Maps Script Bridge

This document provides an overview of the **Maps Script Bridge** plugin, detailing its functionality, usage, and a complete **API reference** for integrating and managing custom scripts within Maps.

Document version: 1.0.0
Author: Thermo Fisher Scientific

Script Bridge plugin introduction

Script Bridge is a Maps plugin available in Maps with a correlative workflow license.

Script Bridge allows Maps to execute external custom scripts and enables two-way communication between Maps and the running scripts. Currently, Python scripts are supported.

Using Script Bridge, scripts can:

- Access and retrieve information about Maps project items.
- Send newly generated images, data, and analysis results back to Maps.
- Create and add new project items, including tile sets, image layers, and annotations, directly within the Maps project.

This plugin extends Maps functionality and integrate custom analysis and workflows.

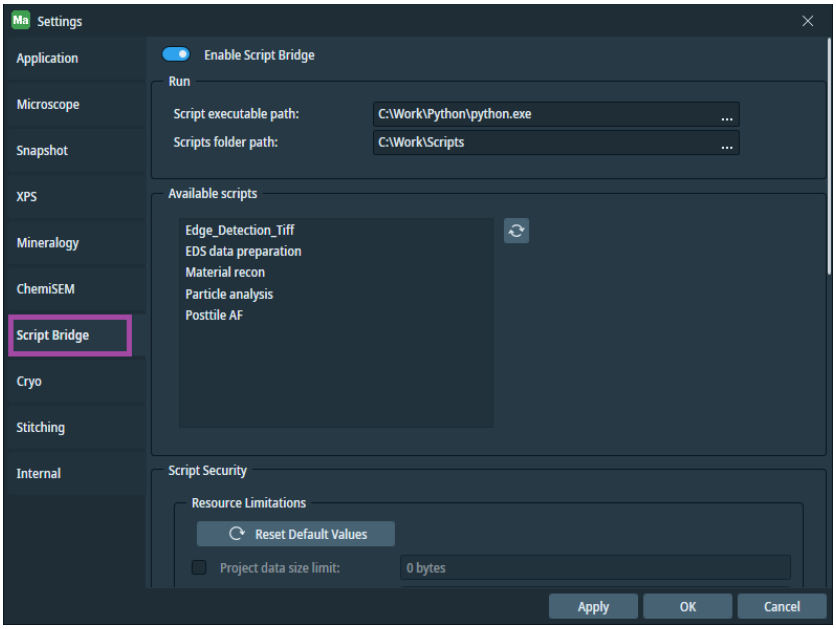
Script Bridge UI

This section describes the elements of the Script Bridge UI in Maps and the configuration of script execution.

Script Bridge settings

Use the Script Bridge settings to configure script execution, including script locations, execution limits, and security options.

To open the settings, navigate to the Options menu, click Settings, and then select the Script Bridge tab.



Script Bridge Enabled checkbox – enables or disables the plugin as a whole. By default, the plugin is enabled. Changing this setting requires restarting Maps. If the plugin is not enabled, the Script Bridge tabs (see below) are hidden, and no script processing can be run. Enabling or disabling the plugin does not affect existing script configuration properties already set in a layer; previously configured properties are preserved.

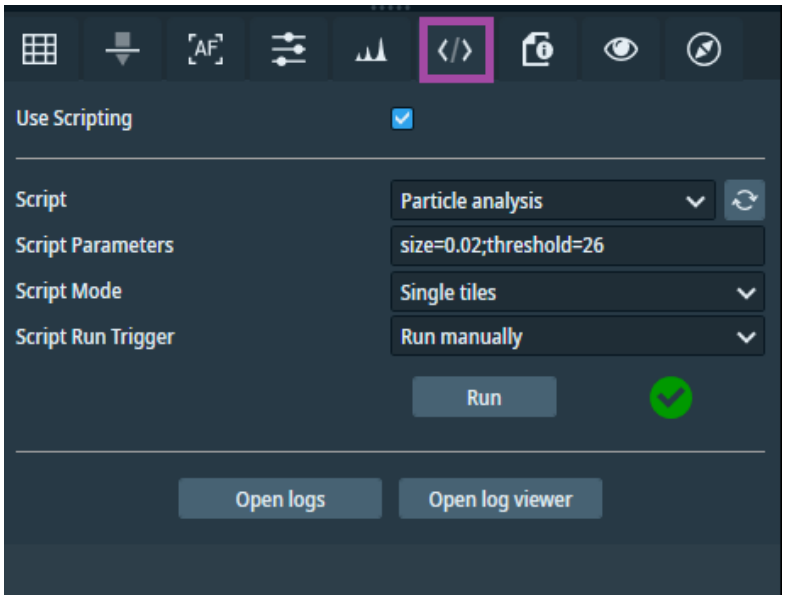
Script executable path - Select the script executable. In most cases, this will be Python.exe.
Note: Python is not included in the Maps installation and must be installed separately to use Script Bridge.

Script folder path – Select the folder containing scripts.

Available scripts - Displays the list of available scripts. The list is updated automatically when a scripts folder is selected. To update the list manually later, click the refresh button.

Tile set Script Bridge settings tab

Script Bridge setup tab for the selected tile set.



Use Scripting – Select Use Scripting to enable script processing for the selected tile set.

Script – Select the script to run.

Script Parameters – Enter any script-specific parameters. Maps sends these parameters to the script when script processing starts as **ScriptParameters** in the initial TileSetRequest. Maps does not process the string in any way.

Script Mode - Select the required script mode. The mode determines how the scripting is executed. See description below.

Script Run Trigger – Select the script run trigger. The selection determines when the script is run. See description below.

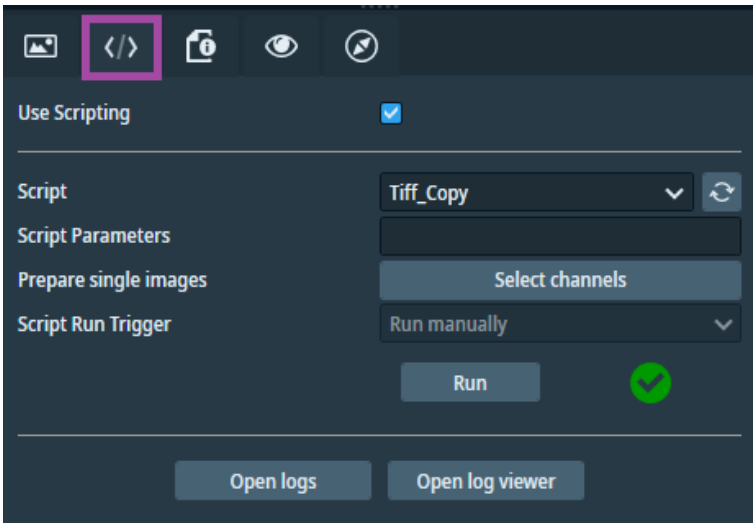
Run– Click this button to start script processing for the tile set. This will create a script processing job in the Processing queue. The job runs immediately if possible or is scheduled to run if other processing jobs are already in the queue. Available only for the manual script trigger.

Open logs – Click this button to open Script Bridge logs for the selected layer in plain text format. The logs open in the default system text viewer.

Open log viewer – Click this button to open logs Script Bridge logs for the selected layer in the Maps log viewer. The viewer displays formatted log entries and allows filtering.

Image layer Script Bridge settings tab

Script Bridge setup tab for the selected image layer.



Use Scripting – Select Use Scripting to enable script processing for the selected tile set.

Script – Select the script to run.

Script Parameters – Enter any script-specific parameters. Maps sends these parameters to the script when script processing starts as **ScriptParameters** in the initial ImageLayerRequest. Maps does not process the string in any way.

Script Run Trigger – The only available run trigger for image layers is Run manually. See description below.

Run– Click this button to start script processing for the image layer. This will create a script processing job in the Processing queue. The job runs immediately if possible or is scheduled to run if other processing jobs are already in the queue.

Open logs – Click this button to open Script Bridge logs for the selected layer in plain text format. The logs open in the default system text viewer.

Open log viewer – Click this button to open logs Script Bridge logs for the selected layer in the Maps log viewer. The viewer displays formatted log entries and allows filtering.

Script execution

This section provides an overview of script execution in Maps, including how scripts are run, the underlying mechanisms, and the communication between the application and the script during processing.

Script source layer

Scripts are always executed for a selected layer. Currently, scripting is supported for **tile sets** and **image layers**. To use scripting on a layer, select the layer and configure the script parameters in the **Script Bridge settings tab** (see above).

The layer for which the script is executed is referred to as the **source layer** in the following sections as well as in script parameters.

Script run trigger

There are the following Script Run Trigger strategies available:

- **Run manually**
  The script is executed manually using the Run button. A scripting job is created in the Processing queue and executed immediately if possible. If other jobs are already scheduled or running, the script will be executed later.
- **Run after tile set acquisition completes**
- The script is executed automatically after the tile set acquisition completes. A scripting job is created in the Processing queue and executed immediately if possible. If other jobs are already scheduled or running, the script will be executed later.
- **Run after tile set acquisition completes (blocking)**
  The script is executed automatically after the tile set acquisition completes. The **acquisition job queue is blocked** until the script execution finishes. For this purpose, a scripting job is created and added to the acquisition job queue immediately after the acquisition job.
- **Run after each tile is acquired**
  The script is triggered during the tile set acquisition. After each tile acquisition and its processing are completed, a scripting request is sent. The acquisition process does **not wait** for the script to finish. The scripting job runs independently and asynchronously from the acquisition in the Processing job queue.
- **Run after each tile is acquired (blocking)**
  The script is triggered during the tile set acquisition after each tile acquisition and its processing are completed. The acquisition process **waits for the script execution to finish** before continuing with the next tile. This is the only mode in which no separate scripting job is created. Instead, the script execution becomes part of the acquisition job.

Scripting jobs can be stopped and restarted again. The order of jobs in the queue can also be rearranged.

Unfinished scripting jobs are persistent – they are saved together with the project and can be restarted again after the project is reloaded.

Important limitation: Only one scripting process can be active or scheduled for a layer.

