape.TargetType
Target type in [CircularTarget12bit, CircularTarget14bit, CircularTarget16bit, CircularTarget20bit, CircularTarget, CrossTarget]

class Metashape.Tasks
Task classes.

class AddFrames
Task class containing processing parameters.

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

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

    chunk
    Chunk to copy frames from.
    Type int

    copy_dense_cloud
    Copy dense cloud.
    Type bool

    copy_depth_maps
    Copy depth maps.
    Type bool

    copy_elevation
    Copy DEM.
    Type bool

    copy_model
    Copy model.
    Type bool

    copy_orthomosaic
    Copy orthomosaic.
    Type bool

    copy_tiled_model
    Copy tiled model.
    Type bool

    decode(dict)
    Initialize task parameters with a dictionary.

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

    encode()
    Create a dictionary with task parameters.

    encodeJSON()
    Create a JSON string with task parameters.

    frames
    List of frame keys to copy.
    Type list of int
name
   Task name.
   Type string

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

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)
   Initialize task parameters with a dictionary.

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

encode()
   Create a dictionary with task parameters.

encodeJSON()
   Create a JSON string with task parameters.

filegroups
   List of file groups.
   Type list of int

filenames
   List of files to add.
   Type list of string

group
   Camera group key.
   Type int

layout
   Image layout.
   Type ImageLayout

load_reference
   Load reference coordinates.
   Type bool

load_rpc_txt
   Load satellite RPC data from auxiliary TXT files.
   Type bool
load_xmp_accuracy
  Load accuracy from XMP meta data.
  Type  bool

load_xmp_antenna
  Load GPS/INS offset from XMP meta data.
  Type  bool

load_xmp_calibration
  Load calibration from XMP meta data.
  Type  bool

load_xmp_orientation
  Load orientation from XMP meta data.
  Type  bool

name
  Task name.
  Type  string

strip_extensions
  Strip file extensions from camera labels.
  Type  bool

supports_gpu
  GPU support flag.
  Type  bool

target
  Task target.
  Type  Tasks.TargetType

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)
  Initialize task parameters with a dictionary.

decodeJSON (json)
  Initialize task parameters from a JSON string.
**encode** ()
Create a dictionary with task parameters.

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

**min_image**
Minimum number of point projections.
Type int

**name**
Task name.
Type string

**reset_alignment**
Reset current alignment.
Type bool

**subdivide_task**
Enable fine-level task subdivision.
Type bool

**supports_gpu**
GPU support flag.
Type bool

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

**workitem_count**
Work item count.
Type int

**class Tasks.AlignChunks**
Task class containing processing parameters.

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

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

**chunks**
List of chunks to be aligned.
Type list of int

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

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

**downscale**
Alignment accuracy.
Type int

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

filter_mask  
Filter points by mask.  
Type bool

fit_scale  
Fit chunk scale during alignment.  
Type bool

generic_preselection  
Enables image pair preselection.  
Type bool

keypoint_limit  
Maximum number of points for each photo.  
Type int

markers  
List of markers to be used for marker based alignment.  
Type list of int

mask_tiepoints  
Apply mask filter to tie points.  
Type bool

method  
Alignment method.  
Type int

name  
Task name.  
Type string

reference  
Chunk to be used as a reference.  
Type int

supports_gpu  
GPU support flag.  
Type bool

target  
Task target.  
Type Tasks.TargetType

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*)
Initialize task parameters with a dictionary.

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

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

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

**filter_mask**
Constrain analyzed image region by mask.

Type  bool

**name**
Task name.

Type  string

**supports_gpu**
GPU support flag.

Type  bool

**target**
Task target.

Type  Tasks(TargetType

**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*)
Initialize task parameters with a dictionary.

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

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

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

**interval**
Contour interval.

Type  float
max_value
    Maximum value of contour range.
    Type float

min_value
    Minimum value of contour range.
    Type float

name
    Task name.
    Type string

prevent_intersections
    Prevent contour intersections.
    Type bool

source_data
    Source data for contour generation.
    Type DataSource

supports_gpu
    GPU support flag.
    Type bool

target
    Task target.
    Type Tasks.TargetType

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)
    Initialize task parameters with a dictionary.

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

flip_x
    Flip X axis direction.
    Type bool
flip_y
   Flip Y axis direction.
   Type  bool

flip_z
   Flip Z axis direction.
   Type  bool

interpolation
   Interpolation mode.
   Type  Interpolation

max_workgroup_size
   Maximum workgroup size.
   Type  int

name
   Task name.
   Type  string

projection
   Output projection.
   Type  OrthoProjection

region
   Region to be processed.
   Type  BBox

resolution
   Output resolution in meters.
   Type  float

source_data
   Selects between dense point cloud and sparse point cloud.
   Type  DataSource

subdivide_task
   Enable fine-level task subdivision.
   Type  bool

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)*
Initialize task parameters with a dictionary.

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

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

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

**keep_depth**
Enable store depth maps option.
**Type** bool

**max_neighbors**
Maximum number of neighbor images to use for depth map filtering.
**Type** int

**max_workgroup_size**
Maximum workgroup size.
**Type** int

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

**point_colors**
Enable point colors calculation.
**Type** bool

**point_confidence**
Enable point confidence calculation.
**Type** bool

**subdivide_task**
Enable fine-level task subdivision.
**Type** bool

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

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

**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)
Initialize task parameters with a dictionary.

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

downslope
Depth map quality.
Type int

encode()
Create a dictionary with task parameters.

decodeJSON()
Create a JSON string with task parameters.

filter_mode
Depth map filtering mode.
Type FilterMode

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

max_workgroup_size
Maximum workgroup size.
Type int

name
Task name.
Type string

reuse_depth
Enable reuse depth maps option.
Type bool

subdivide_task
Enable fine-level task subdivision.
Type bool

supports_gpu
GPU support flag.
Type bool

target
Task target.
Type Tasks.TargetType
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.

class
    List of cameras to process.
    Type list of int

cameras
    Type list of int

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

classes
    Type list of int

decode (dict)
    Initialize task parameters with a dictionary.

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

face_count
    Target face count.
    Type FaceCount

face_count_custom
    Custom face count.
    Type int

interpolation
    Interpolation mode.
    Type Interpolation

keep_depth
    Enable store depth maps option.
    Type bool

max_workgroup_size
    Maximum workgroup size.
    Type int

name
    Task name.
    Type string
source_data
    Selects between dense point cloud, sparse point cloud and depth maps.
    Type DataSource

subdivide_task
    Enable fine-level task subdivision.
    Type bool

supports_gpu
    GPU support flag.
    Type bool

daughter_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

vertex_confidence
    Enable vertex confidence calculation.
    Type bool

volumetric_masks
    Enable strict volumetric masking.
    Type bool

workitem_count
    Work item count.
    Type int

workitem_size_cameras
    Number of cameras in a workitem.
    Type int

class 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)*
Initializes task parameters with a dictionary.

**decodeJSON** *(json)*
Initializes 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`

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

**refine_seamlines**
Refine seamlines based on image content.
Type `bool`

**region**
Region to be processed.
Type `BBox`

**resolution**
Pixel size in meters.
Type `float`

**resolution_x**
Pixel size in the X dimension in projected units.
Type `float`

**resolution_y**
Pixel size in the Y dimension in projected units.
Type `float`

**subdivide_task**
Enable fine-level task subdivision.
supports_gpu
Type bool

GPU support flag.

surface_data
Type DataSource

Orthorectification surface.

target
Type Tasks.TargetType

Task target.

workitem_count
Type int

Work item count.

workitem_size_cameras
Type int

Number of cameras in a workitem.

workitem_size_tiles
Type int

Number of tiles in a workitem.

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)

Initialize task parameters with a dictionary.

decodeJSON (json)

Initialize task parameters from a JSON string.

encode ()

Create a dictionary with task parameters.

encodeJSON ()

Create a JSON string with task parameters.

epsilon
Type float

Contour simplification threshold.

name
Type string

Task name.

supports_gpu
Type bool

GPU support flag.

target
Type Tasks.TargetType

Task target.
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)
    Initialize task parameters with a dictionary.

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

fill_holes
    Enable hole filling.
    Type  bool

ghosting_filter
    Enable ghosting filter.
    Type  bool

name
    Task name.
    Type  string

source_model
    Source model.
    Type  int

supports_gpu
    GPU support flag.
    Type  bool

target
    Task target.
    Type  Tasks.TargetType

texture_size
    Texture page size.
Type int

texture_type
Texture type.
  Type Model.TextureType

transfer_texture
Transfer texture.
  Type bool

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)
  Initialize task parameters with a dictionary.

