

# Script Bride Plugin (SBP) usage

Dev version of the documentation.

Texts in **green** are my side notes or open questions.

## The plugin

Official name: **Script Bridge**

The purpose of the plugin is to allow a process to use and process data acquired in Maps and return new data or add other project content. The original requirement (and expected typical use case) is to support the Python scripting. However, Maps executes and communicates with a (script) executable independently from its actual implementation - the plugin is not limited solely to Python, but it can use any processing that implements necessary communication. Therefore there is no explicit mention of Python in the Maps plugin UI.

All examples in this document will be in Python.

Thus “Script Bridge”, not “Python Bridge”. There is even a question whether we should not drop “script” and call it “Processing Bridge”. Or maybe we can be more conservative and use “Python Bridge” – either for marketing reasons or because nobody has actually tested anything else than Python. Anyway: There is no reason why any other process should not work.

What about licensing? I expect no specific license for the plugin, but obvious `IsOfflineViewer = false` and possibly other common flags?

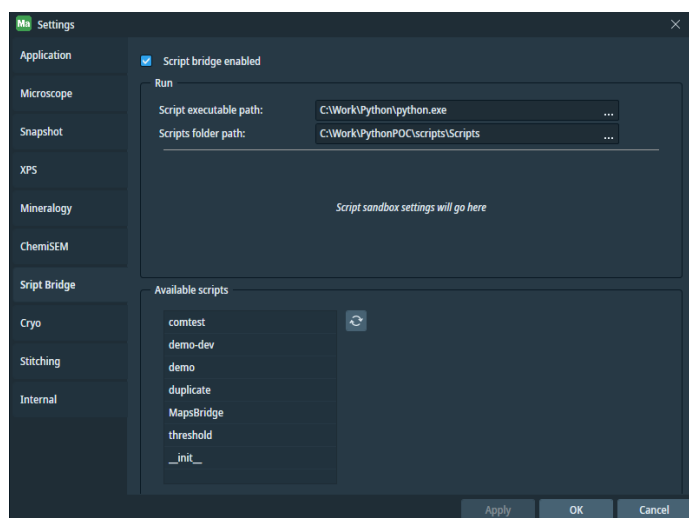
## Licensing

The plugin is licensed for Correlative Workflow, which includes flags:

- ColorCorrelation
- StageCorrelation
- ExternalImageImport

## Script Bridge settings

Settings page is in Maps main menu -> Options -> Script Bridge :



**Script Bridge enabled** checkbox – enables the plugin as whole. By default, the plugin is enabled. Changing this property requires Maps restart. If the plugin is NOT enabled, then the Script Bridge tile set tab (see below) is not displayed and of

course no processing is run. Enabling/disabling does not affect existing properties set in a tile set (properties once set are not erased).

**Script executable path** – Select a full path to the script executable (including the executable itself, i.e. python.exe). Installation, setup and selection of it is up users.

**Script folder path** – Full path to folder that contains scripts. Maps will probe the folder, reads all script files there and fills them to the list of available scripts.

This is the only place where Python and scripts are actually required. Maps searches for \*.py files and presumes that the processing executable uses scripts at all. For other processing systems we would have at least let users to specify script extensions, or let the system to give Maps the list of “scripts”. This was implemented for the remote processing mode (now removed), but not for the local one.

**Script sandbox settings** – TODO. In this place users will setup the sandbox parameters to satisfy the great Cybersecurity Resilience Act (CSA). This will include:

- checkbox to explicitly confirm that the default parameters are to be changed
- checkbox to enable network communication
- settings of max RAM, disk space, process number, etc.
- specification of folders that can be accessed from the processing (writable, read-only, temp folders)
- etc.

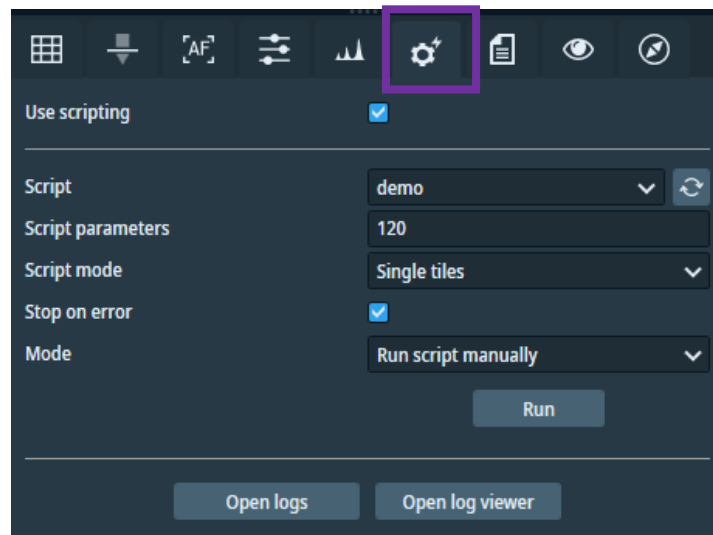
We are all happy about CSA...

**Available scripts** – List of available scripts read from the folder. If for any reason not refreshed automatically, the refresh button can be used.


Any changes in the settings must be confirmed with Apply or OK buttons of the dialog.

## Tile set Script Bridge settings tab

The tab contains setup for the scripting of the selected tile set.



**Use scripting** – Check to enable a scripting for this tile set

**Script** – Select or type a script name to be run. If empty, scripting is disabled. Use the refresh button  to read available scripts and refresh their default parameters (if any). Note that default parameters are applied only when a script is selected or changed.

**Script parameters** – Any string that should be sent to the script as an additional parameter. The string is sent to the string as is and it is up the script how to use it. Maps does not process the string in any way.

**Script mode** comb box – select mode HOW the script is run. See description below.

**Stop on error** – If the script processing should stop if an error occurs. See description below. **Not yet implemented.**

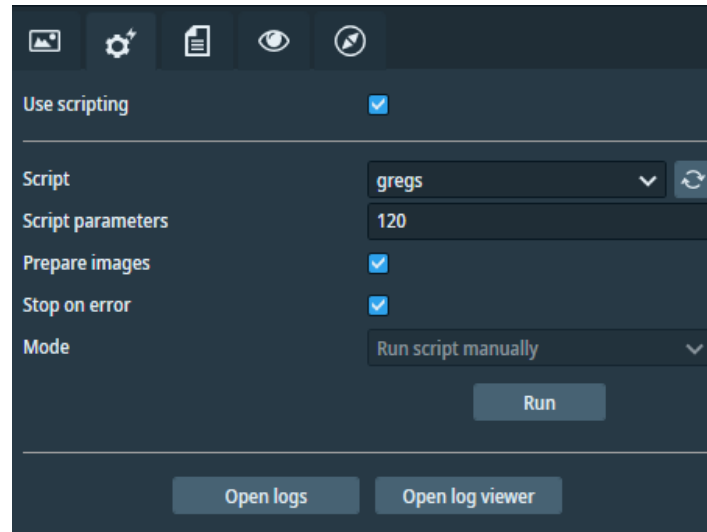
**Mode** combo box – select the run mode WHEN the script should be run. See description below.