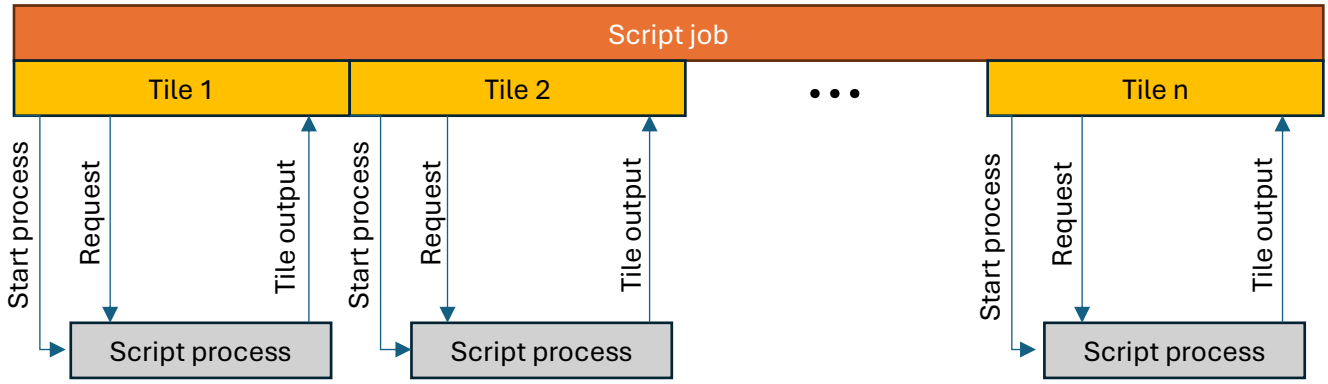
If a scripting process is already active or scheduled, the script setup is disabled until the current scripting job finishes or is removed. To remove it, open the Acquisition or Processing job queue, then stop and remove the scripting job.

Script execution mode - single and batch processing

There is always one scripting job per tile set and script. The difference lies in the number of the script executions in the scripting process.

Single tiles processing (default)

The script is executed separately for each tile. Maps sends a request containing a tile to be processed to the script. The script is expected to finish after processing that tile.



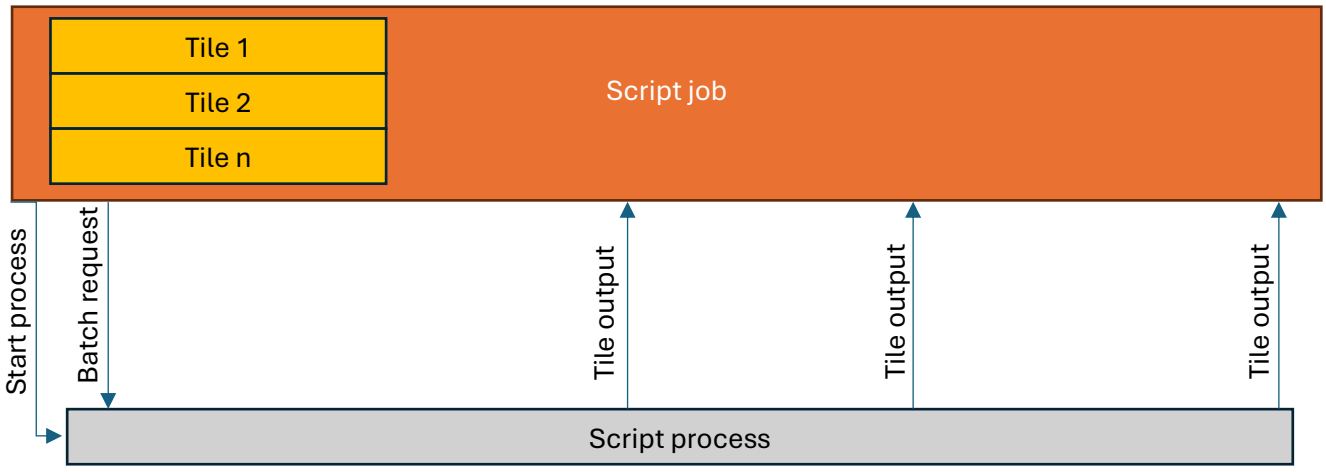
Batch processing

The script is executed only once, at the beginning of the scripting job. Maps sends a request containing all acquired tiles in an array. The script is expected to finish after all tiles have been processed.

Batch processing can be useful for performing analysis on the tile set as a whole. However, it is more complex from the scripting perspective, because the script must iterate over the tile array and is responsible for progress reporting, handling stop requests, and overall execution control.

Batch processing can also reduce overhead because the scripting process is started and stopped only once. On the other hand, it may require more system resources and may exceed limits imposed by cyber security sandboxing.

Batch processing is not available for run-during-acquisition strategies, because tiles are acquired sequentially and cannot be provided as a complete batch in advance.



Maps-script communication

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

Maps reads all text written to the script process standard output. Any text that is not recognized as valid JSON is written to the script logs. This can be useful for debugging and later analysis of the scripting process.

Example:

```
print("Something to be read and logged")
```

This text will be logged but otherwise ignored. However, the output must not start with '{', because Maps will attempt to interpret it as JSON and will likely fail due to an unrecognized structure.

To simplify communication, all JSON messages must be written on a single line, without indentation or newline characters. In Python, this can be done as follows:

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

Each script receives a start (initial) request at startup, which must be read from standard input. Example in Python:

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

Maps start request

This Maps request is sent to the script process through standard input at the very beginning of script execution. Currently, two types of start requests are used: **TileSetRequest** (for tile sets) and **ImageLayerRequest** (for image layers).

The start (initial) request contains all the information required to begin script processing.

Important:
Scripts must always read the start request or otherwise clear the standard input at startup. If this is not done, Maps cannot send any further requests to the script, and the communication may be disrupted.

Synchronous and asynchronous responses

Every script JSON response can include the boolean parameter **RequestConfirmation**.

Synchronous responses

If present and set to True, Maps sends a confirmation containing the result of the response processing to the script's standard input. The script is then expected to read and handle the confirmation. This exchange is **blocking**, making the response and its processing synchronous. Scripts can use this mechanism to synchronize their execution with Maps processing. For example, a script can request confirmation for the creation of a tile set to receive information about the newly created tile set and ensure that it is ready for further processing within the script.

Important:
Always pair responses with confirmations when requested. That is, always read the requested confirmations from the script's standard input. If not done properly, the communication exchange may fail or become mismatched.

Asynchronous responses

If **RequestConfirmation** is not specified or is set to False in a script response, Maps processes the response without sending any confirmation back to the script. Such responses are **asynchronous**. The script does not wait for processing to complete and can continue executing and sending new responses.

This approach is useful when you want to speed up script execution and do not need to use the results from the responses immediately.

MapsBridge Python library

To simplify the development of Python scripts, the MapsBridge.py library is provided. It primarily wraps JSON creation and communication with Maps, making scripting easier and reducing the amount of boilerplate code required.

See detailed description and reference below: MapsBridge library

Script default parameters

When a script is selected in the tile set **Script Bridge setup** tab, any available default parameters are automatically applied.

Default parameters can be specified directly within the script. When Maps scans the script folder, it also parses every script to detect these specifications. The parameters must be provided as a JSON structure within a comment block, delimited by # Default parameters and # Default parameters end. This section can appear anywhere in the script file, but placing it at the beginning is recommended, as it allows faster parsing.

Using default parameters is optional; scripts are not required to include this section.

The following parameters can be specified in the default parameters. All parameters are optional and may be omitted:

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

RunMode – Any of *manual*, *whencompleted*, *whencompletedblocking*, *live*, *liveasync*.
ScriptMode – *batch* or *singletiles*
ScriptParameters – Script-specific parameters, string is used as is.
PrepareImages – comma separated indices of channels for which images should be prepared. For image layer only, otherwise ignored

Example

```
# Default parameters
#{
#"RunMode" : "manual",
#"ScriptMode" : "batch",
#"PrepareImages" : "1,3",
#"ScriptParameters" : "threshold=120;maximum=255",
#}
# Default parameters end
```

Script Bridge Security

The Script Bridge allows for user scripts be executed inside Maps. Because scripts may come from different sources, security measures are in place to protect the system from harmful actions, prevent scripts from accessing sensitive data and ensure the scripts only do what they are allowed to do.

This topic explains how Script Bridge keeps script execution safe and secure.

Resource Limitations

Resource limitations control how much system capacity a script can use while it runs. These limits can be changed or disabled in Script Bridge Settings.

Resource limits are in place to:

- Prevent scripts from slowing down the system
- Stop scripts that run too long or get stuck
- Protect against mistakes in script logic

Even well-intentioned scripts can cause issues if they are allowed to use unlimited resources.

Types of Resource Limits

- **Project data size limit**: Maximum size of data added to Maps project during a single script execution.
- **Created project items limit**: Maximum number of items added to Maps project during a single script execution. The items include tile sets, image layers and annotations.
- **Max CPU Usage**: Maximum CPU usage percentage allowed for the script execution process and its child processes.
- **Max memory for all processes**: Maximum memory that can be used by the script execution and all its child processes.
- **Execution timeout**: Maximum time the script execution can take. If script execution does not finish within this time limit, it will be terminated.

Secure Script Execution

Secure script execution isolates the script execution from the host environment. Scripts executed through the Script Bridge are run in a constrained runtime that prevents direct interaction with the operating system, file system, network, or sensitive application state. This is done by running it inside a Windows security sandbox (also called an AppContainer). This sandbox is a standard Windows security feature that isolates the application from the rest of the system to improve safety and reliability.

> Disabling this option allows scripts to have full access to the system, including reading and writing files, accessing network resources and execution other programs. This can pose significant security risks, especially if the executed scripts were obtained from untrusted sources. It is **strongly recommended** to have Secure script execution enabled at all times.

While enabled, Secure script execution ensures the Sandbox has no access to files on the system with following exceptions:

- The Sandbox will always have permission to read and execute files in the folder containing the script executable and its sub-folders*
- The Sandbox will always have permission to read and execute files in the folder containing the executed script its sub-folders*
- The Sandbox will always have permission to read and execute files in the current project folder and its sub-folders*
- The Sandbox will always have permission to read and execute files in the user specified Readable Folders and their sub-folders*
- The Sandbox will always have permission to create and modify files in the user specified Writeable Folders and their sub-folders*
- The Sandbox will always have permission to create and modify files in its Temp folder

*For sub-folders to inherit the access permission to the parent, the access inheritance has to be enabled for that folder. If it is not, the sub-folder needs to be added as Readable or Writeable folder explicitly.

Secure script execution by default does not allow for internet access, but this capability can be enabled in settings.

Secure script execution does not allow accessing any network resources like network drives or any external systems. All files must be accessible on the local system.

> When using system Temp path or accessing the TMP or TEMP environment variables inside the scripts while using Secure script execution, the system will always direct to paths like this:
>
> *C:\Users\<your-username>\AppData\Local\Packages\潣桴牥潤瑣潲摬敮瑳潰湡獰潤瑳獰畱湩x\AC\Temp*
>
> The seemingly “random” or non-English characters are not an error and do not indicate corrupted files. These names are automatically generated by Windows and are based on an internal security identifier. While the folder works normally and is fully supported by the operating system, because of this it is recommended to avoid using Temp folder in the scripts since depending on the encoding used by the scripting environment such paths may cause issues (e.g. when trying to print them).