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

encode ()
  Create a dictionary with task parameters.

encodeJSON ()
  Create a JSON string with task parameters.

face_count
  Number of faces per megapixel of texture resolution.
  Type int

ghosting_filter
  Enable ghosting filter.
  Type bool

keep_depth
  Enable store depth maps option.
  Type bool

max_workgroup_size
  Maximum workgroup size.
  Type int

name
  Task name.
  Type string

pixel_size
  Target model resolution in meters.
Type float

**source_data**
Selects between dense point cloud and mesh.
Type **DataSource**

**subdivide_task**
Enable fine-level task subdivision.
Type bool

**supports_gpu**
GPU support flag.
Type bool

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

**tile_size**
Size of tiles in pixels.
Type int

**transfer_texture**
Transfer source model texture to tiled model.
Type bool

**workitem_count**
Work item count.
Type int

**workitem_size_cameras**
Number of cameras in a workitem.
Type int

**class** **Tasks.BuildUV**
Task class containing processing parameters.

**adaptive_resolution**
Enable adaptive face detalization.
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.

**camera**
Camera to be used for texturing in MappingCamera mode.
Type int

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

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

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

**mapping_mode**
Texture mapping mode.
  Type *MappingMode*

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

**page_count**
Number of texture pages to generate.
  Type *int*

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

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

**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.

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

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

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

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

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

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

**source_data**
Source data for calibration.
  Type *DataSource*
supports_gpu
    GPU support flag.
    Type bool

target
    Task target.
    Type Tasks.TargetType

white_balance
    Calibrate white balance.
    Type bool

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)
    Initialize task parameters with a dictionary.

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

fit_b1
    Enable optimization of aspect ratio.
    Type bool

fit_b2
    Enable optimization of skew coefficient.
    Type bool

fit_cxcy
    Enable optimization of principal point coordinates.
    Type bool

fit_f
    Enable optimization of focal length coefficient.
    Type bool

fit_k1
    Enable optimization of k1 radial distortion coefficient.
    Type bool
fit_k2
   Enable optimization of k2 radial distortion coefficient.
   Type  bool

fit_k3
   Enable optimization of k3 radial distortion coefficient.
   Type  bool

fit_k4
   Enable optimization of k4 radial distortion coefficient.
   Type  bool

fit_p1
   Enable optimization of p1 tangential distortion coefficient.
   Type  bool

fit_p2
   Enable optimization of p2 tangential distortion coefficient.
   Type  bool

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)
   Initialize task parameters with a dictionary.

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

**decode** (*dict*)

Initialize task parameters with a dictionary.

**decodeJSON** (*json*)

Initialize task parameters from a JSON string.

**encode** ()

Create a dictionary with task parameters.

**encodeJSON** ()

Create a JSON string with task parameters.

**max_angle**

Maximum angle (degrees).

- **Type** float

**max_distance**

Maximum distance (meters).
Class **Tasks.ClassifyPoints**

Task class containing processing parameters.

```python
apply(object[, workitem][, progress])
```

Apply task to specified object.

**Parameters**

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

**confidence**

Required confidence level.

```python
Type float
```

**decode**(dict)

Initialize task parameters with a dictionary.

**decodeJSON**(json)

Initialize task parameters from a JSON string.

**encode**()

Create a dictionary with task parameters.

**encodeJSON**()

Create a JSON string with task parameters.

**name**

Task name.

```python
Type string
```

**source_class**

Class of points to be re-classified.

```python
Type int
```

**subdivide_task**

Enable fine-level task subdivision.

```python
Type bool
```

**supports_gpu**

GPU support flag.
Type bool

target
   Task target.
   Type Tasks.TargetType

target_classes
   Target point classes for classification.
   Type list of int

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)
   Initialize task parameters with a dictionary.

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

encode ()
   Create a dictionary with task parameters.

encodeJSON ()
   Create a JSON string with task parameters.

level
   Hole size threshold in percents.
   Type int

name
   Task name.
   Type string

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

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)
Initialize task parameters with a dictionary.

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

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

name
Task name.
Type string

source_data
Source data to extract colors from.
Type DataSource

supports_gpu
GPU support flag.
Type bool

target
Task target.
Type Tasks.TargetType

workitem_count
Work item count.
Type int

class Tasks.ColorizeModel
Task class containing processing parameters.

apply (object, workitem, progress)
Apply task to specified object.

Parameters

object (Chunk or Document) – Chunk or Document object to be processed.

workitem (int) – Workitem index.

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

decode (dict)
Initialize task parameters with a dictionary.

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

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

name
Task name.
Type string
source_data
Source data to extract colors from.
Type DataSource

supports_gpu
GPU support flag.
Type bool

target
Task target.
Type Tasks.TargetType

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)
Initialize task parameters with a dictionary.

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

encode()
Create a dictionary with task parameters.

encodeJSON()
Create a JSON string with task parameters.

name
Task name.
Type string

supports_gpu
GPU support flag.
Type bool

target
Task target.
Type Tasks.TargetType

workitem_count
Work item count.
Type int

class Tasks.ConvertImages
Task class containing processing parameters.

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

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

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

**color_correction**
- Apply color correction.
  - **Type** bool

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

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

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

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

**fix_pixel_aspect**
- Fix pixel aspect.
  - **Type** bool

**fix_principal_point**
- Fix principal point.
  - **Type** bool

**image_compression**
- Image compression parameters.
  - **Type** `ImageCompression`

**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`

**update_gps_tags**
- Update GPS tags.
  - **Type** bool

**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.

asset

Model to process.

Type int
decode (dict)

Initialize task parameters with a dictionary.
decodeJSON (json)

Initialize task parameters from a JSON string.
encode ()

Create a dictionary with task parameters.
encodeJSON ()

Create a JSON string with task parameters.

face_count

Target face count.

Type int

name

Task name.

Type string

supports_gpu

GPU support flag.

Type bool
target

Task target.

Type Tasks.TargetType

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)

Initialize task parameters with a dictionary.
decodeJSON (json)

Initialize task parameters from a JSON string.
encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

task
Task task.
Type  string

supportsgpu
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)
Initialize task parameters with a dictionary.

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

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

filter_mask
Ignore masked image regions.
Type  bool

frames
List of frames to process.
Type  list of int

inverted
Detect markers on black background.
Type  bool

maximum_residual
Maximum residual for non-coded targets in pixels.
Type float

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

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

**name**
Task name.
   Type string

**noparity**
Disable parity checking.
   Type bool

**supports_gpu**
GPU support flag.
   Type bool

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

**target_type**
Type of targets.
   Type TargetType

**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_clouds**
Copy dense clouds.
   Type bool

**copy_depth_maps**
Copy depth maps.
   Type bool

Chapter 2. Application Modules
copy_elevations
   Copy DEMs.
   Type bool

copy_keypoints
   Copy keypoints.
   Type bool

copy_models
   Copy models.
   Type bool

copy_orthomosaics
   Copy orthomosaics.
   Type bool

copy_tiled_models
   Copy tiled models.
   Type bool

decode (dict)
   Initialize task parameters with a dictionary.

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

encode ()
   Create a dictionary with task parameters.

encodeJSON ()
   Create a JSON string with task parameters.

frames
   List of frame keys to copy.
   Type list of int

label
   New chunk label.
   Type string

name
   Task name.
   Type string

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

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.
  - Type: Chunk or Document

• workitem (int) – Workitem index.
  - Type: int

• progress (Callable[[float], None]) – Progress callback.
  - Type: Callable

binary
  Enables/disables binary encoding for selected format (if applicable).
  - Type: bool

bingo_path_geoin
  Path to BINGO GEO INPUT file.
  - Type: string

bingo_path_gps
  Path to BINGO GPS/IMU file.
  - Type: string

bingo_path_image
  Path to BINGO IMAGE COORDINATE file.
  - Type: string

bingo_path_itera
  Path to BINGO ITERA file.
  - Type: string

bingo_save_geoin
  Enables/disables export of BINGO GEO INPUT file.
  - Type: bool

bingo_save_gps
  Enables/disables export of BINGO GPS/IMU data.
  - Type: bool

bingo_save_image
  Enables/disables export of BINGO IMAGE COORDINATE file.
  - Type: bool

bingo_save_itera
  Enables/disables export of BINGO ITERA file.
  - Type: bool

bundler_path_list
  Path to Bundler image list file.
  - Type: string

bundler_save_list
  Enables/disables export of Bundler image list file.
  - Type: bool

chan_rotation_order
  Rotation order (CHAN format only).
  - Type: RotationOrder

crs
  Output coordinate system.
  - Type: CoordinateSystem

decode (dict)
  Initialize task parameters with a dictionary.

decodeJSON (json)
  Initialize task parameters from a JSON string.
**encode**()  
Create a dictionary with task parameters.