**Run**– Click to run (or schedule) the scripting. Available only for *Run script manually*.

**Open logs** – Opens logs of the scripting for this tile set (regardless the script, i.e. the logs contain the whole scripting history). Logs are open as plain text in the system default txt application.

**Open log viewer** – Open logs in the dedicated ScriptBridge log viewer.

## Image layer Script Bridge settings tab



Almost identical to the tile set setup. There is no **Script mode** to be selected - only “batch” would make sense for image layers as there are no tiles in image layers. **Mode** is disabled and set to *Run script manually*, the only mode possible for image layers.

**Prepare images** – If selected, the script processing will extract the most detailed pyramid level to single images, one per image layers channel. Scripts then can use the images for analysis.

Note: there is a limitation based on max size of bitmaps ((32,767 px, disk size, plus other limits), therefore it is not possible to prepare images for larger image layers.

We will disable and uncheck this feature automatically for large layers. Otherwise the script process would fail immediately during the preparation. Scripts can still use pyramid tiles for their purposes.

There is a possible workaround in using tiled tiffs for the export, but we will probably not pursue this in the current version.

## Running scripts

There are following available running strategies:

Help me reword names of the strategies to be better understood by users.

- **Run script manually**

The script is to be run manually with “Run” button. A script job is created in the Processing queue and it is run immediately if possible. It may be run later if the queue already contains other scheduled or running jobs. The job can be stopped (i.e. the whole processing queue is stopped) and restarted. The order of jobs can be rearranged. The scripting job is also persistent – it is saved together with the containing project and can be started again when the project is reloaded.

- **Run when completed**

The script will be automatically run when the tile set acquisition completes.

- **Run when completed (blocking)**

**Not yet implemented.** The script will be automatically run when the tile set acquisition completes. The acquisition job queue will be blocked until the script processing completes.

- **Run during acquisition**

Script is called during the tile set acquisition after each tile acquisition and processing is finished. The acquisition

process waits for the script processing to complete before it continues with a next tile. This is the only case when there is no dedicated script job created in the processing queue – the script processing becomes a part of the acquisition job. But I consider creating a “dummy” script job for this anyway. The job would be solely for better tracking of the scripting progress, otherwise the job would not be startable on its own.

- **Run during acquisition async**

Script is called during the tile set acquisition. After each tile acquisition and processing is finished, a request is sent to the scripting, but the acquisition does not wait for the scripting completion. The scripting job runs independently and asynchronously from the acquisition.

Only one scripting process can be performed or scheduled for the tile set. In other cases the script setup is disabled, until the current scripting is finished or removed (for this, open the Processing job and stop and remove the scripting job).

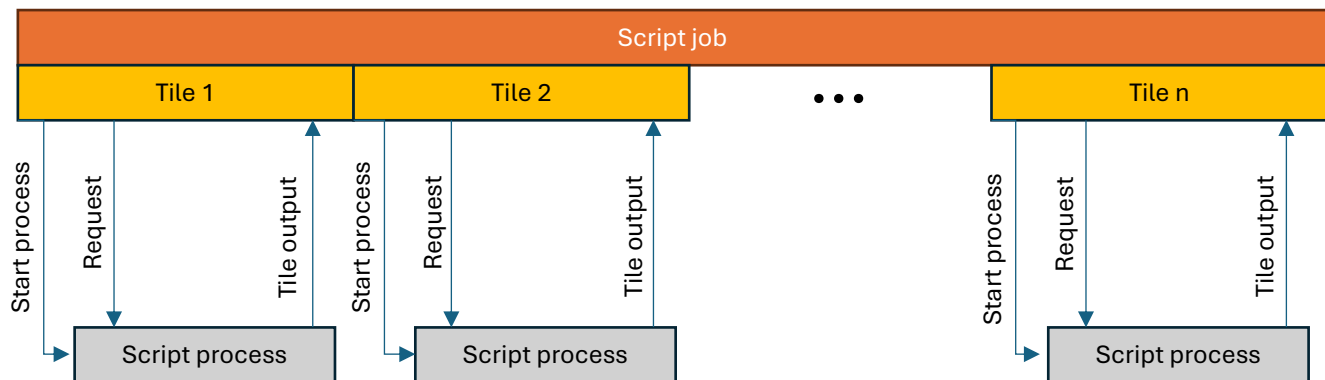
Scripting can be run even if there are no acquired tiles in the tile set. Obviously, the request sent to the script will not contain any tiles.

## Calling scripts, single and batch processing

There is always on job for one tile set and script. The difference is in number of calls to the scripting process.

### Single processing (default)

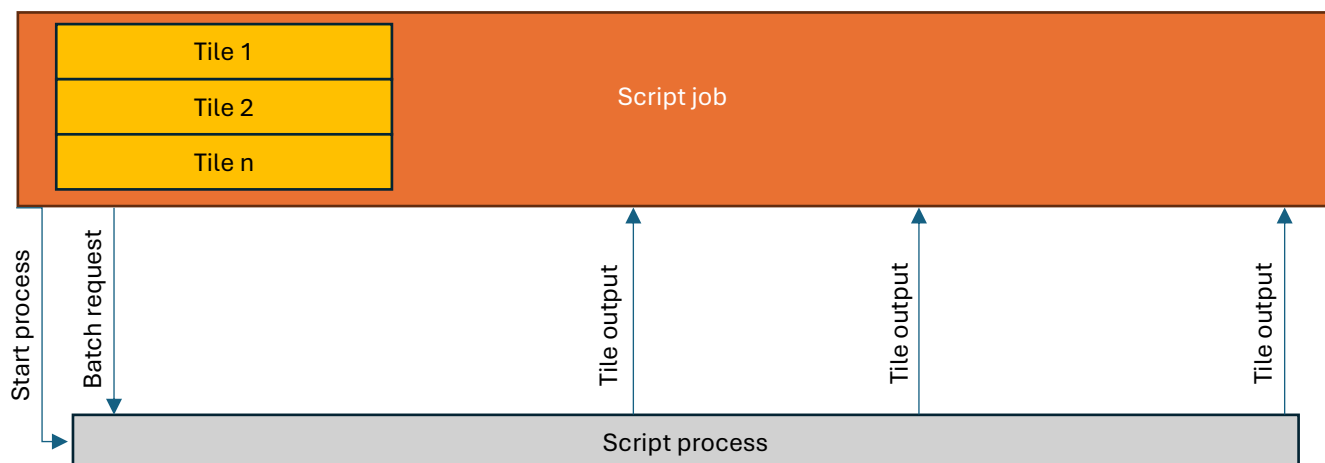
The scripting process is started for each tile. Maps sends a request with individual tile to the process. The script is expected to finish after the tile processing is completed.