Common Issues and How to Fix Them

The issues below are the most common problems with Secure, along with simple suggestions for resolving them.

Missing library

What happens: A script fails when starting execution.

Why: Sandboxed scripts cannot access system files or folders unless they are explicitly allowed.

How to fix it:

- Ensure all script dependencies are readable by the sandbox enviroment

Missing library, File Not Found or Access Denied

What happens: A script fails when trying to read or write a file.

Why: Sandboxed scripts cannot access system files or folders unless they are explicitly allowed.

How to fix it:

- Ensure the required data is passed into the script as input
- Use only approved file locations
- Use only local paths

Script Exceeds Memory Limit

What happens: The script fails while processing data with out-of-memory or memory allocation error

Why: The script used more memory than allowed in the sandbox.

How to fix it:

- Process smaller data sets
- Increase or disable the memory limit

Script Works Outside the System but Fails Here

What happens: A script runs locally but fails in the application.

Why: The sandbox environment is more restricted than a local system.

How to fix it:

- Adjust the script to match sandbox rules
- Avoid relying on system access or external services
- Test scripts with sandbox limitations in mind

JSON requests and responses

This section describes JSON **requests, responses and other messages. It also describes** their functionality, purpose, and usage rules.

JSON schemas

All JSON structures described in this section are fully defined in separate .json schema files.
The schemas can be found in: *<Maps installation folder>\Script Bridge\API*

Maps descriptors

Descriptors are used to describe Maps project objects wherever they are referenced in Script Bridge requests or responses.

LayerInfo

Basic description of Maps project layer. Sent to scripts as an answer to GetLayerInfo.

Request type: LayerInfo
Full JSON schema: LayerInfo.json
In MapsBridge: `LayerInfo` class

Property	Type	Required	Description
RequestType	str	Yes	Request type identifier, always 'LayerInfo'.
RequestGuid	str	Yes	GUID of the request in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ResponseGuid	str	Yes	GUID of the response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
LayerExists	bool	Yes	True if a layer with the specified name exists; False otherwise.
LayerGuid	str	No	GUID of the layer in registry format; null if not found.
LayerName	str	No	Name of the layer; null if not found.
LayerType	str	No	Layer type: 'TileSetInfo', 'ImageLayerInfo', 'Annotation', 'LayerGroup', 'Other'; null if not found.
LayerInfo	TileSetInfo / ImageLayerInfo / AnnotationInfo/ null	No	Detailed layer info of type related to LayerType. Nullif not available or not requested.

Example

```
{
  "RequestType": "LayerInfo",
  "RequestGuid": "{12345678-90ab-cdef-1234-567890abcdef}",
  "ResponseGuid": "{abcdef12-3456-7890-abcd-ef1234567890}",
  "LayerExists": true,
  "LayerGuid": "{d3f1c9a2-7b4e-4c8a-9f12-5a6b7c8d9e01}",
  "LayerName": "NeuronLayer_A01",
  "LayerType": "ImageLayerInfo",
  "LayerInfo": {
    "$ref": "ImageLayerInfo.json"
  }
}
```

TileSetInfo

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

Request type: N/A
Full JSON schema: TileSetInfo.json
In MapsBridge: `TileSetInfo` class

Property	Type	Required	Description
Name	str	Yes	Name of the tile set.
Guid	str	Yes	GUID in registry format "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}".
ColumnCount	int	Yes	Number of columns.
RowCount	int	Yes	Number of rows.
ChannelCount	int	Yes	Number of channels.
IsCompleted	bool	Yes	True if acquisition completed.
DataFolderPath	str	Yes	Full path to tile set data folder.
PixelFormat	str	Yes	Pixel format. String representation of .NET System.Windows.Media.PixelFormat.
Size.Width	float	Yes	Total width in meters.
Size.Height	float	Yes	Total height in meters.
StagePosition	float[X,Y]	Yes	Stage position, components X and Y in meters.
Rotation	float	Yes	Rotation around the layer center in degrees.
AcquisitionStagePosition	float[X,Y]	Yes	Stage position during acquisition, components X and Y in meters.
AcquisitionStageRotation	float	Yes	Stage rotation during acquisition in degrees.
AcquisitionRotation	float	Yes	The tile set rotation during acquisition in degrees.
TileSize.Width	float	Yes	Tile width in meters.
TileSize.Height	float	Yes	Tile height in meters.
TileResolution.Width	int	Yes	Tile width in pixels.
TileResolution.Height	int	Yes	Tile height in pixels.
PixelToStageMatrix	MatrixInfo	Yes	Transformation matrix.
Channels	ChannelInfo[]	Yes	Channel definitions.
Tiles	TileInfo[]	Yes	Tile definitions. See below.

Notes

- **StagePosition** and **Rotation** define the absolute position of the tile set, including all alignments applied to it. The original position at the time of acquisition can be found in **AcquisitionStagePosition**, **AcquisitionStageRotation**, and **AcquisitionRotation**.
- **PixelToStageMatrix** is a 2D matrix for transformation from the tile set pixel space to absolute physical stage coordinates. The matrix is related to the tile set center. See Usage of PixelToStageMatrix in Appendix.

ImageLayerInfo

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

Request type: N/A
Full JSON schema: ImageLayerInfo.json
In MapsBridge: `ImageLayerInfo` class

Property	Type	Required	Description
Name	str	Yes	Name of the image layer.
Guid	str	Yes	GUID in registry format "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}".
ChannelCount	int	Yes	Number of channels in the layer.
DataFolderPath	str	Yes	Full path to the folder containing pyramid data.
StagePosition	float[X,Y]	Yes	Stage position, components X and Y in meters.
Rotation	float	Yes	Layer rotation around its center in degrees.
Size	float[Width, Height]	Yes	Total width of the layer, components Width and Height in meters.
TotalLayerResolution	Int[Width, Height]	Yes	Total size of the layer in pixels, components Width and Height
PixelToStageMatrix	MatrixInfo	Yes	Pixel-to-stage transformation matrix (external reference).
Channels	ChannelInfo[]	Yes	Array of channels in the layer (external reference).
OriginalTileSet	TileSetInfo	No	Optional original tile set used to create the layer if available.

Notes

- **StagePosition** and **Rotation** define the absolute position of the tile set, including all alignments applied to it.
- If the source image layer is a result of a tile set stitching and the original tile set still exists in the project, the image layer info contains also a description of the tile set as **OriginalTileSet**. Scripts can use this description for additional information and context of the image layer.
- **PixelToStageMatrix** is a 2D matrix for transformation from the image layer pixel space to absolute physical stage coordinates. The matrix is related to the image layer center. See usage See Usage of PixelToStageMatrix in Appendix.

Example


```
{
  "Name": "NeuronLayer_A01",
  "Guid": "{7a1b2c3d-4e5f-6789-abcd-1234567890ef}",
  "ChannelCount": 1,
  "DataFolderPath": "D:\\Data\\NeuronLayer_A01",
  "StagePosition": {
    "X": 0.012345,
    "Y": 0.008765
  },
  "Rotation": 0.0,
  "Size": {
    "Width": 0.0004,
    "Height": 0.0003
  },
  "TotalLayerResolution": {
    "Width": 4096,
```

```
    "Height": 3072
  },
  "PixelToStageMatrix": {
    "$ref": "MatrixInfo.json"
  },
  "Channels": [
    {
      "$ref": "ChannelInfo.json"
    }
  ],
  "OriginalTileSet": null
}
```

AnnotationInfo

Describes a Maps annotation wherever it is needed in Maps requests.

Request type: N/A
Full JSON schema: AnnotationInfo.json
In MapsBridge: `AnnotationInfo` class

Name	Type	Required	Description
Guid	str	Yes	GUID of the annotation in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
Name	str	Yes	Name of the annotation.
Rotation	float	Yes	Rotation of the annotation around its center in degrees (counter clock-wise).
Size	float[Width, Height]	Yes	Physical size of the annotation, components Width and Height in meters.
StagePosition	Float[X,Y]	Yes	Stage position of the annotation center, components X and Y in meters.

Example

```
{
  "Guid": "{3f9a1b2c-7d4e-4a8b-9c12-5e6f7a8b9c01}",
  "Name": "Cell Cluster A",
  "Rotation": 15.0,
  "Size": {
    "Width": 0.000085,
    "Height": 0.000060
  },
  "StagePosition": {
    "X": 0.0032962,
    "Y": -0.0008263
  }
}
```

TileInfo

Describes a tile of Maps tile set. It is a part of the tile set information.

Request type: N/A
Full JSON schema: TileInfo.json
In MapsBridge: `TileInfo` class

Property	Type	Required	Description
Column	int	Yes	Tile column index (1-based).
Row	int	Yes	Tile row index (1-based).
StagePosition	float	Yes	Stage position, components X and Y in meters.
TileCenterPixelOffset	Int[X,Y]	Yes	Pixel offset relative to tile set center, components X and Y in pixels
ImageFileNames	dict[str,str]	Yes	Dictionary mapping channel index to image file name. Combine with tile set DataFolderPath to obtain full path.

Example

```
{
  "Column": 2,
  "Row": 1,
  "StagePosition": {
    "X": 0.012445,
    "Y": 0.008765
  },
  "TileCenterPixelOffset": {
    "X": 1024,
    "Y": 0
  },
  "ImageFileNames": {
        "0": "tile_r1_c1_ch0.tif",
        "1": "tile_r1_c1_ch1.tif"
  }
}
```

Notes

TileCenterPixelOffset is the pixel offset relative to the center of the entire tile set. Offsets are negative for tiles to the left or above the center. This offset together with the tile set **PixelToStageMatrix** should be used for converting tile pixel coordinates to stage coordinates. See Usage of PixelToStageMatrix in Appendix.

ChannelInfo

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

Request type: N/A
Full JSON schema: ChannelInfo.json
In MapsBridge: `ChannelInfo` class

Property	Type	Required	Description
Index	int	Yes	Index of the channel within the tile set (0-based).
Name	str	Yes	Name of the channel (e.g., 'DAPI', 'FITC').
Color	str	Yes	Channel color in hex notation (e.g., '#DD00FF').

Example

```
{
  "Index": 0,
  "Name": "DAPI",
  "Color": "#0000FF"
}
```

Maps confirmations

Confirmations are sent from Maps to scripts as the result of processing a script response, when confirmation has been requested.

Confirmation

A general confirmation indicating the result of a non-specific action.

Request type: Confirmation
Full JSON schema: Confirmation.json
In MapsBridge: `Confirmation` class

Name	Type	Required	Description
RequestType	str	Yes	Request type identifier, always 'Confirmation'.
RequestGuid	str	Yes	GUID of this request in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ResponseGuid	str	Yes	GUID of the original response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ScriptName	str	Yes	Name of the executed script.
IsSuccess	bool	Yes	True if the request completed successfully; False otherwise.
ErrorMessage	str	No	Optional error message. Present when IsSuccess is False. May be null or missing otherwise.
WarningMessage	str	No	Optional warning message generated during processing. May be null or missing.