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

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

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

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

**save_markers**  
Enables/disables export of manual matching points.  
Type `bool`

**save_points**  
Enables/disables export of automatic tie points.  
Type `bool`

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

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

**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)  
Initialize task parameters with a dictionary.

**decodeJSON**(json)  
Initialize task parameters from a JSON string.
encode()  
Create a dictionary with task parameters.

encodeJSON()  
Create a JSON string with task parameters.

export_depth  
Enable export of depth map.  
  Type bool

export_diffuse  
Enable export of diffuse map.  
  Type bool

export_normals  
Enable export of normal map.  
  Type bool

name  
Task name.  
  Type string

path_depth  
Path to depth map.  
  Type string

path_diffuse  
Path to diffuse map.  
  Type string

path_normals  
Path to normal map.  
  Type string

supports_gpu  
GPU support flag.  
  Type bool

target  
Task target.  
  Type Tasks.TargetType

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
crs
Output coordinate system.
    Type CoordinateSystem

decode (dict)
Initialize task parameters with a dictionary.
decodeJSON (json)
Initialize task parameters from a JSON string.
encode ()
Create a dictionary with task parameters.
encodeJSON ()
Create a JSON string with task parameters.

name
Task name.
    Type string

path
Path to output file.
    Type string

supports_gpu
GPU support flag.
    Type bool

target
Task target.
    Type Tasks.TargetType

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)
Initialize task parameters with a dictionary.
decodeJSON (json)
Initialize task parameters from a JSON string.
encode ()
Create a dictionary with task parameters.
encodeJSON ()
Create a JSON string with task parameters.
name
   Task name.
   Type string

path
   Path to output file.
   Type string

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

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

crs
   Output coordinate system.
   Type CoordinateSystem

decode (dict)
   Initialize task parameters with a dictionary.

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

encode ()
   Create a dictionary with task parameters.

encodeJSON ()
   Create a JSON string with task parameters.

format
   Export format.
   Type ModelFormat
name
   Task name.
   Type string

path
   Path to output model.
   Type string

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

raster_transform
   Raster band transformation.
   Type RasterTransformType

save_alpha
   Enables/disables alpha channel export.
   Type bool

save_cameras
   Enables/disables camera export.
   Type bool

save_colors
   Enables/disables export of vertex colors.
   Type bool

save_comment
   Enables/disables comment export.
   Type bool

save_markers
   Enables/disables marker export.
   Type bool

save_normals
   Enables/disables export of vertex normals.
   Type bool

save_texture
   Enables/disables texture export.
   Type bool

save_udim
   Enables/disables UDIM texture layout.
   Type bool

save_uv
   Enables/disables uv coordinates export.
   Type bool

shift
   Optional shift to be applied to vertex coordinates.
   Type Vector

strip_extensions
   Strips camera label extensions during export.
   Type bool
supports_gpu
    GPU support flag.
    Type bool

target
    Task target.
    Type Tasks.TargetType
texture_format
    Texture format.
    Type ImageFormat
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)
    Initialize task parameters with a dictionary.
decodeJSON (json)
    Initialize task parameters from a JSON string.
encode ()
    Create a dictionary with task parameters.
encodeJSON ()
    Create a JSON string with task parameters.
image_compression
    Image compression parameters.
    Type ImageCompression
name
    Task name.
    Type string
north_up
    Use north-up orientation for export.
    Type bool
path
    Path to output orthophoto.
    Type string
projection
  Output projection.
  Type OrthoProjection
raster_transform
  Raster band transformation.
  Type RasterTransformType
region
  Region to be exported.
  Type BBox
resolution
  Output resolution in meters.
  Type float
resolution_x
  Pixel size in the X dimension in projected units.
  Type float
resolution_y
  Pixel size in the Y dimension in projected units.
  Type float
save_alpha
  Enable alpha channel generation.
  Type bool
save_kml
  Enable kml file generation.
  Type bool
save_world
  Enable world file generation.
  Type bool
supports_gpu
  GPU support flag.
  Type bool
target
  Task target.
  Type Tasks.TargetType
white_background
  Enable white background.
  Type bool
workitem_count
  Work item count.
  Type int
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.
```python
camera_groups
    List of camera groups to process.
    Type: list of int

decode (dict)
    Initialize task parameters with a dictionary.

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

eencode ()
    Create a dictionary with task parameters.

eencodeJSON ()
    Create a JSON string with task parameters.

height
    Height of output panorama.
    Type: int

image_compression
    Image compression parameters.
    Type: ImageCompression

name
    Task name.
    Type: string

path
    Path to output file.
    Type: string

region
    Region to be exported.
    Type: BBox

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

**block_height**

Block height in meters.

**Type** float

**block_width**

Block width in meters.

**Type** float

**classes**

List of dense point classes to be exported.

**Type** list of int

**colors_rgb_8bit**

Convert colors to 8 bit RGB.

**Type** bool

**comment**

Optional comment (if supported by selected format).

**Type** string

**crs**

Output coordinate system.

**Type** CoordinateSystem

**decode** *(dict)*

Initialize task parameters with a dictionary.

**decodeJSON** *(json)*

Initialize task parameters from a JSON string.

**encode** *

Create a dictionary with task parameters.

**encodeJSON** *

Create a JSON string with task parameters.

**format**

Export format.

**Type** PointsFormat

**image_format**

Image data format.

**Type** ImageFormat

**name**

Task name.

**Type** string

**path**

Path to output file.

**Type** string

**raster_transform**

Raster band transformation.
Type `RasterTransformType`

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

**save_classes**
Enables/disables export of point classes.
Type `bool`

**save_colors**
Enables/disables export of point colors.
Type `bool`

**save_comment**
Enable comment export.
Type `bool`

**save_confidence**
Enables/disables export of point confidence.
Type `bool`

**save_images**
Enable image export.
Type `bool`

**save_normals**
Enables/disables export of point normals.
Type `bool`

**shift**
Optional shift to be applied to point coordinates.
Type `Vector`

**source_data**
Selects between dense point cloud and sparse point cloud. If not specified, uses dense cloud if available.
Type `DataSource`

**split_in_blocks**
Enable tiled export.
Type `bool`

**subdivide_task**
Enable fine-level task subdivision.
Type `bool`

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

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

**viewpoint**
Default view.
Type `Viewpoint`

**workitem_count**
Work item count.
**ExportRaster**

Task class containing processing parameters.

```python
apply(object, workitem, progress)
```

Apply task to specified object.

**Parameters**

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

**block_height**

Raster block height in pixels.

Type int

**block_width**

Raster block width in pixels.

Type int

**decode** *(dict)*

Initialize task parameters with a dictionary.

**decodeJSON** *(json)*

Initialize task parameters from a JSON string.

**description**

Export description.

Type string

**encode**

Create a dictionary with task parameters.

**encodeJSON**

Create a JSON string with task parameters.

**format**

Export format.

Type *RasterFormat*

**height**

Raster height.

Type int

**image_compression**

Image compression parameters.

Type *ImageCompression*

**image_description**

Optional description to be added to image files.

Type string

**image_format**

Tile format.

Type *ImageFormat*

**max_zoom_level**

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

Type int

**min_zoom_level**

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

name
    Task name.
    Type string

network_links
    Enable network links generation for KMZ format.
    Type bool

nodata_value
    No-data value (DEM export only).
    Type float

north_up
    Use north-up orientation for export.
    Type bool

path
    Path to output orthomosaic.
    Type string

projection
    Output projection.
    Type OrthoProjection

raster_transform
    Raster band transformation.
    Type RasterTransformType

region
    Region to be exported.
    Type BBox

resolution
    Output resolution in meters.
    Type float

resolution_x
    Pixel size in the X dimension in projected units.
    Type float

resolution_y
    Pixel size in the Y dimension in projected units.
    Type float

save_alpha
    Enable alpha channel generation.
    Type bool

save_kml
    Enable kml file generation.
    Type bool

save_scheme
    Enable tile scheme files generation.
    Type bool

save_world
    Enable world file generation.
    Type bool
source_data
Selects between DEM and orthomosaic.
Type DataSource

split_in_blocks
Split raster in blocks.
Type bool

supports_gpu
GPU support flag.
Type bool

target
Task target.
Type Tasks.TargetType

tile_height
Tile height in pixels.
Type int

tile_width
Tile width in pixels.
Type int

title
Export title.
Type string

white_background
Enable white background.
Type bool

width
Raster width.
Type int

workitem_count
Work item count.
Type int

world_transform
2x3 raster-to-world transformation matrix.
Type Matrix

class 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, p/q/r - estimated coordinates variance, i/j/k - estimated orientation angles variance, [] - group of multiple values, l - column separator within group).
Type  string

decode (dict)
Initialize task parameters with a dictionary.

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

delimiter
Column delimiter in csv format.
Type  string

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

format
Export format.
Type  ReferenceFormat

items
Items to export in CSV format.
Type  ReferenceItems

name
Task name.
Type  string

path
Path to the output file.
Type  string

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)
Initialize task parameters with a dictionary.

decodeJSON (json)
Initialize task parameters from a JSON string.
**description**
- Report description.
  
  **Type** string

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

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

**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

**user_settings**
- A list of user defined settings to include on the Processing Parameters page.
  
  **Type** list of (string, string) tuples

**workitem_count**
- Work item count.
  
  **Type** int

**class** 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.

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

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

**decodeJSON** (json)
- Initialize task parameters from a JSON string.
encode ()
  Create a dictionary with task parameters.

encodeJSON ()
  Create a JSON string with task parameters.

format
  Export format.
    Type ShapesFormat

groups
  A list of shape groups to export.
    Type list of int

name
  Task name.
    Type string

path
  Path to shape file.
    Type string

polygons_as_polylines
  Save polygons as polylines.
    Type bool

save_attributes
  Export attributes.
    Type bool

save_labels
  Export labels.
    Type bool

save_points
  Export points.
    Type bool

save_polygons
  Export polygons.
    Type bool

save_polylines
  Export polylines.
    Type bool

shift
  Optional shift to be applied to vertex coordinates.
    Type Vector

supports_gpu
  GPU support flag.
    Type bool

target
  Task target.
    Type Tasks.TargetType

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)
Initialize task parameters with a dictionary.

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

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

name
Task name.
Type string

path
Path to output file.
Type string

save_alpha
Enable alpha channel export.
Type bool

supports_gpu
GPU support flag.
Type bool

target
Task target.
Type Tasks.TargetType

texture_type
Texture type.
Type Model.TextureType

workitem_count
Work item count.
Type int

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

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

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

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

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

**format**
Export format.
   - **Type** `TiledModelFormat`

**image_compression**
Image compression parameters.
   - **Type** `ImageCompression`

**model_format**
Model format for zip export.
   - **Type** `ModelFormat`

**name**
Task name.
   - **Type** `string`

**path**
Path to output model.
   - **Type** `string`

**raster_transform**
Raster band transformation.
   - **Type** `RasterTransformType`

**supports_gpu**
GPU support flag.
   - **Type** `bool`

**target**
Task target.
   - **Type** `Tasks.TargetType`

**texture_format**
Texture format.
   - **Type** `ImageFormat`

**workitem_count**
Work item count.
   - **Type** `int`

**class** `Tasks.FilterDenseCloud`
Task class containing processing parameters.

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