### Batch processing

The scripting process is started only at the start of the script job. Maps sends request containing all tiles in an array. The script is expected to finish when all tiles are processed. The batch processing can be useful for broader analysis of the tile set as whole. It is however more complex from the script perspective, as the script must iterate over the tile array and takes responsibility for all progress updates, processing stop, etc. The batch processing can also save some time as the overhead for starting and stopping a process is done only once. However, on the other hand, the process can be more demanding on other resources and it can breach limits given by CSA sandboxing.

Batch processing is not available for run during acquisition strategies for obvious reason – during acquisition, tiles are being acquired one by one and cannot be sent as a batch for scripting.



## Stop on error

**Not implemented yet** – details may differ when available. We may for example need to differentiate between unexpected exceptions and script related errors.

Use **Stop on error** parameter to control how errors are handled.

By default the script processing stops when an error is encountered. These include any errors related to starting a scripting process and communication with the process, errors with the script itself, unexpected or invalid JSON structures and values, as well as errors explicitly reported by the script.

In some cases, it may be beneficial to ignore errors and continue with the processing, although some tile(s) or request(s) may not be fully completed. This is particularly important for *Run during acquisition* strategy, where the scripting is a part of the acquisition process and any error therefore stops not only the script processing, but also the whole acquisition process.

If a script processing fails due to an error, the script job remains in the processing job control (and queue) and displays the error state with error message “Processing failed. See logs for details.” The failed jobs stays in the queue until it is deleted. **This is a limitation of the job control in Maps core. I’d rather see failed jobs in “completed” state, but it is just not possible without some core refactoring. Therefore users must manually delete failed jobs. The jobs however do not block the processing queue or further scripting of the tile set.**

## Source and Target tile sets

**Source tile set** is always the tile set for which the script is setup and run.

**Target output tile set** is a tile set to which all script outputs are sent and stored. If not explicitly specified in scripts (by GUID), the source tile set is also the target tile set.

Target tile sets can be created explicitly by scripts. See description of GetOrCreateOutputTileSet response below. Rules for this creation/reuse are:

- If the target tile set name is same as the source tile set, then the source tile set is also the target tile set.
- If a tile set with the same name already exists, then it is reused and set as the target tile set if it has been created with the SBP. Otherwise a new tile set is created and set as the target tile set. This new tile set name will be postfixed with (N).

**Note about output tile sets:** Output tile set are meant only as a target for images generated by scripts, they **are not meant to be acquired**, they are not queued in the acquisition queue, they cannot be reset, etc. They can be stitched however.

## Script default parameters

When a script is selected in the tile set Script Bridge setup tab, then default parameters are applied if they are available.

The default parameters can be specified in scripts. When Maps probes the script folder, it also parses script content and looks for the specification. The specification is JSON struct in a comment between `# Default parameters` and `# Default parameters end` delimiters. This section can be anywhere in the script file, but specify it on its beginning (as it makes the parsing faster).

All parameters can be specified in the default parameters. JSON is as follows:

```
{
    "RunMode" : str,
    "ScriptMode" : str,
    "ScriptParameters" : str,
    "PrepareImages" : bool
    "StopOnError" : bool,
}
```

**RunMode** – Any of *manual*, *whencompleted*, *live*, *liveasync*.

**ScriptMode** – *batch* or *singletiles*

**ScriptParameters** – Additional script parameters, string stored as is.

**PrepareImages** – *true* to prepare (export) pyramid images. For image layer only (otherwise ignored)

**StopOnError** – *true* to stop on error, *false* otherwise.

Any of the parameters are optional, can be omitted.

Example:

```
# Default parameters
#{
#"RunMode" : "manual",
#"ScriptMode" : "batch",
#"PrepareImages" : true,
#"ScriptParameters" : "threshold=120;maximum=255",
#"StopOnError" : true
#}
# Default parameters end
```

This is another part where Maps presume that script files exist and can be read and parsed. For other processing systems than Python, the default parameters would have to be implemented differently. This was also implemented for the removed remote mode.

## Maps – script process communication

Maps communicates with the script process via the standard input and output with JSON format. Maps sends *requests* to the process standard input, while the script sent *responses* to the standard output.

*This is only wording, if confusing, we can rename requests and responses to something more intuitive.*

Maps reads all strings from the output. Strings not recognized as JSON are logged to the script logs, which can be useful for debugging and (later) analysis of the scripting process:

`print("Something to be read and logged")` – will be read and logged, otherwise ignored, do not start with `{` – Maps would recognize it as JSON and very probably fail with unrecognized structure

To simplify the communication, all JSON must be sent in a single line, i.e. without indents or newlines. In Python this can be done for example with:

```
print(json.dumps(jsonResponse), flush=True)
```

Every script is provided with a request at its start and it is to read it from the standard input:

```
json_input = sys.stdin.readline()
jsonRequest = json.loads(json_input)
```

The Maps requests are described below.

## MapsBridge Python library

To simplify writing Python scripts, MapsBridge library is provided. It mostly simply wraps JSON creation and sending.

The library is provided as MapsBridge.py and can be imported to scripts as is or users can modify it according to their needs.

*MapsBridge.py is a living part of the implementation, I expect more changes there, including adding comments.*

*There is a question whether we want to distribute it to end users and how. I would find it useful for them (when it is finalized and ready).*

## Maps tile set request

The Maps tile set request is sent to the script process standard input at the very beginning of the script processing of a tile set. Its purpose is to request an action from the script and to provide information about the source tile set and tiles to be processed.

JSON request type **TileSetRequest**:

```
{
  "RequestType": "TileSetRequest",
  "ScriptName": str,
  "ScriptParameters": str,
  "SourceTileSet": TileSetInfo
  "TilesToProcess":
    [
      {
        "Column": int,
        "Row": int
      },
      ...
    ]
}
```

**ScriptName** – Name of the executed script.

**ScriptParameters** – Any string as a parameter specified for the script.

**SourceTileSet** – Information of the source tile set, see the description of TileSetInfo below

**TilesToProcess** – list of tiles to be processed by the script, every entry consists of 1-based **Column** and **Row** indices  
In the current implementation. The list will contain either one tile (for Single tile script mode) or it will be empty or missing (for Batch script mode – obviously, because all tiles in the tile set can be processed)

### In MapsBridge

```
request = MapsBridge.ScriptTileSetRequest.FromStdIn()      # Returns ScriptTileSetRequest
```

## Maps image layer request

The Maps image layer request is sent to the script process standard input at the very beginning of the script processing of an image layer. Its purpose is to request an action from the script and to provide information about the source image layer.