Example

```
{
  "RequestType": "Confirmation",
  "RequestGuid": "{7cb17149-28c3-42d7-831b-a52e202b3c7d}",
  "ResponseGuid": "{a1b2c3d4-5678-90ab-cdef-1234567890ab}",
  "ScriptName": "DetectFeatures",
  "IsSuccess": true,
  "WarningMessage": null
}
```

TileSetConfirmation

Confirmation describing the result of a tile set creation or retrieval request.

Request type: TileSetConfirmation
Full JSON schema: TileSetConfirmation.json
In MapsBridge: `TileSetCreateInfo` class

Name	Type	Required	Description
RequestType	str	Yes	Request type identifier. Always 'TileSetConfirmation'.
RequestGuid	str	Yes	GUID of this request in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ResponseGuid	str	Yes	GUID of the original response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ScriptName	str	Yes	Name of the executed script.
IsSuccess	bool	Yes	True if the tile set was successfully created or found; False otherwise.
ErrorMessage	str	No	Optional error message. Present when IsSuccess is False. May be null or missing otherwise.
IsCreated	bool	No	Optional. True if the tile set was newly created; False if an existing tile set was reused. May be null or missing.
TileSet	TileSetInfo | null	No	Optional TileSetInfo object describing the tile set. May be null or missing if IsSuccess is False.
WarningMessage	str	No	Optional warning message generated during processing. May be null or missing.

Example

```
{
  "RequestType": "TileSetConfirmation",
  "RequestGuid": "{7cb17149-28c3-42d7-831b-a52e202b3c7d}",
  "ResponseGuid": "{a1b2c3d4-5678-90ab-cdef-1234567890ab}",
  "ScriptName": "CreateTileSet",
  "IsSuccess": true,
  "IsCreated": true,
  "TileSet": {
      "$ref": "TileSetInfo.json"
  }
}
```

CreateImageLayerConfirmation

Confirmation describing the result of an image layer creation request.

Request type: CreateImageLayerConfirmation
Full JSON schema: CreateImageLayerConfirmation.json
In MapsBridge: `ImageLayerCreateInfo` class

Name	Type	Required	Description
RequestType	str	Yes	Request type identifier. Always 'CreateImageLayerConfirmation'.
RequestGuid	str	Yes	GUID of this request in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ResponseGuid	str	Yes	GUID of the original response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ScriptName	str	Yes	Name of the executed script.
IsSuccess	bool	Yes	True if the image layer was successfully created; False otherwise.
ErrorMessage	str	No	Optional error message. Present if IsSuccess is False. May be null or missing.
WarningMessage	str	No	Optional warning message. May be null or missing.
ImageLayer	ImageLayerInfo | null	No	Optional ImageLayerInfo object describing the created image layer. May be null or missing if IsSuccess is False.

Example

```
{
  "RequestType": "CreateImageLayerConfirmation",
  "RequestGuid": "{7cb17149-28c3-42d7-831b-a52e202b3c7d}",
  "ResponseGuid": "{b2a14f5c-3c9d-4e7f-8a21-6d9f3c2e1a55}",
  "ScriptName": "CreateImageLayer",
  "IsSuccess": true,
  "ImageLayer": {
      "$ref": "ImageLayerInfo.json"
  }
}
```

CreateAnnotationConfirmation

Confirmation describing the result of an annotation creation request.

Request type: CreateAnnotationConfirmation
Full JSON schema: CreateAnnotationConfirmation.json
In MapsBridge: `AnnotationCreateInfo` class

Name	Type	Required	Description
RequestType	str	Yes	Request type identifier. Always 'CreateAnnotationConfirmation'.
RequestGuid	str	Yes	GUID of this request in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ResponseGuid	str	Yes	GUID of the original response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ScriptName	str	Yes	Name of the executed script.
IsSuccess	bool	Yes	True if the annotation was successfully created or found; False otherwise.
ErrorMessage	str	No	Optional error message. Present if IsSuccess is False. May be null or missing.
WarningMessage	str	No	Optional warning message generated during processing. May be null or missing.
Annotation	AnnotationInfo | null	No	Optional AnnotationInfo object describing the created annotation. May be null or missing if IsSuccess is False.

Example

```
{
  "RequestType": "CreateAnnotationConfirmation",
  "RequestGuid": "{7cb17149-28c3-42d7-831b-a52e202b3c7d}",
  "ResponseGuid": "{b2a14f5c-3c9d-4e7f-8a21-6d9f3c2e1a55}",
  "ScriptName": "CreateAnnotation",
  "IsSuccess": true,
  "Annotation": {
      "#ref" : "AnnotationInfo.json"
  }
}
```

Maps requests

Requests are messages sent from Maps to scripts that the scripts are required to process.

Maps tile set request

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

Request type: **TileSetRequest**
Full JSON schema: TileSetRequest.json
In MapsBridge: `ScriptTileSetRequest` class

Property	Type	Required	Description
RequestType	string	Yes	Request type. Always "TileSetRequest".
RequestGuid	string	Yes	Unique identifier for the request. GUID in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ScriptName	string	Yes	Name of the executed script.
SourceTileSet	TileSetInfo	Yes	Information about the source tile set. See TileSetInfo structure definition.
TilesToProcess	TileIndex[]	Yes	List of tiles to be processed by the script. Each entry contains 1-based Column and Row indices.
ScriptParameters	string | null	No	Parameters specified for the script.

TileIndex structure:

Property	Type	Required	Description
Column	int	Yes	1-based column index of the tile.
Row	int	Yes	1-based row index of the tile.

Example

```
{
  "RequestType": "TileSetRequest",
  "RequestGuid": "{123e4567-e89b-12d3-a456-426614174001}",
  "ScriptName": "SegmentNuclei",
  "ScriptParameters": "threshold=0.65;min_area=25",
  "SourceTileSet": {
     . . .
  },
  "TilesToProcess": [
    { "Column": 1, "Row": 1 },
    { "Column": 2, "Row": 1 },
    { "Column": 1, "Row": 2 }
  ]
}
```

Maps image layer request

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

Request type: **ImageLayerRequest**
Full JSON schema: ImageLayerRequest.json
In MapsBridge: `ScriptImageLayerRequest` class

Property	Type	Required	Description
RequestType	string	Yes	Request type. Always "ImageLayerRequest".
RequestGuid	string	Yes	Unique identifier for the request. GUID in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
PreparedImages	dictionary	Yes	Dictionary of full paths to prepared exported pyramid images, organized per channel index.
ScriptName	string	Yes	Name of the executed script.
SourceImageLayer	ImageLayerInfo	Yes	Information about the source image layer. See ImageLayerInfo structure definition.
ScriptParameters	string | null	No	Parameters specified for the script.

Example

```
{
  "RequestType": "ImageLayerRequest",
  "RequestGuid": "{123e4567-e89b-12d3-a456-426614174001}",
  "ScriptName": "MergeChannels",
  "ScriptParameters": "method=maximum;normalize=true",
  "SourceImageLayer": {
    "LayerGuid": "{aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee}"
  },
  "SourceTileSet": {
    . . .
  },
  "PreparedImages": [
    {
      “0”:"C:\\MicroscopyData\\Prepared\\dapi.tif",
      "6”:"C:\\MicroscopyData\\Prepared\\fitc.tif"
    }
  ]
}
```

Maps stop request

The stop request is sent to the script process when processing should be terminated, typically when a user explicitly stops processing in Maps.

Upon receiving this request, the script process is expected to stop and exit. Stopped processing can be restarted later. If necessary, scripts should properly manage their data and resources to allow a clean restart.

Maps will wait for the process to exit. If the process does not exit within 30 seconds, it will be forcibly terminated.

Request type: **Stop**
Full JSON schema: MapsRequest.json
In MapsBridge: N/A

Property	Type	Required	Description
Request	str	Yes	Request type identifier. Allowed values is "Stop".
RequestGuid	str	Yes	GUID of the request in registry format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}".

Example

```
{
  "Request": "Stop",
  "RequestGuid": "{9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6a}"
}
```

Maps exit request

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

Upon receiving this request, the script process is expected to stop and exit. Unlike the Stop request, in case of Exit it is not expected that the scripting will continue (mostly because some Maps internal error). If needed, scripts should gratefully terminate the processing.

Maps will wait for the process to exit. If the process does not exit within 30 seconds, it will be forcibly terminated.

Request type: **Exit**
Full JSON schema: MapsRequest.json
In MapsBridge: N/A

Property	Type	Required	Description
Request	str	Yes	Request type identifier. Allowed values is "Exit".
RequestGuid	str	Yes	GUID of the request in registry format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}".

Example

```
{
  "Request": "Exit",
  "RequestGuid": "{9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6a}"
}
```

Script responses

Responses are messages sent from scripts to Maps that Maps is to process.

GetLayerInfo

Send this response to get information about a project layer with the given name.

Response type: **GetLayerInfo**
Full JSON schema: GetLayerInfo.json
In MapsBridge: `Get layer information` ()

Property	Type	Required	Description
ResponseType	str	Yes	Response type identifier, always 'GetLayerInfo'.
ResponseGuid	str	Yes	GUID of the response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
LayerName	str	Yes	Name of the layer to retrieve. Case insensitive
RequestFullInfo	bool	No	Optional. False for basic info, True for detailed info (TileSetInfo, ImageLayerInfo, AnnotationInfo). Default is False.

Example:

```
{
  "ResponseType": "GetLayerInfo",
  "ResponseGuid": "{a1b2c3d4-5678-90ab-cdef-1234567890ab}",
  "LayerName": "Neuron_A01",
  "RequestFullInfo": true
}
```

TileOutput

This response sends output images to a tile in the target tile set. It also informs Maps that the script processing of the tile finished.

Response type: **TileOutput**
Full JSON schema: TileOutput.json
In MapsBridge: `send_single_tile_output` ()

Property	Type	Required	Description
ResponseType	str	Yes	Response type identifier, always 'TileOutput'.
ResponseGuid	str	Yes	GUID of the response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
Column	int	Yes	1-based column index of the tile.
Row	int	Yes	1-based row index of the tile.
ImageFileOutputs	list[dict]	Yes	Array of image outputs to be sent to tile sets. See ImageFileOutput description below.

ImageFileOutput

Full JSON schema: part of TileOutput.json

Property	Type	Required	Description
TargetTileSetGuid	str	No	Optional GUID of the target tile set in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
TargetChannelName	str	Yes	Name of the target channel.
ImageFilePath	str	Yes	Full path to the image file to be sent.
KeepFile	bool	No	Optional. True to keep the original file; False deletes it after sending. Default is False.