```python
asset
Dense cloud key to filter.
  Type  int
decode (*dict*)
Initialize task parameters with a dictionary.
decodeJSON (*json*)
Initialize task parameters from a JSON string.
encode ()
  Create a dictionary with task parameters.
encodeJSON ()
  Create a JSON string with task parameters.
name
  Task name.
  Type  string
point_spacing
  Desired point spacing (m).
  Type  float
supports_gpu
  GPU support flag.
  Type  bool
target
  Task target.
  Type  Tasks.TargetType
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*)
Initialize task parameters with a dictionary.
decodeJSON (*json*)
Initialize task parameters from a JSON string.
encode ()
  Create a dictionary with task parameters.
encodeJSON ()
  Create a JSON string with task parameters.
format
   File format.
   Type CamerasFormat

image_list
   Path to image list file (Bundler format only).
   Type string

load_image_list
   Enable Bundler image list import.
   Type bool

name
   Task name.
   Type string

path
   Path to the file.
   Type string

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

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)
   Initialize task parameters with a dictionary.

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

encode ()
   Create a dictionary with task parameters.

encodeJSON ()
   Create a JSON string with task parameters.

name
   Task name.
   Type string

path
   Path to the file.
   Type string
supports_gpu
    GPU support flag.
    Type bool

target
    Task target.
    Type Tasks.TargetType

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)
    Initialize task parameters with a dictionary.
decodeJSON (json)
    Initialize task parameters from a JSON string.
encode ()
    Create a dictionary with task parameters.
encodeJSON ()
    Create a JSON string with task parameters.

name
    Task name.
    Type string

operation
    Mask operation.
    Type MaskOperation

path
    Mask file name template.
    Type string

source
    Mask source.
    Type MaskSource

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.

crs
  Model coordinate system.
  Type CoordinateSystem

decode (dict)
  Initialize task parameters with a dictionary.

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

encode ()
  Create a dictionary with task parameters.

encodeJSON ()
  Create a JSON string with task parameters.

format
  Model format.
  Type ModelFormat

name
  Task name.
  Type string

path
  Path to model.
  Type string

shift
  Optional shift to be applied to vertex coordinates.
  Type Vector

supports_gpu
  GPU support flag.
  Type bool

target
  Task target.
  Type Tasks.TargetType

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.

calculate_normals
Calculate point normals.

Type bool
crs
Point cloud coordinate system.

Type CoordinateSystem
decode (dict)
Initialize task parameters with a dictionary.
decodeJSON (json)
Initialize task parameters from a JSON string.
encode ()
Create a dictionary with task parameters.
encodeJSON ()
Create a JSON string with task parameters.
format
Point cloud format.

Type PointsFormat
name
Task name.

Type string
path
Path to point cloud.

Type string
shift
Optional shift to be applied to point coordinates.

Type Vector
supports_gpu
GPU support flag.

Type bool
target
Task target.

Type Tasks.TargetType
workitem_count
Work item count.

Type int
class Tasks.ImportRaster
Task class containing processing parameters.
apply (object[, workitem][, progress])
Apply task to specified object.

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

crs
Default coordinate system if not specified in GeoTIFF file.
Type CoordinateSystem
decode (dict)
Initialize task parameters with a dictionary.
decodeJSON (json)
Initialize task parameters from a JSON string.
encode ()
Create a dictionary with task parameters.
encodeJSON ()
Create a JSON string with task parameters.

name
Task name.
Type string

path
Path to elevation model in GeoTIFF format.
Type string

raster_type
Type of raster layer to import.
Type DataSource

supports_gpu
GPU support flag.
Type bool
target
Task target.
Type Tasks.TargetType

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).
create_markers
Create markers for missing entries (csv format only).
   Type bool

crs
Reference data coordinate system (csv format only).
   Type CoordinateSystem

decode (dict)
Initialize task parameters with a dictionary.

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

delimiter
Column delimiter in csv format.
   Type string

decode ()
Create a dictionary with task parameters.

decodeJSON ()
Create a JSON string with task parameters.

format
File format.
   Type ReferenceFormat

group_delimiters
Combine consecutive delimiters in csv format.
   Type bool

ignore_labels
Matches reference data based on coordinates alone (csv format only).
   Type bool

items
List of items to load reference for (csv format only).
   Type ReferenceItems

name
Task name.
   Type string

path
Path to the file with reference data.
   Type string

shutter_lag
Shutter lag in seconds (APM format only).
   Type float

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

supports_gpu
GPU support flag.
   Type bool
target
    Task target.
    Type Tasks.TargetType

threshold
    Error threshold in meters used when ignore_labels is set (csv format only).
    Type float

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

columns
    Column order in csv format (n - label, x/y/z - coordinates, d - description, [] - group of multiple values,
    l - column separator within group).
    Type string

crs
    Reference data coordinate system (csv format only).
    Type CoordinateSystem

decode(dict)
    Initialize task parameters with a dictionary.

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

delimiter
    Column delimiter in csv format.
    Type string

encode()
    Create a dictionary with task parameters.

encodeJSON()
    Create a JSON string with task parameters.

format
    Shapes format.
    Type ShapesFormat

group_delimiters
    Combine consecutive delimiters in csv format.
    Type bool

name
    Task name.
Type  string

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

**replace**
Replace current shapes with new data.
  Type  bool

**skip_rows**
Number of rows to skip in (csv format only).
  Type  int

**supports_gpu**
GPU support flag.
  Type  bool

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

**workitem_count**
Work item count.
  Type  int

### Class: `Tasks.ImportTiledModel`
Task class containing processing parameters.

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

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

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

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

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

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

**name**
Task name.
  Type  string

**path**
Path to tiled model.
  Type  string

**supports_gpu**
GPU support flag.
  Type  bool

**target**
Task target.
### 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.

**decode** *(dict)*

Initialize task parameters with a dictionary.

**decodeJSON** *(json)*

Initialize task parameters from a JSON string.

**encode** *

Create a dictionary with task parameters.

**encodeJSON** *

Create a JSON string with task parameters.

**name**

Task name.

**supports_gpu**

GPU support flag.

**target**

Task target.

**workitem_count**

Work item count.

### 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)*

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

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

name
Task name.
Type string

path
Path to project file.
Type string

read_only
Open project in read only mode.
Type bool

supports_gpu
GPU support flag.
Type bool

target
Task target.
Type Tasks.TargetType

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)
Initialize task parameters with a dictionary.

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

downsise
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

generic_preselection
  Enable generic preselection.
  Type bool

guided_matching
  Enable guided image matching.
  Type bool

keep_keypoints
  Store keypoints in the project.
  Type bool

keypoint_limit
  Key point limit.
  Type int

mask_tiepoints
  Apply mask filter to tie points.
  Type bool

max_workgroup_size
  Maximum workgroup size.
  Type int

name
  Task name.
  Type string

pairs
  User defined list of camera pairs to match.
  Type list of (int, int) tuples

reference_preselection
  Enable reference preselection.
  Type bool

reference_preselection_mode
  Reference preselection mode.
  Type ReferencePreselectionMode

reset_matches
  Reset current matches.
  Type bool

subdivide_task
  Enable fine-level task subdivision.
  Type bool

supports_gpu
  GPU support flag.
  Type bool

target
  Task target.
  Type Tasks.TargetType
tiepoint_limit
   Tie point limit.
   Type int

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)
   Initialize task parameters with a dictionary.