JSON request type **ImageLayerRequest**:

```
{
  "RequestType": "ImageLayerRequest",
  "ScriptName": str,
  "ScriptParameters": str,
  "SourceImageLayer": ImageLayerInfo
  "SourceTileSet": TileSetInfo
  "PreparedImages": { "channelIndex": "fullpath"}
}
```

**ScriptName** – Name of the executed script.

**ScriptParameters** – Any string as a parameter specified for the script.

**SourceImageLayer** – information of the image layer, see description of ImageLayerInfo below

**PreparedImages** – channel index (as string) and the full path to the prepared exported pyramid image (if available)

**This will change to a dictionary of images organized by channel infos when we support export of multiple channels**

### In Maps bridge

```
request = MapsBridge.ScriptImageLayerRequest.FromStdIn()    # returns ScriptImageLayerRequest
```

## Maps stop request

The request is sent to the script process when the processing should be stopped.

The request JSON:

```
{
    "Request" : "Stop"
}
```

The script process is expected to stop the processing upon the request and exit. Maps will wait for the process to exit. The process will be killed if it not exists within some time. Currently the timeout is set to 20 sec.

### In MapsBridge

Currently not implemented in MapsBridge.

Note: In demo.py and MapsBridge.py catching this request is not implemented. Stopping will always wait for the scripts to complete or kills them eventually. It can also clash with MapsBridge.CreateTileSet() and in such case the script fails. While this is the responsibility of scripts, I'll make the mechanism more sturdy in Maps too.

## Maps exit request

The request is sent to the script process when the processing should terminate.

The request JSON:

```
{
    "Request" : "Exit"
}
```

Similar to Stop request. In this case it is not expected that the scripting will continue (mostly because some Maps internal error). The script process is expected to gratefully terminate the processing and exit upon the request. Maps will wait for the process to exit. The process will be killed if it not exists within some time. Currently the timeout is set to 20 sec.

### In MapsBridge

Currently not implemented in MapsBridge.

## Maps tile set description – TileSetInfo

Describes a Maps tile set wherever it is needed in Maps requests.

JSON structure:

```
{
  "Name": str,
  "Guid": str,
  "ColumnCount": int,
  "RowCount": int,
  "ChannelCount": int,
  "IsCompleted": bool,
  "DataFolderPath": str,
  "PixelFormat": str,
  "Size":
  {
    "Width": float,
    "Height": float
  },
  "StagePosition":
  {
    "X": float,
    "Y": float
  },
  "Rotation": float,
  "TileSize":
  {
    "Width": float,
    "Height": float
  },
  "TileResolution":
  {
    "Width": int,
    "Height": int
  },
  "PixelToStageMatrix": MatrixInfo,
  "Channels":
  [
    {
      "Index": int,
      "Name": str,
      "Color": str
    },
    . . .
  ]
  "Tiles":
  [
    {
      "Column": int,
      "Row": int,
      "StagePosition":
      {
        "X": float,
        "Y": float
      },
      "TileCenterPixelOffset":
      {
        "X": int,
        "Y": int
      },
      "ImageFileNames": { "channelIndex": str }
    },
    . . .
  ]
}
```

**Name** – Name of the tile set.

**Guid** – GUID of the tile set.

**DataFolderPath** – Full path to the folder with tile set data. Base for all tile files.

**ColumnCount** – Number of columns in the source tile set.

**RowCount** – Number of rows in the source tile set.

**ChannelCount** – Number of channels in the source tile set.

**IsCompleted** – True if the tile set acquisition is completed.

**StagePosition** – Position of the tile set center on the stage (X, Y) in meters.

**Rotation** – The tile set rotation around its center in degrees. In most platforms this value is the scan rotation angle.

**Size** – Total physical size of the tile set (Width and Height) in meters.

**TileSize** – Physical size of one tile (Width and Height) in meters.

**TileResolution** – Resolution (Width and Height) of a single tile.

**PixelFormat** – Name of the pixel format of the tile set. It is the string representation of `.NET System.Windows.Media.PixelFormat`.

**PixelToStageMatrix** – Pixel to stage transformation matrix. See description below.

**Channels** – Array of ChannelCount length channels where each JSON entry:

**Index** – Index of the channel in the tileset

**Name** – Name

**Color** – Color hex notation (e.g. `#DD00FF`)

**Tiles** – **Array of all (so far) acquired tiles**. Every entry is JSON with parameters:

**Column** – Tile column, 1-based index.

**Row** – Tile row, 1-based index.

**StagePosition** – stage position of the tile center (X and Y) in meters.

**TileCenterPixelOffset** – offset (X, Y) in pixels from the tile set center

**ImageFileNames** – Dictionary of available image files organized by channel index, every entry is *channelIndex:filename*. Combine filename with **DataFolderPath** to get the full path. Note that some channels may be missing in the dictionary if there is no images available for them.

## In MapsBridge

Represented with `TileSetInfo` class.

## Maps image layer description – ImageLayerInfo

Describes a Maps image layer wherever it is needed in Maps requests.

JSON structure:

```
{
  "Name": str,
  "Guid": str,
  "ChannelCount": int,
  "DataFolderPath": str,
  "Size":
  {
    "Width": float,
    "Height": float
  },
  "StagePosition":
  {
    "X": float,
    "Y": float
  },
  "Rotation": float,
  "TotalLayerResolution":
  {
    "Width": int,
    "Height": int
  },
  "PixelToStageMatrix": MatrixInfo,
  "Channels":
  [
    {
      "Index": int,
      "Name": str,
      "Color": str
    }
  ],
  "OriginalTileSet": TileSetInfo
}
```

**Name** – Name of the image layer.

**Guid** – GUID of the image layer.

**ChannelCount** – Number of channels in the image layer.

**DataFolderPath** – Full path to the folder with pyramid data.

**StagePosition** – Position of the layer center on the stage (X , Y) in meters.

**Rotation** – The layer rotation around its center in degrees.

**Size** – Total physical size of the layer (Width, Height) in meters.

**TotalLayerResolution** – Resolution of the whole layer (Width, Height).

**PixelFormat** – Name of the pixel format of the tile set. It is the string representation of `.NET System.Windows.Media.PixelFormat`. – missing in this version, will be added.

**PixelToStageMatrix** – Pixel to stage transformation matrix. See description below.

**Channels** – Array of ChannelCount length channels where each JSON entry:

**Index** – Index of the channel in the tileset

**Name** – Name

**Color** – Color hex notation (e.g. `#DD00FF`)

**OriginalTileSet** – Description of the original tile set if the image layer is a result of the tile set stitching. This property will be empty or missing if no such tile set exists in the project.