Notes

- The **TileOutput** response informs Maps that processing of the given tile has finished. Maps will remove the tile from the list of tiles to be processed and update the progress of the script processing job. Multiple **TileOutput** responses can be sent for a single tile if multiple images need to be produced, but the recommended approach is to send a single response containing multiple entries in **ImageFileOutputs**. Note, that MapsBridge offers only the single output response with send_single_tile_output (). Feel free to modify the function for multiple file outputs.
- TileOutput can be sent even with empty **ImageFileOutputs**, only to inform Maps that the script the tile processing finished.
- If **TargetTileSetGuid** is specified, a tile set with that GUID must exist in the project; otherwise, the script process will terminate with an error. If not specified, the file will be sent back to the source tile layer.
- Each file may target a different tile set using **TargetTileSetGuid**, so a single tile output can update multiple tile sets.
- If the channel specified in **TargetChannelName** does not exist in the target tile layer, it will be created. Existing channels must have been created via Script Bridge (by any script); otherwise, the script process will terminate with an error. Maps does not allow scripts to overwrite data from other sources.
- Maps does not allow files to be deleted from read-only folders. In such cases, **KeepFile** = False will be ignored, and the file will be copied instead of moved. A warning is logged in the script logs.

Important

- The image file must match the pixel format and resolution of the target tile set if the tile set already contains data. Maps will automatically convert and resize the image as needed. Currently, the supported pixel formats in Maps are **Gray8** and **Gray16**.
- Image files are renamed internally during import to comply with Maps filename conventions.
- Never attempt to overwrite tile images in a Maps project directly. Maps will not recognize such changes, and doing so may corrupt the project.

Example

```
{
  "ResponseType": "TileOutput",
  "ResponseGuid": "{f1a2b3c4-5678-90ab-cdef-1234567890ab}",
  "Column": 2,
  "Row": 1,
  "ImageFileOutputs": [
    {
```

```
      "TargetTileSetGuid": "{d3f1c9a2-7b4e-4c8a-9f12-5a6b7c8d9e01}",
      "TargetChannelName": "DAPI",
      "ImageFilePath": "C:\\Temp\\tile_r001_c002_ch0.tif",
      "KeepFile": true
    },
    {
      "TargetChannelName": "FITC",
      "ImageFilePath": "C:\\Temp\\tile_r001_c002_ch1.tif"
    }
  ]
}
```

CreateTileSet

This response creates a new tile set and prepare it for acquisition.

Response type: **CreateTileSet**
Full JSON schema: CreateTileSet.json
In MapsBridge: `create_tile_set` ()

Property	Type	Required	Description
ResponseType	string	Yes	Response type. Always "CreateTileSet".
ResponseGuid	string	Yes	Unique identifier for the response. GUID in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
TileSetName	string	Yes	Desired tile set name.
StagePosition	[float | str, float | str, float | str]	Yes	Stage position of the tile set center: X, Y (meters), StageR (degrees). Each value may be a number or string (may include value and unit).
TileHfw	float | str	Yes	Horizontal field of view of one tile in meters or string (may include value and unit).
TotalSize	[float | str, float | str]	Yes	Total size of the tile set: Width and Height in meters or string (may include value and unit).
PixelSize	float | str | null	No	Size of one pixel in meters or string (may include value and unit).
Rotation	float | str | null	No	Rotation around the center in degrees or string (may include value and unit). Default is 0.
ScheduleAcquisition	bool | null	No	True to schedule the tile set acquisition. Default is True.
TargetLayerGroupName	string | null	No	Name of a layer group in which the tile set should be created.
TemplateName	string | null	No	Name of the tile set template to be used for creation.
TileResolution	[int, int] | null	No	Resolution of one tile in the tile set: [width, height] in pixels.
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a TileSetConfirmation response after this response is processed (see TileSetConfirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

Notes

- The tile set is always created for the currently active platform to which Maps is connected (e.g., SEM, TEM, Phenom). It is not possible to create a tile set for a different platform, nor is it possible to create a tile set while Maps is completely offline.
- This response always creates a new tile set. If a layer with the same name already exists in the project, the new tile set name is automatically postfixed with (N).
- **TileSetName** must not contain characters that are invalid in file names. Maps cannot create tile sets with such names, and attempting to do so will terminate the script processing with an error.
- If **TargetLayerGroupName** is specified, the tile set is added to that layer group. The group is created if it does not exist. If not specified, the new tile set is added to the same layer group as the source layer.
- The tile set is always created using a template. This can be either the default (or currently selected) template in Maps, or a template specified with **TemplateName**. If a template name is specified but does not exist, the script process will terminate with an error.
- **TileHfw**, **Rotation**, **TileResolution**, or **PixelSize** can be specified to override the corresponding values in the template.
- Specify either **TileResolution** or **PixelSize** to calculate and override the resolution of the tile set from the template. Only one of these parameters may be specified; otherwise, the script process will terminate with an error. Note that some microscopes support only specific resolutions; if the specified resolution is not supported, the nearest available resolution will be used instead.
- To prepare the tile set for acquisition, set **ScheduleAcquisition** to True. The tile set acquisition job will be scheduled int the acquisition queue. The tile set will be acquired when the queue reaches the job. Maps does not start the acquisition queue automatically; if the queue is not running, it must be started manually.
- **TileSetName** must not contain characters that are invalid in file names. Maps cannot create tile sets with such names, and attempting to do so will terminate the script process with an error.
- If **RequestConfirmation** is True, Maps will send a TileSetConfirmation (see above).

Example

```
{
  "ResponseType": "CreateTileSet",
  "ResponseGuid": "{123e4567-e89b-12d3-a456-426614174001}",
  "TileSetName": "HighResTiles",
  "TotalSize": ["20 mm", "10 mm"],
  "StagePosition": ["3 mm", "10 mm", "45 deg"],
  "TileHfw": "10 mm",
  "TileResolution": [1024, 1024],
  "PixelSize": "5 um",
  "Rotation": "15 deg",
  "TemplateName": "DefaultTemplate",
  "TargetLayerGroupName": "Acquisitions",
  "ScheduleAcquisition": true
}
```

GetOrCreateOutputTileSet

This response creates a new tile set or reuses an existing one to serve as the target tile set for script outputs. The tile set is **read-only** and cannot be acquired; it contains only data provided by the script.

Response type: **GetOrCreateOutputTileSet**
Full JSON schema: GetOrCreateOutputTileSet.json
In MapsBridge: `get_or_create_output_tile_set()`

Property	Type	Required	Description
ResponseType	string	Yes	Response type. Always "GetOrCreateOutputTileSet".
ResponseGuid	string	Yes	Unique identifier for the response. GUID in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
TileSetName	string	Yes	Desired tile set name.
Resolution	[int, int] | null	No	Resolution of one tile in the tile set: [width, height] in pixels.
TargetLayerGroupName	string | null	No	Name of a layer group in which the tile set should be created.
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a TileSetConfirmation response after this response is processed (see TileSetConfirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

- If a tile set of the same name exists in the project, this response will reuse it. This is useful for keeping a single output tile set for script results. Rules for reusing:
    - The tile set must have been created by Script Bridge (by any script)
    - The tile set has the same number of columns and rows as the source tile set.
- If these conditions are not met, a **new tile set** is created. Since the tile set is not intended for acquisition, it can be created even if Maps is offline. No template is required for creating the tile set.
- If a new tile set is created, **TileSetName** must not contain characters that are invalid in file names. Maps cannot create tile sets with such names, and attempting to do so will terminate the script processing with an error.
- If a layer with the same name already exists in the project, the new tile set name is automatically postfixed with (N).
- If a new tile set is created and **TargetLayerGroupName** is specified, the tile set is added to that layer group. The group will be created if it does not exist. If not specified, the new tile set is added to the same layer group as the source layer.
- If **RequestConfirmation** is True, Maps will send a TileSetConfirmation (see above).

Example

```
{
  "ResponseType": "GetOrCreateOutputTileSet",
  "ResponseGuid": "{123e4567-e89b-12d3-a456-426614174001}",
  "TileSetName": "HighResTiles",
  "Resolution": [2048, 2048],
  "TargetLayerGroupName": "ProcessedData"

}
```

CreateChannel

Creates or alters a channel in a tile set.

Response type: **CreateChannel**
Full JSON schema: CreateChannel.json
In MapsBridge: `create_channel` ()

Property	Type	Required	Description
ResponseType	string	Yes	Response type. Always "CreateChannel".
ResponseGuid	string	Yes	Unique identifier for the response. GUID in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
TargetChannelName	string	Yes	Name of the target channel.
ChannelColor	string | null	No	Channel color in hex notation (e.g. #DD00FF). Default is #FFFFFF.
IsAdditive	bool | null	No	If True, the channel is additive. Default is False.
TargetTileSetGuid	string | null	No	GUID of the target tile set in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a Confirmation response after this response is processed (see Confirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

Notes

- If **TargetTileSetGuid** is specified, a tile set with that GUID must exist in the project; otherwise, the script processing will terminate with an error. If omitted or null, the source tile set is used.
- If a channel with **TargetChannelName** does not exist in the target tile set, it will be created. If the channel already exists, its properties (color, additivity) will be updated. Existing channels must have been created by Script Bridge (by any script); otherwise, the script processing will terminate with an error. Maps does not allow scripts to modify channels created by other sources.