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

encode ()
   Create a dictionary with task parameters.

encodeJSON ()
   Create a JSON string with task parameters.

name
   Task name.
   Type string

source_data
   Asset type.
   Type DataSource

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

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)
Initialize task parameters with a dictionary.

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

encode ()
Create a dictionary with task parameters.

encodeJSON ()
Create a JSON string with task parameters.

merge_dense_clouds
Merge dense clouds.

Type bool

merge_depth_maps
Merge depth maps.

Type bool

merge_elevations
Merge DEMs.

Type bool

merge_markers
Merge markers.

Type bool

merge_models
Merge models.

Type bool

merge_orthomosaics
Merge orthomosaics.

Type bool

merge_tiepoints
Merge tie points.

Type bool

name
Task name.

Type string

supports_gpu
GPU support flag.

Type bool
target
    Task target.
    Type Tasks.TargetType

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)
        Initialize task parameters with a dictionary.

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

    encode ()
        Create a dictionary with task parameters.

    encodeJSON ()
        Create a JSON string with task parameters.

    fit_b1
        Enable optimization of aspect ratio.
        Type bool

    fit_b2
        Enable optimization of skew coefficient.
        Type bool

    fit_corrections
        Enable optimization of additional corrections.
        Type bool

    fitCx
        Enable optimization of X principal point coordinates.
        Type bool

    fitCy
        Enable optimization of Y principal point coordinates.
        Type bool

    fitF
        Enable optimization of focal length coefficient.
        Type bool

    fitK1
        Enable optimization of k1 radial distortion coefficient.
        Type bool
fit_k2
   Enable optimization of k2 radial distortion coefficient.
   Type bool

fit_k3
   Enable optimization of k3 radial distortion coefficient.
   Type bool

fit_k4
   Enable optimization of k3 radial distortion coefficient.
   Type bool

fit_p1
   Enable optimization of p1 tangential distortion coefficient.
   Type bool

fit_p2
   Enable optimization of p2 tangential distortion coefficient.
   Type bool

name
   Task name.
   Type string

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

tiepoint_covariance
   Estimate tie point covariance matrices.
   Type bool

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)
   Initialize task parameters with a dictionary.

decodeJSON (json)
   Initialize task parameters from a JSON string.
encode()  
Create a dictionary with task parameters.

encodeJSON()  
Create a JSON string with task parameters.

max_pitch  
Maximum camera pitch angle.  
Type int

max_waypoints  
Maximum waypoints per flight.  
Type int

min_altitude  
Minimum altitude (m).  
Type float

min_pitch  
Minimum camera pitch angle.  
Type int

min_waypoint_spacing  
Minimum waypoint spacing (m).  
Type float

name  
Task name.  
Type string

overlap  
Overlap percent.  
Type int

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_selection  
Focus on model selection.  
Type bool

workitem_count  
Work item count.  
Type int

class Tasks.ReduceOverlap  
Task class containing processing parameters.
Metashape Python Reference, Release 1.6.2

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

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

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

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

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

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

**max_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)
   Initialize task parameters with a dictionary.

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

downscale
   Refinement quality.
   Type int

encode ()
   Create a dictionary with task parameters.

encodeJSON ()
   Create a JSON string with task parameters.

iterations
   Number of refinement iterations.
   Type int

name
   Task name.
   Type string

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

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

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)*
Initialize task parameters with a dictionary.

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

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

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

**internal_blur**
Internal blur. Should be in range [0, 4].
Type float

**mesh_noise_suppression**
Mesh normals noise suppression strength. Should be in range [0, 4].
Type float

**name**
Task name.
Type string

**supports_gpu**
GPU support flag.
Type bool

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

**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)*
Initialize task parameters with a dictionary.

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

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

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

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType

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)
   Initialize task parameters with a dictionary.

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

encode ()
   Create a dictionary with task parameters.

encodeJSON ()
   Create a JSON string with task parameters.

name
   Task name.
   Type string

path
   Script path.
   Type string

supports_gpu
   GPU support flag.
   Type bool

target
   Task target.
   Type Tasks.TargetType
**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])
Initialize task parameters with a dictionary.

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

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

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

**name**
Task name.
  Type  string

**path**
Path to project.
  Type  string

**supports_gpu**
GPU support flag.
  Type  bool

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

**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_selection
    Apply to selected faces.

    Type  bool

decode (dict)
    Initialize task parameters with a dictionary.

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

encode ()
    Create a dictionary with task parameters.

encodeJSON ()
    Create a JSON string with task parameters.

fix_borders
    Fix borders.

    Type  bool

name
    Task name.

    Type  string

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 TasksTargetException
    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*)
Initialize task parameters with a dictionary.

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

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

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

**first_frame**
Starting frame index.

  Type int

**last_frame**
Ending frame index.

  Type int

**name**
Task name.

  Type string

**supports_gpu**
GPU support flag.

  Type bool

**target**
Task target.

  Type *Tasks.TargetType*

**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*)
Initialize task parameters with a dictionary.

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

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

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

**max_error**
Reprojection error threshold.
```python
Type  float

min_image
   Minimum number of point projections.
   Type  int

name
   Task name.
   Type  string

supports_gpu
   GPU support flag.
   Type  bool

target
   Task target.
   Type  Tasks.TargetType

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.TiledModel
Tiled model data.

class FaceCount
Tiled model face count in [LowFaceCount, MediumFaceCount, HighFaceCount]

TiledModel.bands
List of color bands.
Type list of string

TiledModel.clear()
Clears tiled model data.

TiledModel.copy()
Create a copy of the tiled model.

Returns Copy of the tiled model.

Return type TiledModel

TiledModel.crs
Reference coordinate system.
Type CoordinateSystem or None

TiledModel.data_type
Data type used to store color values.
Type DataType

TiledModel.key
Tiled model identifier.
Type int

TiledModel.label
Tiled model label.
Type string

TiledModel.meta
Tiled model meta data.
Type Metadata

TiledModel.modified
Modified flag.
Type bool
TiledModel.

pickPoint (origin, target, endpoints=1)

Returns ray intersection with the tiled model.

Parameters

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

Returns Coordinates of the intersection point.

Return type Vector

renderDepth (transform, calibration, resolution=1, cull_faces=True, add_alpha=True)

Render tiled model depth image for specified viewpoint.

Parameters

- **transform (Matrix)** – Camera location.
- **calibration (Calibration)** – Camera calibration.
- **resolution (float)** – Level of detail resolution in screen pixels.
- **cull_faces (bool)** – Enable back-face culling.
- **add_alpha (bool)** – Generate image with alpha channel.

Returns Rendered image.

Return type Image

renderImage (transform, calibration, resolution=1, cull_faces=True, add_alpha=True, raster_transform=RasterTransformNone)

Render tiled model image for specified viewpoint.

Parameters

- **transform (Matrix)** – Camera location.
- **calibration (Calibration)** – Camera calibration.
- **resolution (float)** – Level of detail resolution in screen pixels.
- **cull_faces (bool)** – Enable back-face culling.
- **add_alpha (bool)** – Generate image with alpha channel.
- **raster_transform (RasterTransformType)** – Raster band transformation.

Returns Rendered image.

Return type Image

renderMask (transform, calibration, resolution=1, cull_faces=True)

Render tiled model mask image for specified viewpoint.

Parameters

- **transform (Matrix)** – Camera location.
- **calibration (Calibration)** – Camera calibration.
- **resolution (float)** – Level of detail resolution in screen pixels.
- **cull_faces (bool)** – Enable back-face culling.
Returns Rendered image.

Return type \texttt{Image}

\texttt{TiledModel.renderNormalMap(transform, calibration, resolution=1, cull_faces=True, add_alpha=True)}

Render image with tiled model normals for specified viewpoint.

Parameters

- \texttt{transform (Matrix)} – Camera location.
- \texttt{calibration (Calibration)} – Camera calibration.
- \texttt{resolution (float)} – Level of detail resolution in screen pixels.
- \texttt{cull_faces (bool)} – Enable back-face culling.
- \texttt{add_alpha (bool)} – Generate image with alpha channel.

Returns Rendered image.

Return type \texttt{Image}

\texttt{TiledModel.transform}

4x4 dense cloud transformation matrix.

Type \texttt{Matrix}

\texttt{class Metashape.TiledModelFormat}

Tiled model format in \{TiledModelFormatNone, TiledModelFormatTLS, TiledModelFormatLOD, TiledModelFormatZIP, TiledModelFormatCesium, TiledModelFormatSLPK, TiledModelFormatOSGB\}

\texttt{class Metashape.Utils}

Utility functions.

\texttt{createChessboardImage(calib, cell_size=150, max_tilt=30)}

Synthesizes photo of a chessboard.

Parameters

- \texttt{calib (Calibration)} – Camera calibration.
- \texttt{cell_size (float)} – Chessboard cell size.
- \texttt{max_tilt (float)} – Maximum camera tilt in degrees.

Returns Resulting image.

Return type \texttt{Image}

\texttt{createDifferenceMask(image, background, tolerance=10, fit_colors=True)}

Creates mask from a pair of images or an image and specified color.

Parameters

- \texttt{image (Image)} – Image to be masked.
- \texttt{background (Image or color tuple)} – Background image or color value.
- \texttt{tolerance (int)} – Tolerance value.
- \texttt{fit_colors (bool)} – Enables white balance correction.

Returns Resulting mask.

Return type \texttt{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=0, minimum_dist=0)

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

euler2mat (rotation, euler_angles=EulerAnglesYPR)

Calculate camera to world rotation matrix from euler rotation angles.

Parameters
- rotation (Vector) – Rotation vector.
- euler_angles (EulerAngles) – Euler angles to use.

Returns Rotation matrix.

Return type Matrix

mat2euler (R, euler_angles=EulerAnglesYPR)

Calculate euler rotation angles from camera to world rotation matrix.

Parameters
- R (Matrix) – Rotation matrix.
- euler_angles (EulerAngles) – Euler angles to use.
Returns Rotation angles in degrees.
Return type Vector

**mat2opk** (*R*)
Calculate omega, phi, kappa from camera to world rotation matrix.

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

Returns Omega, phi, kappa angles in degrees.
Return type Vector

**mat2ypr** (*R*)
Calculate yaw, pitch, roll from camera to world rotation matrix.

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

Returns Yaw, pitch roll angles in degrees.
Return type Vector

**opk2mat** (*angles*)
Calculate camera to world rotation matrix from omega, phi, kappa angles.

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

Returns Rotation matrix.
Return type Matrix

**ypr2mat** (*angles*)
Calculate camera to world rotation matrix from yaw, pitch, roll angles.

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

Returns Rotation matrix.
Return type Matrix

class Metashape.Vector
n-component vector