### In MapsBridge

Represented with ImageLayerInfo class.

## Pixel to stage matrix usage

**PixelToStageMatrix** structure is an array of the matrix members:

```
[
    [a11, a12, a13],
    [a21, a22, a23],
    [a31, a32, a33]
]
```

This array can be for example read by numpy as `np.array(PixelToStageMatrix, dtype=float)`

### Example:

```
"PixelToStageMatrix":
[
    [2.9296874999999997E-07, 0, 0],
    [0, 2.9296874999999997E-07, 0],
    [-0.012195525216850297, 0.0035056840776182996, 1]
]
```

This matrix is a transformation from pixel coordinates in the image to an absolute stage coordinates. Use it to determine the stage position from tile or image layer image.

### In case of a tile images:

The matrix is always related to the whole tile set center. To calculate absolute pixel offset from the center for a tile image, use **TileCenterPixelOffset** of the tile:

```
# Load the matrix
np.array(PixelToStageMatrix, dtype=float)
# Calculate the absolute X pixel offset
pixelX = TileCenterPixelOffset.X - (tileResolution.Width / 2) + imagePixelX

# Calculate the absolute Y pixel coord
# Note the opposite operators here! Y axis on the stage is opposite to pixel coords
pixelY = - TileCenterPixelOffset.Y + (tileResolution.Height / 2) - imagePixelY

stagePosition = p @ M      # The resulting absolute position on the stage in meters
```

### In MapsBridge:

```
TilePixelToStage(tileImagePixelX: int, tileImagePixelY: int, tileColumn: int, tileRow: int,
tileSet: TileSetInfo) -> PointF
```

### In case of image layers:

Image layers represent a single image, refer to the image center:

```
# Load the matrix
np.array(PixelToStageMatrix, dtype=float)
# Calculate the absolute X pixel coord
pixelX = - (totalImageResoulution.Width / 2) + imagePixelX

# Calculate the absolute Y pixel coord
# Note the opposite operators here! Y axis on the stage is opposite to pixel coords
pixelY = (totalImageResoulution.Height / 2) - imagePixelY

stagePosition = p @ M      # The resulting absolute position on the stage in meters:
```

### In MapsBridge:

```
ImagePixelToStage(imagePixelX: int, imagePixelY: int, imageLayer: ImageLayerInfo) -> PointF
```

## Script responses in general

As stated above, scripts are expected to send responses to the standard output. Any response from the script must be identified with the response type, list of currently supported responses and their descriptions are below. Responses with an unknown script type are treated as errors.

List of currently supported responses and their descriptions are below.

## Tile Output

JSON response type **TileOutput**

```
{
  "ResponseType": "TileOutput",
  "Column": int,
  "Row": int,
  "ImageFileOutputs":
  [
    {
      "TargetTileSetGuid": str,
      "TargetChannelName": str,
      "ImageFileName": str,
      "OutputLocationPath": str,
      "KeepFile": bool
    },
    . . .
  ]
}
```

**Column** – Tile column, 1-based index.

**Row** – Tile row, 1-based index.

**ImageFileOutputs** – Array of image outputs to be sent to a tile set.

Every image file entry is a JSON with parameters:

**TargetTileSetGuid** – Optional. GUID of the target tile set in registry format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}".

**TargetChannelName** – Name of the target channel.

**ImageFilePath** – Full path to the image file to be sent.

**KeepFile** – Optional. If *False* then the image file is deleted from the output folder after it is sent to Maps. Use *True* to keep the original file . Default value is *False*.

## Description

If TargetTileSetGuid is specified, a tile set with the GUID must exists in the project, otherwise an error is reported. If omitted or None, the target script tile set is used.

If a channel with TargetChannelName does not exists in the target tile set, the channel is created. The existing channel must have been created with SBP, otherwise an error is reported – Maps does not allow scripts to alter channels from other sources.

Note, that folders with write permissions will be restricted for scripts when CRA is implemented. Maps will not permit to move files from other (read-only) folders, in such cases KeepFile will be always considered *True* and the file will be copied.

## Important

- The image file must match the target tile set pixel format and resolution. Maps will convert and resize the image accordingly. Currently supported pixel formats in Maps are Gray8 and Gray16.
- Image files are renamed internally during import to match Maps filename patterns.

## Example

```
{
  "ResponseType": "TileOutput",
  "Column": 1,
  "Row": 1,
  "ImageFileOutputs":
  [
    {
      "TargetTileSetGuid": "{61E76B4A-4D1D-4B78-BC9C-DAB82D0374B1}",
      "TargetChannelName": "Filtered",
      "ImageFilePath": " C:\\Temp\\filteredresults\\Tile_001-001_filtered.tif",
      "KeepFile": true
    },
    {
      "TargetChannelName": "Particles",
      "ImageFilePath": " C:\\Temp\\particlereults\\Tile_001-001_particles.tif",
      "KeepFile": false
    }
  ]
}
```

## In MapsBridge

There is a method for sending a single image file to Maps:

```
SendSingleTileOutput(
  tileRow: int,
  tileColumn: int,
  targetChannelName: str,
  imageFileNamePath: str,
  keepFile: Optional[bool] = False,
  targetTileSetGuid: Optional[uuid] = None)
```

## Create tile set

Creates a new tile set and prepares it for acquisition.

JSON response type **CreateTileSet**:

```
{
  "ResponseType": "CreateTileSet",
  "TileSetName": str,
  "TargetLayerGroupName" : str,
  "TotalSize": [float/str, float/str],
  "StagePosition": [float/str, float/str, float/str],
  "Rotation": float/str,
  "TemplateName": str,
  "TileHfw": float/str,
  "TileResolution": [int, int],
  "PixelSize": float/str
  "ScheduleAcquisition" : bool
}
```

**TileSetName** – Desired tile set name

**TargetLayerGroupName** – Optional. Name of a layer group in which the tile set should be created.

**TotalSize** – Total size of the tile set. Values are Width, Height in meters.\*

**StagePosition** – Stage position of the tile set center. Values are: X in meters, Y in meters, StageR in degrees.\*

**Rotation** – Optional. Rotation around the center in degrees (clock-wise), the default value is 0.\*

**TemplateName** – Optional. Name of the tile set template to be used for creation.

**TileHfw** – Horizontal field of view of one tile in the tile set in meters.\*

**TileResolution** – Optional. [width, height] specification of resolution of one tile in the tile set.

**PixelSize** – Optional. Size of one pixel in meters.\*

**ScheduleAcquisition** – Optional. True to schedule the tile set acquisition. Default value is *True*.

**\*]** All positions, lengths and angles can be also specified with a string representation of “*value unit*”.

Position (length) properties accept *m*, *mm*, *um*, *μm*, *nm* units, for example “1.3 um”.