Example

```
{
  "ResponseType": "CreateChannel",
  "ResponseGuid": "{123e4567-e89b-12d3-a456-426614174001}",
  "TargetChannelName": "Fluorescence",
  "ChannelColor": "#DD00FF",
  "IsAdditive": true,
  "TargetTileSetGuid": "{aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee}"
}
```

CreateImageLayer

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

Response type: **CreateImageLayer**
Full JSON schema: CreateImageLayer.json
In MapsBridge: `create_image_layer` ()

Property	Type	Required	Description
ResponseType	string	Yes	Response type. Always "CreateImageLayer".
ResponseGuid	string	Yes	Unique identifier for the response. GUID in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ImageFilePath	string	Yes	Full path to the image file to import.
LayerName	string	Yes	Desired layer name.
AlignToSourceLayer	bool | null	No	If True, the created image layer is aligned to the source layer (pixels mapped over the source). Default is True.
KeepFile	bool | null	No	True to keep the imported image file, False to delete it after import. Default is False.
PixelPosition	[int, int] | null	No	Pixel position relative to the source layer, where (0,0) is the top-left corner.
PixelSize	float | str | null	No	Size of one pixel in meters or string (may include value and unit).
Rotation	float | str | null	No	Rotation around the layer center in degrees or string (may include value and unit). Default is 0.
StagePosition	[float | str, float | str, float | str] | null	No	Stage position of the layer center: X, Y (meters), StageR (degrees). Each value may be a number or string (may include value and unit).
TargetLayerGroupName	string | null	No	Name of a layer group in which the layer should be created.
TotalWidth	float | str | null	No	Total width of the layer in meters or string (may include value and unit).
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a CreateImageLayerConfirmation response after this response is processed (see CreateImageLayerConfirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

- **LayerName** must not contain characters that are invalid in file names. Maps cannot create layers with such names, and attempting to do so will terminate the **script processing** with an error.
- If a layer with the same name already exists in the project, the new layer name is automatically postfixed with (N).
- If **TargetLayerGroupName** is specified, the layer is added to that layer group. The group will be created if it does not exist. If not specified, the new layer is added to the same layer group as the source layer.
- If none of **StagePosition**, **PixelPosition**, or **TotalWidth** are specified (null or missing), the image layer will be positioned and sized to exactly match the source layer (tile set or image layer). To overlay the new layer over the source layer, simply leave these parameters unspecified.
- The total height of the resulting layer is determined by **TotalWidth** and the image aspect ratio.
- The position of the new image layer can be specified using either **StagePosition** or **PixelPosition**:
- **StagePosition** specifies the absolute stage coordinates of the image layer center.
- **PixelPosition** specifies the offset of the image layer center from the top-left corner of the source layer in pixels. Maps automatically calculates the stage position from this offset. Combined with **AlignToSourceLayer**, this can be used to create smaller layers aligned over the source layer.

- If **StagePosition** is specified, **TotalWidth** must also be specified; otherwise, the **script processing** will terminate with an error.
- **PixelPosition** and **StagePosition** cannot be specified simultaneously. If both are provided, **PixelPosition** takes precedence and a warning is logged in the script logs.
- The imported source image may be of any size (large images must be TIFFs). Note that importing very large images may take several minutes or longer.
- Maps does not allow files to be deleted from read-only folders. In such cases, **KeepFile** = False is ignored, and the file is copied instead of moved. A warning is logged in the script logs.
- If **RequestConfirmation** is True, Maps will send a CreateImageLayerConfirmation (see above).

Example

```
{
  "ResponseType": "CreateImageLayer",
  "ResponseGuid": "{123e4567-e89b-12d3-a456-426614174001}",
  "LayerName": "FilteredImage",
  "ImageFilePath": "C:\\images\\filtered.tiff",
  "StagePosition": ["10.5 um", "20.6 mm", "0 deg"],
  "TotalWidth": "50 mm",
  "Rotation": "15 deg",
  "AlignToSourceLayer": true,
  "KeepFile": true,
  "TargetLayerGroupName": "Output images"
}
```

CreateAnnotation

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

Response type: **CreateAnnotation**
Full JSON schema: CreateAnnotation.json
In MapsBridge: `create_annotation()`

Property	Type	Required	Description
ResponseType	string	Yes	Response type. Always "CreateAnnotation".
ResponseGuid	string | guid	Yes	Unique identifier for the response. GUID in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
AnnotationName	string	Yes	Name of the annotation.
StagePosition	[float | str, float | str, float | str]	Yes	Stage position of the annotation center: X, Y (meters), StageR (degrees). String representation can include value and unit.
Color	string | null	No	Color in hex notation (e.g. #DD00FF). Default is #FFFFFF.
IsEllipse	bool | null	No	True if AOI should be an ellipse. Default is False.
Notes	string | null	No	Notes for the annotation.
Rotation	float | str | null	No	Rotation around the center in degrees. String representation can include value and unit. Default is 0.
Size	[float | str, float | str] | null	No	Size of the annotation [Width, Height] in meters. String representation can include value and unit. Default is None.
TargetLayerGroupName	string | null	No	Name of a layer group in which the annotation should be created.
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a CreateAnnotationConfirmation response after this response is processed (see CreateAnnotationConfirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

- **AnnotationName** must not contain characters that are invalid in file names. Maps cannot create layers with such names, and attempting to do so will terminate the **script processing** with an error.
- If a layer with the same name already exists in the project, the new layer name is automatically postfixed with (N).
- If **TargetLayerGroupName** is specified, the layer is added to that layer group. The group will be created if it does not exist. If not specified, the new layer is added to the same layer group as the source layer.
- Depending on whether **Size** is specified, this response creates either Site of Interest or Area of Interest.
- **Rotation** and **IsEllipse** are meaningful only for an Area of Interest, otherwise, they are ignored.
- If **RequestConfirmation** is True, Maps will send a CreateAnnotationConfirmation (see above).

Example

```
{
  "ResponseType": "CreateAnnotation",
  "ResponseGuid": "{123e4567-e89b-12d3-a456-426614174001}",
  "AnnotationName": "AOI_2",
  "StagePosition": ["15 mm", "-25 mm", "45 deg"],
  "Size": ["5.5 mm", "3.6 mm"],
  "Rotation": "30 deg",
  "Notes": "Sample annotation",
  "Color": "#DD00FF",
  "IsEllipse": true,
  "TargetLayerGroupName": "LayerGroup1"
}
```

StoreFile

Stores a file to the target layer.

Response type: **StoreFile**
Full JSON schema: StoreFile.json
In MapsBridge: `store_file()`

Property	Type	Required	Description
ResponseType	string	Yes	Response type. Always "StoreFile".
ResponseGuid	string	Yes	Unique identifier for the response. GUID in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
FilePath	string	Yes	Full path to a file to be stored.
KeepFile	bool | null	No	If True then the file is copied rather than moved, keeping the original. Default is False.
Overwrite	bool | null	No	If True, existing files are overwritten. Otherwise renamed with an (N) postfix. Default is False.
TargetLayerGuid	string | null	No	GUID of the target layer in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a Confirmation response after this response is processed (see Confirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

Notes

- If **TargetTileSetGuid** is specified, a tile set with that GUID must exist in the project; otherwise, the script processing will terminate with an error. If omitted or null, the source tile set is used as the target layer.
- Maps does not allow files to be deleted from read-only folders. In such cases, **KeepFile** = False is ignored, and the file is copied instead of moved. A warning is logged in the script logs.
- Maps does not allow storing potentially harmful files, such are executable files. Attempt to do so will terminate the script processing with an error.

Example

```
{
  "ResponseType": "StoreFile",
  "ResponseGuid": "{123e4567-e89b-12d3-a456-426614174000}",
  "FilePath": "C:/Data/tileset1.dat"
}
```

AppendNotes

Append notes to the target layer.

Response type: **AppendNotes**
Full JSON schema: AppendNotes.json
In MapsBridge: `append_notes()`

Property	Type	Required	Description
ResponseType	string	Yes	Response type. Always "AppendNotes".
ResponseGuid	string	Yes	Unique identifier for the response in registry format: '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
NotesToAppend	string	Yes	Any string to be appended to notes.
TargetLayerGuid	string | null	No	GUID of the target layer in registry format: "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}".
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a Confirmation response after this response is processed (see Confirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

Notes

- If **TargetTileSetGuid** is specified, a tile set with that GUID must exist in the project; otherwise, the script processing will terminate with an error. If omitted or null, the source tile set is used as the target layer.
- Scripts can only append notes, they cannot overwrite or delete existing notes.

Example

```
{
  "ResponseType": "AppendNotes",
  "ResponseGuid": "{123e4567-e89b-12d3-a456-426614174000}",
  "NotesToAppend": "Review completed for this layer."
}
```

ReportActivity

Reports activity. The description will be visible in the script processing job while it is running.

Response type: **ReportActivity**
Full JSON schema: ReportActivity.json
In MapsBridge: `report_activity_description()`

Property	Type	Required	Description
ResponseType	str	Yes	Response type identifier, always 'ReportActivity'.
ResponseGuid	str	Yes	GUID of the response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ActivityDescription	str	Yes	Description of the activity to be reported.
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a Confirmation response after this response is processed (see Confirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

Example

```
{
  "ResponseType": "ReportActivity",
  "ResponseGuid": "{3f5a1c2e-9b7d-4c21-8f6a-2e5d9c1a7b3f}",
  "ActivityDescription": "Processing tile row 3, column 5 (channel DAPI)"
}
```

ReportProgress

Reports progress. The progress will be set to the script processing job.

Response type: **ReportProgress**
Full JSON schema: ReportProgress.json
In MapsBridge: `report_progress()`

Property	Type	Required	Description
ResponseType	str	Yes	Response type identifier, always 'ReportProgress'.
ResponseGuid	str	Yes	GUID of the response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
Progress	float	Yes	Progress value in range 0.0–1.0, where 1.0 represents 100% completion.
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a Confirmation response after this response is processed (see Confirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

Notes

Use Report Progress in batch mode or for image layer processing. For single-tile processing or during acquisition, progress is updated automatically as Maps processes the list of tiles.

Example

```
{
  "ResponseType": "ReportProgress",
  "ResponseGuid": "{8a2c1f4e-6d3b-4e9a-b7c2-5f1d9e3a4c6b}",
  "Progress": 0.42
}
```

Log

Allows logging of information to the script logs.

Response type: **Log**
Full JSON schema: Log.json
In MapsBridge: `log_info()`, `log_warning()`, `log_error()`

Property	Type	Required	Description
ResponseType	str	Yes	Response type identifier, always 'Log'.
ResponseGuid	str	Yes	GUID of the response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
LogErrorMessage	str	No	Optional error message to be logged.
LogInfoMessage	str	No	Optional informational message to be logged.
LogWarningMessage	str	No	Optional warning message to be logged.
RequestConfirmation	bool | null	No	Optional. If True, the client will receive a Confirmation response after this response is processed (see Confirmation schema). If False or missing, the response is processed without confirmation. Default value is False.

Notes

Any of **LogInfoMessage**, **LogWarningMessage**, or **LogErrorMessage** can be specified and will be recorded in the **script logs**. Note that this is purely for logging purposes; sending logs does not affect the script processing workflow or its results.

Example

```
{
  "ResponseType": "Log",
  "ResponseGuid": "{b7a3c9d2-5e4f-4a8b-9c1d-2e6f7a8b9c0d}",
  "LogInfoMessage": "Tile processing started."
}
```

ReportFailure

Allow reporting a failure.

Response type: **ReportFailure**
Full JSON schema: ReportFailure.json
In MapsBridge: `report_failure()`

Property	Type	Required	Description
ResponseType	str	Yes	Response type identifier, always 'ReportFailure'.
ResponseGuid	str	Yes	GUID of the response in registry format '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'.
ErrorMessage	str	No	Optional error message describing the failure. May be null or missing.

Notes

Report Failure is treated as a script error. Processing of the entire script or the current request **will be terminated**. The error is logged and reported to the user.