```python
>>> 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.
• \( \mathbf{b} (\text{Vector}) \) – Second vector.

Returns  Cross product.

Return type  Vector

\text{norm}() 
Return norm of the vector.

\text{norm2}() 
Return squared norm of the vector.

\text{normalize}() 
Normalize vector to the unit length.

\text{normalized}() 
Return a new, normalized vector.

Returns  a normalized copy of the vector

Return type  Vector

\text{size} 
Vector dimensions.

Type  int

\text{w} 
Vector \( W \) component.

Type  float

\text{x} 
Vector \( X \) component.

Type  float

\text{y} 
Vector \( Y \) component.

Type  float

\text{z} 
Vector \( Z \) component.

Type  float

\text{zero}() 
Set all elements to zero.

class Metashape.Version
Version object contains application version numbers.

\text{build} 
Build number.

Type  int

\text{copy}() 
Return a copy of the object.

Returns  A copy of the object.

Return type  Version

\text{major} 
Major version number.
Type int

**micro**
Micro version number.

Type int

**minor**
Minor version number.

Type int

**class** Metashape.Viewpoint(app)
Represents viewpoint in the model view

**center**
Camera center.

Type Vector

**coo**
Center of orbit.

Type Vector

**copy()**
Return a copy of the object.

**Returns** A copy of the object.

**Return type** Viewpoint

**fov**
Camera vertical field of view in degrees.

Type float

**height**
OpenGL window height.

Type int

**mag**
Camera magnification defined by distance to the center of rotation.

Type float

**rot**
Camera rotation matrix.

Type Matrix

**width**
OpenGL window width.

Type int

**class** Metashape.Vignetting
Vignetting polynomial

**copy()**
Return a copy of the object.

**Returns** A copy of the object.

**Return type** Vignetting
3.1 Metashape version 1.6.2

- Added Application.ModelView and Application.OrthoView classes
- Added Application.removeMenuItem() method
- Added Model.transform() method
- Added PointCloud.cleanup() method
- Added Application.model_view and Application.ortho_view attributes
- Added BuildTexture.transfer_texture attribute
- Added PlanMission.min_pitch and PlanMission.max_pitch attributes
- Added columns, crs, delimiter, group_delimiters and skip_rows attributes to ImportShapes class
- Added CamerasFormatNVM to CamerasFormat enum
- Added PointsFormatPTX to PointsFormat enum
- Added ShapesFormatCSV to ShapesFormat enum
- Added transfer_texture argument to Chunk.buildTexture() method
- Added columns, crs, delimiter, group_delimiters and skip_rows arguments to Chunk.importShapes() method
- Moved ModelViewMode enum to ModelView class
- Renamed Application.console attribute to console_pane
- Renamed Application.captureModelView() method to ModelView.captureView()
- Renamed Application.captureOrthoView() method to OrthoView.captureView()
- Renamed Application.viewpoint attribute to ModelView.viewpoint
- Removed ReduceOverlap.capture_distance attribute
- Removed capture_distance argument from Chunk.reduceOverlap() method
- Changed default values of AlignCameras.reset_alignment and MatchPhotos.reset_matches attributes to False
- Changed default value of reset_alignment argument in Chunk.alignCameras() method to False
- Changed default value of reset_matches argument in Chunk.matchPhotos() method to False
3.2 Metashape version 1.6.1

- Added Application.releaseFreeMemory() method
- Added CoordinateSystem.towgs84 attribute
- Added Marker.enabled attribute
- Added BuildModel.subdivide_task attribute
- Added subdivide_task argument to Chunk.buildModel() method
- Changed default value of keep_depth argument in Chunk.buildModel() and Chunk.buildTiledModel() to True