Angle properties accept *r*, *rad*, *deg*, ° units, for example “62 °”

## Description

The tile set is always create for the current active platform to which Maps is connected. It may be SEM, TEM, Phenom. It is not possible to create a tile set when Maps is completely offline.

This response always creates a new tile set. If a tile set of the same already exists in the project, the newly created tile set name is postfixed with (N). If TargetLayerGroupName is specified, the tile set is added to that layer group. The group is created if it does not exist.

The tile set is always created using a template. It either a default (or currently selected) template in Maps, or a template specified with TemplateName. If specified, the template must exists.

Specify TileHfw to override HFW set from the template.

Specify either TileResolution or PixelSize to calculate and override resolution of one tile set from the template. Only one of the parameters may be specified. Note that on some microscopes it is possible to set only specific supported resolutions. If the new resolution is not among the list of the available resolutions, a nearest resolution is used.

If the tile set should be ready for acquisition, set ScheduleAcquisition to *True*. An acquisition job will be queued and scheduled. The tile set acquired when the queue goes to the job.

## Example

```
{
  "ResponseType": "CreateTileSet",
  "TileSetName": "Particle A",
  "StagePosition": [-0.013943306762449606, 0.009816733067729087, "30 deg"],
  "TotalSize": ["30um", "20um"],
  "TemplateName": "Detailed 15kV 1us",
  "TileHfw": "5um",
  "PixelSize": "3nm"
}
```

## In MapsBridge

```
CreateTileSet(
  tileSetName: str,
  stagePosition: Tuple[str, str, str],
  totalSize: Tuple[str, str],
  rotation: Optional[str] = None,
  templateName: Optional[str] = None,
  tileResolution: Optional[Tuple[int, int]] = None,
  tileHfw : Optional[str] = None,
  pixelSize: Optional[str] = None,
  scheduleAcquisition: Optional[bool] = None,
  targetLayerGroupName: Optional[str] = None
) -> TileSetCreateInfo:
```

## Maps response

When the tile creation request is processed, Maps sends a JSON response with the created tile set info to the script process standard input:

```
{
  "Info": "TileSetCreateInfo",
  "IsSuccess" : bool,
  "ErrorMessage" : str,
  "IsCreated" : bool,
  "TileSet" : TileSetInfo
}
```

**IsSuccess** – True if the tile set successfully created or found, False otherwise (Name and Guid will be empty)

**ErrorMessage** – Error message in case of IsSuccess = true

**Name** – Actual name of the tile set.

**Guid** – GUID of the tile set in registry format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}".

**IsCreated** – True if newly created, False if an existing tile set reuse.

**TileSet** – TileSetInfo of the tile set (see TileSetInfo description)

Scripts should read this info to get the tile set actual name and GUID.

Note: While a tile set name may change in Maps (for example renamed by users), the tile set GUID is fixed – use it to identify the specific tile set in scripts.

## In MapsBridge

The response is represented with TileSetCreateInfo class.

## Create output tile set

Creates or gets a target tile set for script outputs. The tile set cannot be acquired, it only contains data set from a script.

JSON response type **GetOrCreateOutputTileSet**:

```
{
  "ResponseType": "GetOrCreateOutputTileSet",
  "TileSetName": str,
  "Resolution": [int, int],
  "TargetLayerGroupName" : str,
}
```

**TileSetName** – Desired tile set name

**Resolution** – Optional. [width, height] specification of resolution of one tile in the tile set

**TargetLayerGroupName** – Optional. Name of a layer group in which the tile set should be created.

This response creates a new tile set which can be used as the target tile set for script outputs.

Same rules for the creation are applied as for the default target tile sets (see above).

### Example

```
{
  "ResponseType": " GetOrCreateOutputTileSet ",
  "TileSetName": "Blue copy",
  "Resolution": [2048, 1024]
}
```

### In MapsBridge

```
GetOrCreateOutputTileSet (
  tileSetName: Optional[str] = None,
  tileResolution: Optional[Tuple[int, int]] = None,
  targetLayerGroupName: Optional[str] = None
) -> TileSetCreateInfo
```

### Maps response

Maps sends the same response as in Create tile set.

## Create or alter channel

Creates or alter a channel of a tile set.

JSON response type **CreateChannel**:

```
{
  "ResponseType": "CreateChannel",
  "TargetChannelName": str,
  "TargetTileSetGuid": str,
  "ChannelColor": str,
  "IsAdditive": false
}
```

**TargetTileSetGuid** – Optional. GUID of the target tile set in registry format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}".

**TargetChannelName** – Name of the target channel.

**ChannelColor** – Optional. In hex notation (e.g. #DD00FF), the default value is #FFFFFF (white).

**IsAdditive** – Optional, the default value is *False*.

### Description

If TargetTileSetGuid is specified, a tile set with the GUID must exists in the project, otherwise an error is reported. If omitted or None, the target script tile set is used.

If a channel with TargetChannelName does not exists in the target tile set, the channel is created. If such channel exists, its properties are altered (color, additivity). The existing channel must have been created with SBP, otherwise an error is reported – Maps does not allow scripts to alter channels from other sources.

### Example

```
{
  "ResponseType": "CreateChannel",
  "TargetTileSetGuid": "{61E76B4A-4D1D-4B78-BC9C-DAB82D0374B1}",
  "TargetChannelName": "HighlightGreen",
  "ChannelColor": "#00FF00",
  "IsAdditive": true
}
```

### In MapsBridge

```
CreateChannel(
  channelName: str,
  channelColor: Optional[Tuple[int, int, int]] = (255, 255, 255),
  isAdditive: Optional[bool] = False,
  targetTileSetGuid: Optional[uuid] = None)
```

## Create image layer

Creates an image layer. The layer will be based on an import of specified image.

JSON response type CreateImageLayer:

```
{
  "ResponseType": "CreateImageLayer",
  "LayerName": str,
  "StagePosition": [float/str, float/str, float/str],
  "TotalSize": [float/str, float/str],
  "Rotation": float/str,
  "ImageFilePath": str,
  "TargetLayerGroupName": str,
  "KeepFile": bool
}
```

**LayerName** – Desired layer name

**TargetLayerGroupName** – Optional. Name of a layer group in which the layer should be created.

**TotalSize** – Optional. Total size of the layers. Values are Width, Height in meters.\*

**StagePosition** – Optional. Stage position of layer center. Values are: X in meters, Y in meters, StageR in degrees.\*

**Rotation** – Optional. Rotation around the layer center in degrees (clock-wise), the default value is 0.\*

**ImageFilePath** – Full path to the image to import.

**KeepFile** – *True* to keep the imported image, *False* to delete it after the import.

**\*]** All positions, lengths and angles can be also specified with a string representation of “*value unit*”.

Position (length) properties accept *m*, *mm*, *um*, *μm*, *nm* units, for example “1.3 um”.