Example

```
{
  "ResponseType": "ReportFailure",
  "ResponseGuid": "{9d3f7a21-8c4b-4e5a-b2c1-6f8e3a9b2c4d}",
  "ErrorMessage": "Failed to process tile at Column 4, Row 2 due to missing source image."
}
```

MapsBridge library

To simplify the development of Python scripts, the MapsBridge.py library is provided. It primarily wraps JSON creation and communication with Maps, making scripting easier and reducing the amount of boilerplate code required.

Purpose and usage

The library can be imported into scripts as provided, or users may modify it according to their needs. It is supplied as an example, without warranties or guarantees of any kind.

The module can be found in:
<Maps installation folder>\Script Bridge\Examples

MapsBridge requires **Python 3.10 or newer**.

Library API reference

Usage of physical values

All physical values in the library that are defined as `float|str` can be also specified with a string representation of *"value unit"*. This includes length and angle properties.

Supported units for length are *m*, *mm*, *µm*, *um*, *nm*. Examples:
"1.3 um", *"- 1.3 mm"*, *"3E-8 m"*

Supported units for angle are *rad*, *r*, *°*, *deg*, *d*. Examples:
"1.3 rad", *"- 45 deg"*, *"60 °"*

Get or create output tileset

`get_or_create_output_tile_set()`

This function creates a new tile set or reuses a suitable existing one to serve as the target tile set for script outputs. The tile set is **read-only** and cannot be acquired; it contains only data provided by the script.

Wrapper for **GetOrCreateOutputTileSet** response.
The same rules and behaviors described for GetOrCreateOutputTileSet also apply to this function.

```
def get_or_create_output_tile_set(
    tile_set_name: str | None = None,
    tile_resolution: Tuple[int, int] | None = None,
    target_layer_group_name: str | None = None,
    request_confirmation: bool | None = True
) -> TileSetCreateInfo | None:
```

Arguments

tile_set_name (str | None, default: None)
Name of the tile set to get or create. If not specified, the name is derived from the source layer (tile set).

tile_resolution (Tuple[int, int] | None, default: None)
Desired resolution of the tile set. If not specified, the resolution is taken from the source tile set.

target_layer_group_name (str | None, default: None)
Name of the target layer group. If not specified, the tile set is created in the same layer group as the source layer (tile set).

request_confirmation (bool | None, default: True)
Whether to wait for confirmation and parse the response.

Returns

TileSetCreateInfo
Information about the created or retrieved tile set if confirmation is requested.

None
Returned if confirmation is not requested.

Notes

This function sends a request to stdout to get or create an output tile set with the specified parameters. If confirmation is requested, it waits and reads the response from stdin.

Create tile set

`create_tile_set()`

This function sends a request to stdout to create a new tile set and prepare it for acquisition.

Wrapper for **CreateTileSet** response.
The same rules and behaviors described for CreateTileSet also apply to this function.

```
def create_tile_set(
    tile_set_name: str,
    stage_position: Tuple[float | str, float | str, float | str],
    total_size: Tuple[float | str, float | str],
    rotation: float | str | None = None,
    template_name: str | None = None,
    tile_resolution: Tuple[int, int] | None = None,
    tile_hfw: float | str | None = None,
    pixel_size: float | str | None = None,
    schedule_acquisition: bool | None = None,
    target_layer_group_name: str | None = None,
    request_confirmation: bool | None = True
) -> TileSetCreateInfo | None:
```

Arguments

tile_set_name (str)
Name of the tile set to create.

stage_position (Tuple[float | str, float | str, float | str])
Stage position as a tuple in the form **(X, Y, R)**, where X and Y are stage linear coordinates and R is stage rotation.

total_size (Tuple[float | str, float | str])
Total size of the tile set, specified as width and height.

rotation (float | str | None, default: None)
Optional rotation value of the tile set.

template_name (str | None, default: None)
Optional template name. If not specified, the active tile set template is used (the one selected in Maps).

tile_resolution (Tuple[int, int] | None, default: None)
Optional tile resolution to override the resolution from the template.

tile_hfw (float | str | None, default: None)
Optional horizontal field width for tiles to override the value from the template.

pixel_size (float | str | None, default: None)
Optional pixel size to override the value from the template.

schedule_acquisition (bool | None, default: None)
Whether to schedule acquisition.

target_layer_group_name (str | None, default: None)
Optional target layer group name. If not specified, the tile set is created in the same layer group as the source layer (tile set).

request_confirmation (bool | None, default: True)
Whether to wait for confirmation and parse the response.

Returns

TileSetCreateInfo
Information about the created tile set if confirmation is requested.

None
Returned if confirmation is not requested.

Get layer information

`get_layer_info()`

This function sends a request to stdout to retrieve information about a project layer with the given name.

Wrapper for **GetLayerInfo** response.
The same rules and behaviors described for GetLayerInfo also apply to this function.

```
def get_layer_info(
        layer_name: str,
        request_full_info: bool | None = False
    ) -> LayerInfo:
```

Arguments

layer_name (str)
The name of the layer to query.

request_full_info (bool | None, default: False)
Whether to request full information about the layer.

Returns

LayerInfo
The parsed LayerInfo object if the request is successful.

None
Returned if the request fails or the layer information cannot be retrieved.

Create image layer

`create_image_layer()`

This function sends a request to stdout to create an image layer. The layer will be based on an import of specified image.

Wrapper for **CreateImageLayer** response.
The same rules and behaviors described for CreateImageLayer also apply to this function.

```
def create_image_layer(
    layer_name: str,
    image_file_path: str,
    stage_position: Tuple[float | str, float | str, float | str] | None = None,
    pixel_position: Tuple[int, int] | None = None,
    total_size: Tuple[float | str, float | str] | None = None,
    total_width: float | str | None = None,
    pixel_size: float | str | None = None,
    rotation: float | str | None = None,
    target_layer_group_name: str | None = None,
    keep_file: bool | None = False,
    align_to_source_layer: bool | None = True,
    request_confirmation: bool | None = True
) -> ImageLayerCreateInfo | None:
```

Arguments

layer_name (str)
Name of the image layer to create.

image_file_path (str)
Full path to the image file to use for the layer.

stage_position (Tuple[float | str, float | str, float | str] | None, default: None)
Optional stage position as a tuple in the form **(X, Y, R)**, where X and Y are stage linear coordinates and R is stage rotation. If not specified, the stage position of the source layer is used.

pixel_position (Tuple[int, int] | None, default: None)
Optional pixel position as a tuple of integers (X, Y). The position specifies the created image center relative to the source image. Point (0, 0) is at the top-left corner of the source image.

total_size (Tuple[float | str, float | str] | None, default: None)
Optional total size of the image layer (width, height). This parameter is kept for backward compatibility.

total_width (float | str | None, default: None)
Optional total width of the image layer.

pixel_size (float | str | None, default: None)
Optional pixel size of the image layer.

rotation (float | str | None, default: None)
Optional rotation value of the image layer. If not specified, the rotation of the source layer is used.

target_layer_group_name (str | None, default: None)
Optional target layer group name. If not specified, the image layer is created in the same layer group as the source layer.

keep_file (bool | None, default: False)
Whether to keep the source image file after processing.

align_to_source_layer (bool | None, default: True)
Whether to align the new image layer to the source image layer. Valid only if position is not specified or specified with pixel_position.

request_confirmation (bool | None, default: True)
Whether to wait for confirmation and parse the response.

Returns

ImageLayerCreateInfo
Information about the created image layer if confirmation is requested.

None
Returned if confirmation is not requested.

Create or alter channel

`create_channel()`

This function sends a request to stdout to create a channel, including its name, color, and whether it is additive.
If confirmation is requested, it waits for and returns a Confirmation object.
If a channel with the same name already exists, it will be reused and modified instead of creating a new one.

Wrapper for **CreateChannel** response.
The same rules and behaviors described for CreateChannel also apply to this function.

```
def create_channel(
    channel_name: str,
    channel_color: Tuple[int, int, int] | None = (255, 255, 255),
    is_additive: bool | None = False,
    target_tile_set_guid: uuid.UUID | None = None,
    request_confirmation: bool | None = True
) -> Confirmation | None:
```

Arguments

channel_name (str)
Name of the channel to create.

channel_color (Tuple[int, int, int] | None, default: (255, 255, 255))
RGB color tuple for the channel.

is_additive (bool | None, default: False)
Whether the channel is additive.

target_tile_set_guid (uuid.UUID | None, default: None)
Optional GUID of the target tile set. If not specified, the channel is created in the source tile set.

request_confirmation (bool | None, default: True)
Whether to wait for confirmation and parse the response.

Returns

Confirmation
Result of the channel creation operation if confirmation is requested.

None
Returned if confirmation is not requested.

Create annotation

`create_annotation()`

This function sends a request to stdout to create an annotation, including its name, position, size, color, and other properties. If confirmation is requested, it reads the response from stdin and returns an AnnotationCreateInfo object describing the result.

Wrapper for **CreateAnnotation** response.
The same rules and behaviors described for CreateAnnotation also apply to this function.

```
def create_annotation(
    annotation_name: str,
    stage_position: Tuple[float | str, float | str, float | str],
    rotation: str | None = 0,
    size: Tuple[float | str, float | str] | None = None,
    notes: str | None = "",
    color: Tuple[int, int, int] | None = None,
    is_ellipse: bool | None = False,
    target_layer_group_name: str | None = None,
    request_confirmation: bool | None = True
) -> AnnotationCreateInfo | None:
```

Arguments

annotation_name (str)
Name of the annotation to create.

stage_position (Tuple[float | str, float | str, float | str])
Stage position as a tuple in the form **(X, Y, R)**, where X and Y are stage linear coordinates and R is stage rotation.

rotation (str | None, default: 0)
Optional rotation value of the annotation.

size (Tuple[float | str, float | str] | None, default: None)
Optional size of the annotation.

notes (str | None, default: None)
Optional notes for the annotation.

color (Tuple[int, int, int] | None, default: None)
Optional RGB color tuple for the annotation.

is_ellipse (bool | None, default: False)
Whether the annotation is an ellipse.

target_layer_group_name (str | None, default: None)
Optional target layer group name. If not specified, the annotation is created in the same layer group as the source layer.

request_confirmation (bool | None, default: True)
Whether to wait for confirmation and parse the response.

Returns

AnnotationCreateInfo
Information about the created annotation if confirmation is requested.

None
Returned if confirmation is not requested.

Send single tile output

`send_single_tile_output()`

This function constructs a JSON request describing the output operation for a single tile, including its position, channel, and file path, and sends it to stdout. If confirmation is requested, it waits for and returns a Confirmation object. This function also notifies Maps that the processing of the specified tile is complete.

Note: If multiple outputs for a tile need to be sent, modify this function to accept an array of file outputs.