3.3 Metashape version 1.6.0

- Added BBox, ImageCompression, RPCModel and Model.Texture classes
- Added Tasks.ImportTiledModel and Task.ColorizeModel classes
- Added CalibrationFormat and ReferencePreselectionMode enums
- Added Model.addTexture() and Model.remove() methods
- Added Model.getActiveTexture() and Model.setActiveTexture() methods
- Added NetworkClient.setMasterServer() method
- Added setClassesFilter(), setConfidenceFilter(), setSelectionFilter() and resetFilters() methods to DenseCloud class
- Added renderDepth(), renderImage(), renderMask() and renderNormalMap() methods to PointCloud, DenseCloud and TiledModel classes
- Added Chunk.renderPreview() method
- Added Utils.euler2mat() and Utils.mat2euler() methods
- Added Calibration.rpc attribute
- Added Marker.position_covariance attribute
- Added Model.textures attribute
- Added TiledModel.crs and TiledModel.transform attributes
- Added EulerAnglesPOK and EulerAnglesANK values to EulerAngles enum
- Added PointsFormatPCD to PointsFormat enum
- Added ShapesFormatGeoJSON to ShapesFormat enum
- Added RPC to Sensor.Type enum
- Added image_compression attribute to ExportOrthophotos, ExportRaster, ExportTiledModel and UndistortPhotos classes
- Added AddPhotos.load_rpc_txt attribute
- Added AlignCameras.min_image attribute
- Added BuildDenseCloud.point_confidence attribute
- Added BuildModel.vertex_confidence, BuildModel.max_workgroup_size and BuildModel.workitem_size_cameras attributes
• Added BuildTexture.source_model and BuildTexture.texture_type attributes
• Added BuildUV.adaptive_resolution attribute
• Added DecimateModel.asset attribute
• Added ExportPanorama.image_compression attribute
• Added ExportPoints.save_classes and ExportPoints.save_confidence attributes
• Added ExportTexture.texture_type attribute
• Added ExportTiledModel.crs attribute
• Added ImportCameras.image_list and ImportCameras.load_image_list attributes
• Added ImportPoints.calculate_normals attribute
• Added MatchPhotos.guided_matching and MatchPhotos.reference_preselection_mode attributes
• Added MergeChunks.merge_depth_maps, MergeChunks.merge_elevations and MergeChunks.merge_orthomosaics attributes
• Added OptimizeCameras.fit_corrections attribute
• Added TriangulatePoints.max_error and TriangulatePoints.min_image attributes
• Added endpoints argument to PointCloud.pickPoint(), DenseCloud.pickPoint(), Model.pickPoint() and Tiled-Model.pickPoint() methods
• Added compression argument to Image.save() method
• Added cull_faces and add_alpha arguments to Model.renderDepth() method
• Added cull_faces, add_alpha and raster_transform arguments to Model.renderImage() method
• Added cull_faces argument to Model.renderMask() method
• Added cull_faces and add_alpha arguments to Model.renderNormalMap() method
• Moved TiffCompression enum to ImageCompression class
• Renamed Tasks.UndistortPhotos class to Tasks.ConvertImages
• Renamed Chunk.estimateImageQuality() method to Chunk.analyzePhotos()
• Renamed Chunk.buildPoints() method to Chunk.triangulatePoints()
• Renamed Chunk.loadReference() method to Chunk.importReference()
• Renamed Chunk.saveReference() method to Chunk.exportReference()
• Renamed Chunk.refineModel() method to Chunk.refineMesh()
• Renamed network_distribute tasks attribute to subdivide_task
• Renamed AlignChunks.align_method attribute to method
• Renamed AlignChunks.match_downscale attribute to downscale
• Renamed AlignChunks.match_filter_mask attribute to filter_mask
• Renamed AlignChunks.match_mask_tiepoints attribute to mask_tiepoints
• Renamed AlignChunks.match_point_limit attribute to keypoint_limit
• Renamed AlignChunks.match_select_pairs attribute to generic_preselection
• Renamed BuildDenseCloud.store_depth attribute to keep_depth
• Renamed BuildModel.store_depth attribute to keep_depth
• Renamed BuildOrthomosaic.ortho_surface attribute to surface_data
• Renamed BuildTiledModel.store_depth attribute to keep_depth
• Renamed BuildUV.texture_count attribute to page_count
• Renamed CalibrateColors.data_source attribute to source_data
• Renamed CalibrateColors.calibrate_color_balance attribute to white_balance
• Renamed ClassifyGroundPoints.cls_from attribute to source_class
• Renamed ClassifyPoints.cls_from attribute to source_class
• Renamed ClassifyPoints.cls_to attribute to target_classes
• Renamed DecimateModel.target_face_count attribute to face_count
• Renamed DuplicateChunk.copy_dense_cloud attribute to copy_dense_clouds
• Renamed ClassifyPoints.copy_elevation attribute to copy_elevations
• Renamed ClassifyPoints.copy_model attribute to copy_models
• Renamed ClassifyPoints.copy_orthomosaic attribute to copy_orthomosaics
• Renamed ClassifyPoints.copy_tiled_model attribute to copy_tiled_models
• Renamed ExportCameras.bingo_export_geoin attribute to bingo_save_geoin
• Renamed ExportCameras.bingo_export_gps attribute to bingo_save_gps
• Renamed ExportCameras.bingo_export_image attribute to bingo_save_image
• Renamed ExportCameras.bingo_export_itera attribute to bingo_save_itera
• Renamed ExportCameras.bundler_export_list attribute to bundler_save_list
• Renamed ExportCameras.chan_order_rotate attribute to chan_rotation_order
• Renamed ExportCameras.coordinates attribute to crs
• Renamed ExportCameras.export_markers attribute to save_markers
• Renamed ExportCameras.export_points attribute to save_points
• Renamed ExportMarkers.coordinates attribute to crs
• Renamed ExportModel.coordinates attribute to crs
• Renamed ExportModel.export_alpha attribute to save_alpha
• Renamed ExportModel.export_cameras attribute to save_cameras
• Renamed ExportModel.export_colors attribute to save_colors
• Renamed ExportModel.export_comment attribute to save_comment
• Renamed ExportModel.export_markers attribute to save_markers
• Renamed ExportModel.export_normals attribute to save_normals
• Renamed ExportModel.export_texture attribute to save_texture
• Renamed ExportModel.export_udim attribute to save_udim
• Renamed ExportModel.export_uv attribute to save_uv
• Renamed ExportOrthophotos.write_alpha attribute to save_alpha
• Renamed ExportOrthophotos.write_kml attribute to save_kml
• Renamed ExportOrthophotos.write_world attribute to save_world
• Renamed ExportPoints.coordinates attribute to crs
• Renamed ExportPoints.data_source attribute to source_data
• Renamed ExportPoints.export_colors attribute to save_colors
• Renamed ExportPoints.export_comment attribute to save_comment
• Renamed ExportPoints.export_images attribute to save_images
• Renamed ExportPoints.export_normals attribute to save_normals
• Renamed ExportPoints.tile_height attribute to block_height
• Renamed ExportPoints.tile_width attribute to block_width
• Renamed ExportPoints.write_tiles attribute to split_in_blocks
• Renamed ExportRaster.data_source attribute to source_data
• Renamed ExportRaster.kmz_section_enable attribute to network_links
• Renamed ExportRaster.tile_width attribute to block_width
• Renamed ExportRaster.tile_height attribute to block_height
• Renamed ExportRaster.write_alpha attribute to save_alpha
• Renamed ExportRaster.write_kml attribute to save_kml
• Renamed ExportRaster.write_scheme attribute to save_scheme
• Renamed ExportRaster.write_tiles attribute to split_in_blocks
• Renamed ExportRaster.write_world attribute to save_world
• Renamed ExportRaster.xyz_level_min attribute to min_zoom_level
• Renamed ExportRaster.xyz_level_max attribute to max_zoom_level
• Renamed ExportShapes.coordinates attribute to crs
• Renamed ExportShapes.export_attributes attribute to save_attributes
• Renamed ExportShapes.export_labels attribute to save_labels
• Renamed ExportShapes.export_points attribute to save_points
• Renamed ExportShapes.export_polygons attribute to save_polygons
• Renamed ExportShapes.export_polylines attribute to save_polylines
• Renamed ExportTexture.write_alpha attribute to save_alpha
• Renamed ExportTiledModel.mesh_format attribute to model_format
• Renamed ImportMasks.method attribute to source
• Renamed ImportModel.coordinates attribute to crs
• Renamed ImportPoints.coordinates attribute to crs
• Renamed ImportReference.coordinates attribute to crs
• Renamed MatchPhotos.preselection_generic attribute to generic_preselection
• Renamed MatchPhotos.preselection_reference attribute to reference_preselection
• Renamed MatchPhotos.store_keypoints attribute to keep_keypoints
• Renamed RefineMesh.niterations attribute to iterations
• Renamed SmoothModel.apply_to_selected attribute to apply_to_selection
• Renamed TrackMarkers.frame_start attribute to first_frame
• Renamed TrackMarkers.frame_end attribute to last_frame
• Renamed processing methods arguments to match task parameters names (e.g. dx/dy -> resolution_x/resolution_y, write_xxx -> save_xxx, export_xxx -> save_xxx, import_xxx -> load_xxx, preselection_generic -> generic_preselection, preselection_reference -> reference_preselection, source_data -> data_source, etc.)
• Replaced Chunk.importDem() method with Chunk.importRaster() method
• Replaced Chunk.exportDem() and Chunk.exportOrthomosaic() methods with Chunk.exportRaster() method
• Removed Accuracy and Quality enums
• Removed Model.texture() and Model.setTexture() methods
• Removed ExportPoints.precision attribute
• Removed OptimizeCameras.fit_p3 and OptimizeCameras.fit_p4 attributes
• Removed PlanMission.max_cameras and PlanMission.use_cameras attributes
• Removed tiff_big, tiff_tiled and tiff_overviews attributes from ExportOrthophotos and ExportRaster classes
• Removed tiff_compression attribute from ExportOrthophotos, ExportRaster and UndistortPhotos classes
• Removed jpeg_quality attribute from ExportOrthophotos, ExportRaster, ExportTiledModel and UndistortPhotos classes

3.4 Metashape version 1.5.5

No Python API changes

3.5 Metashape version 1.5.4

• Added Tasks.FilterDenseCloud class
• Added TiledModel.FaceCount enum
• Added copy() method to Antenna, Calibration, ChunkTransform, CirTransform, CoordinateSystem, Document, MetaData, OrthoProjection, RasterTransform, Region, Shutter, Target, Version, Viewpoint and Vignetting classes
• Added CameraTrack.save() and CameraTrack.load() methods
• Added Chunk.reduceOverlap() method
• Added location_enabled and rotation_enabled attributes to Sensor.Reference class
• Added CameraTrack.chunk and CameraTrack.meta attributes
• Added BuildTiledModel.ghosting_filter and BuildTiledModel.transfer_texture attributes
• Added ExportPoints.network_distribute and ExportPoints.region attributes
• Added ExportTiledModel.jpeg_quality and ExportTiledModel.texture_format attributes
• Added prevent_intersections argument to Chunk.buildContours() method
• Added transfer_texture argument to Chunk.buildTiledModel() method
• Added region argument to Chunk.exportPoints() method
• Added texture_format and jpeg_quality arguments to Chunk.exportTiledModel() method
• Added progress argument to Chunk.importMarkers() method
• Added ImageFormatWebP to ImageFormat enum

3.6 Metashape version 1.5.3

• Added DepthMap.getCalibration() and DepthMap.setCalibration() methods
• Added NetworkClient.dumpBatches(), NetworkClient.loadBatches() and NetworkClient.setBatchNodeLimit() methods
• Added location_enabled and rotation_enabled attributes to Camera.Reference class
• Added keep_depth argument to Chunk.buildTiledModel() method
• Added uv argument to Chunk.exportModel() method
• Added level argument to DepthMap.image() and DepthMap.setImage() methods
• Changed default value of keep_depth argument in Chunk.buildDenseCloud() and Chunk.buildModel() methods to True
• Changed default value of max_neighbors argument in Chunk.buildDenseCloud() method to 100

3.7 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

3.8 Metashape version 1.5.1

• Added License class
• Added Tasks.MergeAssets class
• Added Metashape.license attribute
• Renamed Tasks.OptimizeCoverage class to Tasks.PlanMotion

3.9 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

3.10 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

3.11 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

3.12 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

3.13 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

3.14 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 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 fixBorders 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

3.15 PhotoScan version 1.3.5

No Python API changes

3.16 PhotoScan version 1.3.4

No Python API changes

3.17 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
3.18 PhotoScan version 1.3.2