Angle properties accept *r*, *rad*, *deg*, ° units, for example “62 °”

### Further description

If StagePosition and TotalSize are not specified (null or missing), the imported image will be positioned and sized to exactly match the source layer (be it a tile set or an image layer). If you want to overlay the new layer over the source layer, just do not specify the parameters.

StagePosition, TotalSize and Rotation allow to import image of any size to any location. All three properties must be specified. Any missing property will be treated as an error.

The imported image may be of any size (large images must be tiffs). However, expect that import of such images may take minutes or tens of minutes to import.

And the process cannot be cancelled/stopped in Maps – we will try to fix this. Now, users must wait until the image is imported.

### Example

Message (formatted JSON):

```
{
  "ResponseType": "CreateImageLayer",
  "LayerName": "Viridis - Tile Set (stitched) (2)",
  "ImageFilePath": "C:\\Users\\vymola\\AppData\\Local\\Temp\\false_color_output\\0_color.png",
  "TargetLayerGroupName": "Outputs",
  "KeepFile": true
}
```

## In MapsBridge

```
CreateImageLayer(  
    layerName : str,  
    imageFilePath: str,  
    stagePosition: Optional[Tuple[str, str, str]] = None,  
    totalSize: Optional[Tuple[str, str]] = None,  
    rotation: Optional[str] = None,  
    targetLayerGroupName: Optional[str] = None,  
    keepFile: Optional[bool] = False  
)
```

## Create annotation

Creates an area of interest (AOI) or site of interest (SOI).

JSON response type **CreateAnnotation**:

```
{
  "ResponseType": "CreateAnnotation",
  "AnnotationName": str,
  "StagePosition": [float/str, float/str, float/str],
  "Size": [float/str, float/str],
  "Rotation": float/str,
  "Notes": str,
  "Color": str,
  "IsEllipse": bool,
  "TargetLayerGroupName": str
}
```

**AnnotationName** – Name of the annotation.

**StagePosition** – Stage position of the center. Values are: X in meters, Y in meters, StageR in degrees.\*

**Size** – Optional. [x,y] size in meters, the default value is *None*.\*

**Rotation** – Optional. Rotation around the center in degrees (clock-wise), the default value is 0.\*

**Notes** – Optional. Notes for the annotation.

**Color** – Optional. Color in hex notation (e.g. *#DD00FF*), the default value is *#FFFFFF* (white).

**IsEllipse** – Optional. true if AOI should be an ellipse, the default value is *False*.

**TargetLayerGroupName** – Optional. Name of a layer group in which the annotation should be created.

**\*]** All positions and angles can be also specified with a string representation of “*value unit*”.

Position (length) properties accept *m*, *mm*, *um*, *μm*, *nm* units, for example “1.3 um”.

Angle properties accept *r*, *rad*, *deg*, ° units, for example “62 °”

### Example

```
{
  "ResponseType": "CreateAnnotation",
  "AnnotationName": "Sample AOI",
  "StagePosition": ["1 mm", -0.0002, 0],
  "Size": [0.0001, 0.0005],
  "Rotation": "30 deg",
  "Notes": "Some notes",
  "Color": "#00FF00",
  "IsEllipse": false
}
```

### Description

If an annotation with AnnotationName already exists, a new one with (N) postfix is created.

If Size is not specified, SOI is created. If specified AOI is created.

Rotation, IsEllipse, Color are meaningful for AOI only (script will not fail if specified for SOI though)

### In MapsBridge

```
CreateAnnotation(
  annotationName: str,
  stagePosition: Tuple[str, str, str],    rotation: Optional[float] = 0,
  size: Optional[Tuple[str, str]] = None,
  notes: Optional[str] = "",
  color: Optional[Tuple[int, int, int]] = None,
  isEllipse: Optional[bool] = False
)
```



## Store file to tile set

Stores a file to the target layer.

JSON response type **StoreFile**:

```
{
    "ResponseType" : "StoreFile",
    "TargetLayerGuid": str,
    "FilePath" : str,
    "Overwrite" : bool,
    "KeepFile" : bool
}
```

**TargetLayerGuid** – Optional. GUID of the target layer in registry format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}".

**FilePath** - Full path to a file to be stored.

**Overwrite** – Optional. If *True* then any existing file of the same name will be overwritten. Otherwise the stored file will be renamed with an (N) postfix (like in the file explorer). Default value is *False*.

**KeepFile** – Optional. If *True* then the file is copied from the specified rather than moved, keeping the original file . Default value is *False*.

### Description

If TargetLayerGuid is specified, a layer with the GUID must exists in the project, otherwise an error is reported. If omitted or *None*, the default target script layer is used.

Note that folders with write permission will be restricted for scripts when CRA is implemented. Maps will not permit to move files from other (read-only) folders, in such cases KeepFile will be always considered *True* and the file will be copied.

### Example

```
{
    "ResponseType": "StoreFile",
    "TargetLayerGuid": "{61E76B4A-4D1D-4B78-BC9C-DAB82D0374B1}",
    "FilePath": "C:\\AnalysisOutputs\\Some PPT.pptx",
    "Overwrite": true,
    "KeepFile": true
}
```

### In MapsBridge

```
StoreFile(
    filePath: str,
    overwrite: Optional[bool] = False,
    keepFile: Optional[bool] = True,
    targetLayerGuid: Optional[uuid] = None)
```

## Append notes to tile set

Append notes to the target layer.

JSON response type **AppendNotes** :

```
{
    "ResponseType" : "AppendNotes",
    "TargetLayerGuid": str,
    "NotesToAppend" : str
}
```

**TargetLayerGuid** – Optional. GUID of the target layer in registry format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}".

**NotesToAppend** – Any string to be append to notes.

### Description

If TargetLayerGuid is specified, a Maps layer with the GUID must exists in the project, otherwise an error is reported. If omitted or *None*, the default target script layer set is used.

Notes are always only appended – scripts are not allowed to completely overwrite or delete already existing notes, as they may have been added from different sources or edited by users.

### Example

```
{
    "ResponseType": "AppendNotes",
    "TargetLayerGuid": "{61E76B4A-4D1D-4B78-BC9C-DAB82D0374B1}",
    "NotesToAppend": "Tile [1, 5] processed with a clever script \r\n"
}
```

### In MapsBridge

```
AppendNotes(
    notesToAppend: str,
    targetLayerGuid: Optional[uuid] = None)
```

## Log information

Allows logging of information to the script logs.