Wrapper for **TileOutput** response.
The same rules and behaviors described for TileOutput also apply to this function.

```
def send_single_tile_output(
    tile_row: int,
    tile_column: int,
    target_channel_name: str,
    image_file_path: str,
    keep_file: bool | None = False,
    target_tile_set_guid: uuid.UUID | None = None,
    request_confirmation: bool | None = True
) -> Confirmation | None:
```

Arguments

tile_row (int)
The row index of the tile. Indexing is 1-based.

tile_column (int)
The column index of the tile. Indexing is 1-based.

target_channel_name (str)
The name of the target channel. If the channel does not exist, a new channel with that name is created automatically.

image_file_path (str)
The full file path of the image to output.

keep_file (bool | None, default: False)
Whether to keep the source image file after processing.

target_tile_set_guid (uuid.UUID | None, default: None)
Optional GUID of the target tile set. If not specified, the source tile set is used.

request_confirmation (bool | None, default: True)
Whether to wait for confirmation.

Returns

Confirmation
Result of the tile output operation if confirmation is requested.

None
Returned if confirmation is not requested.

Store file

`store_file()`

This function sends a request as JSON to stdout to store a file, optionally overwriting an existing file, keeping the file after processing, and associating it with a specific layer. If confirmation is requested, it waits for and returns a Confirmation object.

Wrapper for **StoreFile** response.
The same rules and behaviors described for StoreFile also apply to this function.

```
def store_file(
    file_path: str,
    overwrite: bool | None = False,
    keep_file: bool | None = True,
    target_layer_guid: uuid.UUID | None = None,
    request_confirmation: bool | None = True
) -> Confirmation | None:
```

Arguments

file_path (str)
Full path to the file to be stored.

overwrite (bool | None, default: False)
Whether to overwrite the file if it exists. If not overwriting, the new file is renamed with a (n) suffix.

keep_file (bool | None, default: True)
Whether to keep the file after storing.

target_layer_guid (uuid.UUID | None, default: None)
Optional GUID of the target layer. If not specified, the file is stored in the source layer.

request_confirmation (bool | None, default: True)
Whether to wait for confirmation.

Returns

Confirmation
Result of the store file operation if confirmation is requested.

None
Returned if confirmation is not requested.

Append notes

`append_notes()`

This function constructs a request to append notes to a layer, identified by its GUID. The request is sent as JSON to stdout. If confirmation is requested, it waits for and returns a Confirmation object.

Wrapper for **AppendNotes** response.
The same rules and behaviors described for AppendNotes also apply to this function.

```
def append_notes(
    notes_to_append: str,
    target_layer_guid: uuid.UUID | None = None,
    request_confirmation: bool | None = True
) -> Confirmation | None:
```

Arguments

notes_to_append (str)
The notes text to append.

target_layer_guid (uuid.UUID | None, default: None)
Optional GUID of the target layer. If not specified, notes are appended to the source layer.

request_confirmation (bool | None, default: True)
Whether to wait for confirmation.

Returns

Confirmation
Result of the append notes operation if confirmation is requested.

None
Returned if confirmation is not requested.

Report progress

`report_progress()`

This function constructs a JSON response containing a progress report and sends it to stdout.

Wrapper for **ReportProgress** response.
The same rules and behaviors described for **ReportProgress** also apply to this function.

```
def report_progress(progress_percentage: float):
```

Arguments

progress_percentage (float)
The progress percentage to report (0.0 to 100.0).

Report activity description

`report_activity_description()`

This function constructs a JSON response containing an activity description and sends it to stdout.
The activity description is shown in the processing job details in Maps.

Wrapper for **ReportActivity** response.
The same rules and behaviors described for **ReportActivity** also apply to this function.

```
def report_activity_description(activity_description: str):
```

Arguments

activity_description (str)
The description of the activity to report.

Report failure

`report_failure()`

This function constructs a JSON response indicating an error, including the provided message, and sends it to stdout. **This will fail and terminate the scripting process**.

Wrapper for **ReportFailure** response.
The same rules and behaviors described for **ReportFailure** also apply to this function.

```
def report_failure(error_message: str):
```

Arguments

error_message (str)
The error message to report.

Log information, warning,error

`log_info()`
`log_error()`
`log_warning()`

This functions constructs a JSON response containing an informational, warning or error message to be logged and sends it to stdout.

Wrappers for Log response.
The same rules and behaviors described for Log also apply to these functions.

```
def log_info(info_message: str):
def log_warning(warning_message: str):
def log_error(error_message: str):
```

Arguments

info_message (str)
The informational message to log.

Arguments

The info, warning or error message to log.

Read Maps start request

`read_request_from_stdin()`

This is a helper function that reads a JSON string from standard input (stdin), determines the request type, and constructs the corresponding request object. It can return a ScriptTileSetRequest, ScriptImageLayerRequest, or a general ScriptRequest.

Use this function at the start of the script to clear the standard input and retrieve information about the requested action, the source layer and script parameters.

```
def read_request_from_stdin() -> ScriptTileSetRequest | ScriptImageLayerRequest | ScriptRequest | None:
```

Returns

ScriptTileSetRequest | ScriptImageLayerRequest | ScriptRequest | None
The constructed request object, or None if the request type is unknown.

Get tile information

`get_tile_info()`

This helper function retrieves the **TileInfo** object in the specified tile set.

```
def get_tile_info(tile_column: int, tile_row: int, tile_set: TileSetInfo) -> TileInfo | None:
```

Arguments

tile_column (int)
The column index of the tile (1-based).

tile_row (int)
The row index of the tile (1-based).

tile_set (TileSetInfo)
The tile set containing the tiles.

Returns

TileInfo
The TileInfo object for the specified tile if found.

None
Returned if the tile is not found.

Tile pixel to stage tranformation

`tile_pixel_to_stage()`

This helper function converts tile image image pixel coordinates to absolute physical stage coordinates using the tile set transformation matrix. Pixel coordinate (0, 0) is at the top-left corner of the tile image.

```
def tile_pixel_to_stage(pixel_x: int, pixel_y: int,
       tile_column: int, tile_row: int, tile_set: TileSetInfo) -> PointFloat:
```

Arguments

pixel_x (int)
The X coordinate in the tile image pixel space.

pixel_y (int)
The Y coordinate in the tile image pixel space.

tile_column (int)
The column index of the tile (1-based).

tile_row (int)
The row index of the tile (1-based).

tile_set (TileSetInfo)
The tile set containing geometry and transformation information.

Returns

PointFloat
The corresponding stage coordinates as a PointFloat.

Image pixel to stage transformation

`image_pixel_to_stage()`

This helper function converts image pixel coordinates to absolute physical stage coordinates using the image layer's transformation matrix. Pixel coordinate (0, 0) is at the top-left corner of the image.

```
def image_pixel_to_stage(pixel_x: int, pixel_y: int, image_layer: ImageLayerInfo) -> PointFloat:
```

Arguments

pixel_x (int)
The X coordinate in the image pixel space.

pixel_y (int)
The Y coordinate in the image pixel space.

image_layer (ImageLayerInfo)
The image layer containing geometry and transformation information.

Returns

PointFloat
The corresponding stage coordinates as a PointFloat.

Calculate total pixel offset in a tile

`calculate_total_pixel_position()`

This helper function calculates the total pixel offset from the tile set center for a pixel coordinate in a tile image space.

```
def calculate_total_pixel_position(
       pixel_x: int, pixel_y:
       int, tile_column: int, tile_row: int,
       tile_set: TileSetInfo) -> PointInt:
```

Arguments

pixel_x (int)
The X coordinate in the tile image pixel space.

pixel_y (int)
The Y coordinate in the tile image pixel space.

tile_column (int)
The column index of the tile (1-based).

tile_row (int)
The row index of the tile (1-based).

tile_set (TileSetInfo)
The tile set containing geometry and transformation information.

Returns

PointInt
The total pixel offset in the tile set as a PointInt.

Appendix

Usage of PixelToStageMatrix

This matrix is a 2D transformation from pixel coordinates in the image space to an absolute stage coordinates. Use it to determine the stage position from tile, tile set or image layer image.

PixelToStageMatrix structure is an array of the matrix members:

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

Example:

```
"PixelToStageMatrix":
[
        [2.929687E-07, 0, 0],
        [0, 2.929687E-07, 0],
        [-0.012195525216850297, 0.0035056840776182996, 1]
]
```

In case of tile images:

The matrix is always defined relative to the center of the entire tile set. To calculate the absolute pixel offset from the center for a tile image, use the tile's **TileCenterPixelOffset**.

Commented example:

```
# These are pixel coords within the tile image, where [0,0] is at top left
 pixel_x
 pixel_y

 # Get tile resolution
 tile_resolution = tile_set.tile_resolution

 # Calculate absolute pixel offset from the tile set center
 # Note the opposite operators Y axic calculation!
 # Y axis on the stage is opposite to pixel coords
 x = tile.tile_center_pixel_offset.x - (tile_resolution.width / 2) + pixel_x
 y = -tile.tile_center_pixel_offset.y + (tile_resolution.height / 2) - pixel_y

 # Read the matrix
 m = tile_set.pixel_to_stage_matrix

 # Transform the coordinates with the matrix
 # The resulting absolute position on the stage in meters
 stage_x = x * m[0][0] + y * m[1][0] + 1 * m[2][0]
 stage_y = x * m[0][1] + y * m[1][1] + 1 * m[2][1]
```

Also see `image_pixel_to_stage()` in MapsBridge library.

In case of image layers:

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

Commented example:

```
# These are pixel coords within the image layer image, where [0,0] is at top left
pixel_x
pixel_y

# Calculate pixel offset from the layer center
# Note the opposite operators in Y axic calculation!
# Y axis on the stage is opposite to pixel coords
x = -image_layer.total_layer_resolution.width / 2 + pixel_x
y = image_layer.total_layer_resolution.height / 2 - pixel_y

# Read the matrix
m = image_layer.pixel_to_stage_matrix

# Transform the coordinates with the matrix
# The resulting absolute position on the stage in meters
stage_x = x * m[0][0] + y * m[1][0] + 1 * m[2][0]
stage_y = x * m[0][1] + y * m[1][1] + 1 * m[2][1]
```

Also see `image_pixel_to_stage()` in MapsBridge library.

Examples of Python scripts

Examples use MapsBridge.py library and can be found in *<Maps installation folder>\Script Bridge\Examples*

TileSet_SingleTiles_AllDemo.py

This script demonstrates a tile set processing script, executed for each tile in the tile set (Single tiles mode). It can be run manually or during acquisition.

TileSet_Batch_IterationDemo.py

This script demonstrates iteration of tiles in a batch tile set processing (Batch mode). It can be run manually or When acquisition completed

ImageLayer_AllDemo.py

This script demonstrates an image layer processing script. It can be run manually.