- Added `vertex_colors` argument to `Chunk.buildModel()` method
- Added `Shape.vertex_ids` attribute

3.19 PhotoScan version 1.3.1

- Added `Settings` and `TiledModel` classes
- Added `Application.getBool()` method
- Added `Camera.unproject()` method
- Added `Chunk.addFrames()`, `Chunk.addMarkerGroup()`, `Chunk.addScalebarGroup()` and `Chunk.buildSeamlines()` methods
- Added `DenseCloud.pickPoint()` and `DenseCloud.updateStatistics()` methods
- Added `Elevation.altitude()` method
- Added `Matrix.svd()` method
- Added `Model.pickPoint()` method
- Added `Orthomosaic.reset()` and `Orthomosaic.update()` methods
- Added `PointCloud.pickPoint()` method
- Added `filter` argument to `Application.getOpenFileName()`, `Application.getOpenFileNames()` and `Application.getSaveFileName()` methods
- Added `point` and `visibility` arguments to `Chunk.addMarker()` method
- Added `raster_transform` and `write_scheme` arguments to `Chunk.exportDem()` method
- Added `write_scheme` and `white_background` arguments to `Chunk.exportOrthomosaic()` method
- Added `white_background` argument to `Chunk.exportOrthophotos()` method
- Added `projection` argument to `Chunk.exportMarkers()` method
- Added `markers` argument to `Chunk.exportModel()` method
- Added `pairs` argument to `Chunk.matchPhotos()` method
- Added `columns` and `delimiter` arguments to `Chunk.saveReference()` method
- Added `version` argument to `Document.save()` method
- Renamed `npasses` argument in `Chunk.smoothModel()` method to `strength` and changed its type to `float`
- Renamed `from` and `to` arguments in `CoordinateSystem.transform()`, `DenseCloud.assignClass()`, `DenseCloud.assignClassToSelection()` and `DenseCloud.classifyGroundPoints()` methods to avoid collision with reserved words
- Added `Application.settings` attribute
- Added `Chunk.tiled_model` attribute
- Added `ShapeGroup.color` and `ShapeGroup.show_labels` attributes
- Added `ImageFormatTGA` to `ImageFormat` enum
3.20 PhotoScan version 1.3.0

- Added MarkerGroup, Masks, ScalebarGroup, Shutter and Thumbnails classes
- Added Application.PhotosPane class
- Added Model.Statistics class
- Added Orthomosaic.Patch and Orthomosaic.Patches classes
- Added PointCloud.Filter class
- Added CamerasFormat, EulerAngles, ImageFormat, ImageLayout, MaskOperation, MaskSource, MatchesFormat, ModelFormat, ModelViewMode, PointClass, PointsFormat, RasterFormat, ReferenceFormat, ReferenceItems, RotationOrder, TiffCompression, TiledModelFormat enums
- Added Application.captureOrthoView() method
- Added Chunk.refineMarkers() method
- Added CoordinateSystem.listBuiltinCRS() class method
- Added Matrix.translation() method
- Added Model.statistics() method
- Added NetworkClient.serverInfo(), NetworkClient.nodeStatus(), NetworkClient.setNodeCapability() and NetworkClient.quitNode() methods
- Added Photo.imageMeta() method
- Added Shape.area(), Shape.perimeter2D(), Shape.perimeter3D() and Shape.volume() methods
- Added Utils.createMarkers() method
- Added source argument to Application.captureModelView() method
- Added image_format argument to Chunk.exportDem() method
- Added write_alpha argument to Chunk.exportOrthophotos() method
- Added image_format and write_alpha arguments to Chunk.exportOrthomosaic() method
- Added groups, projection, shift and progress arguments to Chunk.exportShapes() method
- Added items and progress arguments to Chunk.copy() method
- Added sensor argument to Chunk.addCamera() method
- Added layout argument to Chunk.addPhotos() method
- Added jpeg_quality argument to Chunk.exportOrthomosaic() and Chunk.exportOrthophotos() methods
- Added fill_holes argument to Chunk.buildOrthomosaic() method
- Added fit_shutter argument to Chunk.optimizeCameras() method
- Added settings argument to Chunk.exportReport() method
- Added progress argument to various DenseCloud methods
- Added from argument to DenseCloud.classifyGroundPoints() method
- Added chunks and progress arguments to Document.append() method
- Added progress argument to Document.alignChunks() and Document.mergeChunks() methods
- Added revision argument to NetworkClient.batchList(), NetworkClient.batchStatus() methods
• Added Application.photos_pane attribute
• Added Camera.shutter attribute
• Added Chunk.masks and Chunk.thumbnails attributes
• Added Chunk.marker_groups and Chunk.scalebar_groups attributes
• Added Chunk.euler_angles and Chunk.scalebar_accuracy attributes
• Added CoordinateSystem.name attribute
• Added Marker.group and Scalebar.group attributes
• Added Orthomosaic.patches attribute
• Added RasterTransform.false_color attribute
• Added Sensor.bands attribute
• Added Shape.attributes attribute
• Added DepthMapsData, TiledModelData and OrthomosaicData to DataSource enum
• Added CircularTarget14bit to TargetType enum
• Renamed CameraReference class to Camera.Reference
• Renamed ConsolePane class to Application.ConsolePane
• Renamed MarkerProjection class to Marker.Projection
• Renamed MarkerProjections class to Marker.Projections
• Renamed MarkerReference class Marker.Reference
• Renamed MeshFace class to Model.Face
• Renamed MeshFaces class to Model.Faces
• Renamed MeshTexVertex class to Model.TexVertex
• Renamed MeshTexVertices class to Model.TexVertices
• Renamed MeshVertex class to Model.Vertex
• Renamed MeshVertices class to Model.Vertices
• Renamed PointCloudCameras class to PointCloud.Cameras
• Renamed PointCloudPoint class to PointCloud.Point
• Renamed PointCloudPoints class to PointCloud.Points
• Renamed PointCloudProjection class to PointCloud.Projection
• Renamed PointCloudProjections class to PointCloud.Projections
• Renamed PointCloudTrack class to PointCloud.Track
• Renamed PointCloudTracks class to PointCloud.Tracks
• Renamed ScalebarReference class to Scalebar.Reference
• Renamed ShapeVertices class to Shape.Vertices
• Renamed Application.enumOpenCLDevices() method to Application.enumGPUDevices()
Renamed 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 rotation_order argument in Chunk.exportCameras() to RotationOrder
Changed type of format argument in Chunk.exportCameras() and Chunk.exportCameras() methods to CamerasFormat
Changed type of format argument in Chunk.exportDem() and Chunk.exportOrthomosaic() methods to RasterFormat
Changed type of format argument in Chunk.exportMatches() method to MatchesFormat
Changed type of texture_format argument in Chunk.exportModel() method to ImageFormat
Changed type of format argument in Chunk.importModel() and Chunk.exportModel() methods to ModelFormat
Changed type of format argument in Chunk.exportPoints() method to PointsFormat
Changed type of tiff_compression argument in Chunk.exportOrthomosaic() and Chunk.exportOrthophotos() methods to TiffCompression
Changed type of items argument in Chunk.exportShapes() method to Shape.Type
Changed type of format argument in Chunk.exportTiledModel() method to TiledModelFormat
Changed type of mesh_format argument in Chunk.exportTiledModel() method to ModelFormat
Changed type of operation argument in Chunk.importMasks() method to MaskOperation
Changed type of format argument in Chunk.loadReference() and Chunk.saveReference() methods to ReferenceFormat
Changed type of items argument in Chunk.saveReference() method to ReferenceItems
Removed return values from Camera.open(), Chunk.addPhotos(), Chunk.alignCameras(), Chunk.buildContours(), Chunk.buildDem(), Chunk.buildDenseCloud(), Chunk.buildModel(), Chunk.buildOrthomosaic(), Chunk.buildPoints(), Chunk.buildTexture(), Chunk.buildTiledModel(), Chunk.buildUV(), Chunk.decimateModel(), Chunk.detectMarkers(), Chunk.estimateImageQuality(), Chunk.exportCameras(), Chunk.exportDem(), Chunk.exportMarkers(), Chunk.exportMatches(), Chunk.exportModel(), Chunk.exportOrthomosaic(), Chunk.exportOrthophotos(), Chunk.exportPoints(), Chunk.exportReport(), Chunk.exportShapes(), Chunk.exportTiledModel(), Chunk.importCameras(), Chunk.importDem(), Chunk.importMarkers(), Chunk.import Masks(), 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(),

- 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

### 3.21 PhotoScan version 1.2.6

No Python API changes

### 3.22 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

### 3.23 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

3.24 PhotoScan version 1.2.3

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

3.25 PhotoScan version 1.2.2

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

3.26 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

3.27 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

3.28 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

3.29 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

3.30 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

3.31 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

3.32 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)

### 3.33 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

### 3.34 PhotoScan version 0.8.4

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

### 3.35 PhotoScan version 0.8.3

Initial version of PhotoScan Python API
m
MetaShape, 5