JSON response type **Log** :

```
{
  "ResponseType": "Log",
  "LogInfoMessage": str,
  "LogWarningMessage": str,
  "LogErrorMessage" : str
}
```

Any of **LogInfoMessage**, **LogWarningMessage**, and **LogErrorMessage** can be specified and will be logged. Note, that this is really logging only, sending logs has no effect on the script workflow or result.

### In MapsBridge

```
LogInfo(infoMessage: str)
LogWarning(warningMessage: str)
LogError(errorMessage: str)
```

## Report failure

Allow reporting a failure.

JSON response type **ReportFailure** :

```
{
  "ResponseType": "ReportFailure",
  "ErrorMessage": str
}
```

**ErrorMessage** – Any error message to be reported.

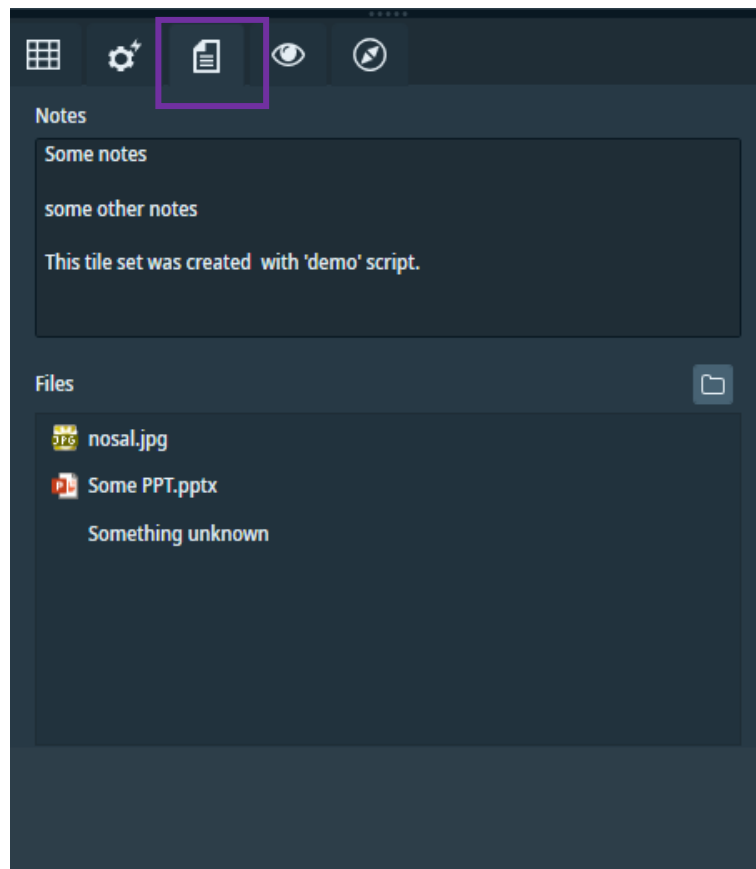
Report failure will be treated as a script error. The processing of the whole script or the current request will be terminated (depending on Stop On Error, see above). The error will be logged and reported to users.

### In MapsBridge

```
ReportFailure(errorMessage: str)
```

## Layer metadata tab

Newly introduced is the Metadata tab for layers. This is not a part of the plugin, it is implemented in Maps core and the tab as well as files and notes are available for all other plugins even if the SBP is not enabled or present at all.



### Notes

Notes related to the layer. They can be filled from the plugin (~ script) or edited by users.

### Files

Any additional files related to the layer. The folder where files are stored is:

`<project path>MetaData\<layer GUID>\StoredData`

Click to Folder button to directly open the folder in File Explorer. The button action also creates the folder if it does not exist yet – this allows users to add something manually.

List of files is automatically synchronized with the folder content – even if files there are copied, deleted or renamed manually by users. [We can add details to the list later, like datetime of creation, size...](#)

Double click a file to open it with a system default executable. If such executable is not available or specified, the standard shell prompt to select one is displayed.

If the layer is exported, both notes and files are exported with the layer.

## Examples

Several examples of requests and responses.

### TileSetInfo

```
{
  "Name": "Tile Set",
  "Guid": "{8835600A-2793-489C-814B-028C22CB2CDB}",
  "ColumnCount": 2,
  "RowCount": 2,
  "ChannelCount": 1,
  "IsCompleted": true,
  "DataFolderPath": "C:\\\\Work\\\\Maps data 3.35\\\\sdd\\\\LayersData\\\\Layer\\\\Tile Set",
  "PixelFormat": "Gray8",
  "Size":
  {
    "Width": 0.00114,
    "Height": 0.000984140625
  },
  "StagePosition":
  {
    "X": -0.012195525216850297,
    "Y": 0.0035056840776182996
  },
  "Rotation": 0,
  "TileSize":
  {
    "Width": 0.0006,
    "Height": 0.00051796875
  },
  "TileResolution":
  {
    "Width": 2048,
    "Height": 1768
  },
  "PixelToStageMatrix":
  [
    [2.9296875, 0, 0],
    [0, 2.9296875E-07, 0],
    [-0.012195525216850297, 0.0035056840776182996, 1]
  ],
  "Channels":
  [
    {
      "Index": 0,
      "Name": "BSD; None",
      "Color": "#FFFFFF"
    }
  ]
  "Tiles":
  [
    {
      "Column": 1,
      "Row": 1,
      "StagePosition":
      {
        "X": -0.012465525216850296,
        "Y": 0.0037387700151182996
      },
      "TileCenterPixelOffset":
      {
        "X": -921,
        "Y": -795
      },
      "ImageFileNames": { "0": "Tile_001-001-000000_0-000.s0001_e00.tif" }
    },
    . . .
  ]
}
```

## ImageLayerInfo

```
{
  "Name": "Tile Set (stitched) (2)",
  "Guid": "{3026755E-B2C4-4488-927F-7F7746E9BB87}",
  "ChannelCount": 1,
  "DataFolderPath": "C:\\\\Work\\\\Maps_data_3.35\\\\sdd\\\\LayersData\\\\Layer\\\\Tile Set (stitched) (2)",
  "Size":
  {
    "Width": 0.001090723,
    "Height": 0.0009618165
  },
  "StagePosition":
  {
    "X": -0.011208538446274134,
    "Y": 0.0031560017700832184
  },
  "Rotation": 0,
  "TotalLayerResolution":
  {
    "Width": 3723,
    "Height": 3283
  },
  "PixelToStageMatrix":
  [
    [2.9296875, 0, 0],
    [0, 2.929685, 0],
    [-0.012195525216850297, 0.0035056840776182996, 1]
  ],
  "Channels":
  [
    {
      "Index": 0,
      "Name": "BSD; None",
      "Color": "#FFFFFF"
    }
  ],
  "OriginalTileSet": null,
}
```