



Version 1.5

# Plug-in your own AI

Programmers' Manual



## Table of Contents

<b>1. Overview .....</b>	<b>3</b>
<b>2. Getting started / Prepare virtual Python Environment.....</b>	<b>4</b>
<b>3. Launch Script Service .....</b>	<b>4</b>
<b>4. Adding a Plugin to the MIKAIA® App Center .....</b>	<b>5</b>
<b>5. MIKAIA Python Console .....</b>	<b>7</b>
<b>6. Example Plugins .....</b>	<b>8</b>
6.1     ApiExamplePlugin.py.....	8
6.2     TensorFlowClassificationPlugin.py .....	8
<b>7. How to debug a Plugin .....</b>	<b>8</b>
<b>8. Instantiating client-side Python API .....</b>	<b>9</b>
<b>9. Analyzing a ROI, FoV or the entire Slide .....</b>	<b>9</b>
<b>10. Full API documentation.....</b>	<b>10</b>
<b>11. Contact Information.....</b>	<b>11</b>

## 1. Overview

The *MIKAIA® Plug-in your own AI App* enables users to plug in their own Python AI scripts and make them available as a discrete App in the App Center.

*MIKAIA®* is an whole-slide-image analysis software developed by Fraunhofer IIS and available for download from [www.mikaia.ai](http://www.mikaia.ai).

While *MIKAIA®* is only available for Windows, a python plugin script can run

- on the same PC as MIKAIA
- in a docker
- or on a remote computer, even on a Linux or Mac.

Multiple plugins residing on the same or on multiple computers can be plugged into *MIKAIA®* at the same time.

In order for *MIKAIA®* to be able to detect the script, the *MIKAIA® Script Service*, written in Python and open sourced, must be launched on the same computer where the script resides. The Script Service serves two purposes:

- 1) It discovers Python scripts in its workspace and, when asked by MIKAIA, reports their file names.
- 2) When the MIKAIA user starts an analysis, MIKAIA will contact the Script Service and ask it to launch the local Python plugin, passing along the URL under which MIKAIA can be reached

The Script Service is written in Python and is available open source. It can be started on any platform (we have tested on Windows and Ubuntu).

In the *Plug-in your own AI App*'s configuration pane, a plugin developer can add a new plugin configuration and then specify the IP address and port where the *Script Service* can be reached.

In order to discover a remote script, MIKAIA will then call the *Script Service* and ask it for a list of the locally detected Python scripts (\*.py), which it will retrieve by scanning it's work space.

The plugin developer then select the plugin's main \*.py file. Additionally, they can enter a description text and pick an icon that represents the plugin in the *MIKAIA®* App Center. Subsequently, when a *MIKAIA®* user wants to analyze a ROI, FoV or entire slide with the new plugin, *MIKAIA®* will again call the *Script Service* and ask it to launch the configured local script.

It will pass a *Session ID* along as a command line parameter as well as *MIKAIA®*'s own IP address and port.

The Python script can then directly interact with the REST API exposed by *MIKAIA®* without having to go through the *Script Service*.

*MIKAIA®* currently exposes one REST API - the ***Slide Service*** - that can be used to

- retrieve metadata on the currently opened whole-slide-image (WSI)
- retrieve pixels for a desired ROI (stated in µm) and at a desired resolution (stated in µm/pixel)
- read in existing annotation classes and annotations
- generate new annotation classes and annotations

## 2. Getting started / Prepare virtual Python Environment

You should have the MIKAIA-Plug-in-your-own-AI-API.zip. Extract it to a local folder, e.g., C:\my\_mikaia\_plugin.

- 1) Open a console (on Linux, a gnome-terminal is required.)
- 2) We recommend to set up a virtual python environment:

```
python -m venv C:\my_mikaia_plugin-venv
C:\my_mikaia_plugin-venv\Scripts\activate
```

- 3) Install the MIKAIA client API

```
pip install C:\my_mikaia_plugin\mikaia_plugin_api-0.9.4-py3-none-any.whl
```

- 4) (optional) The TensorFlowClassificationPlugin.py example additionally requires the TensorFlow module

```
pip install tensorflow
```

## 3. Launch Script Service

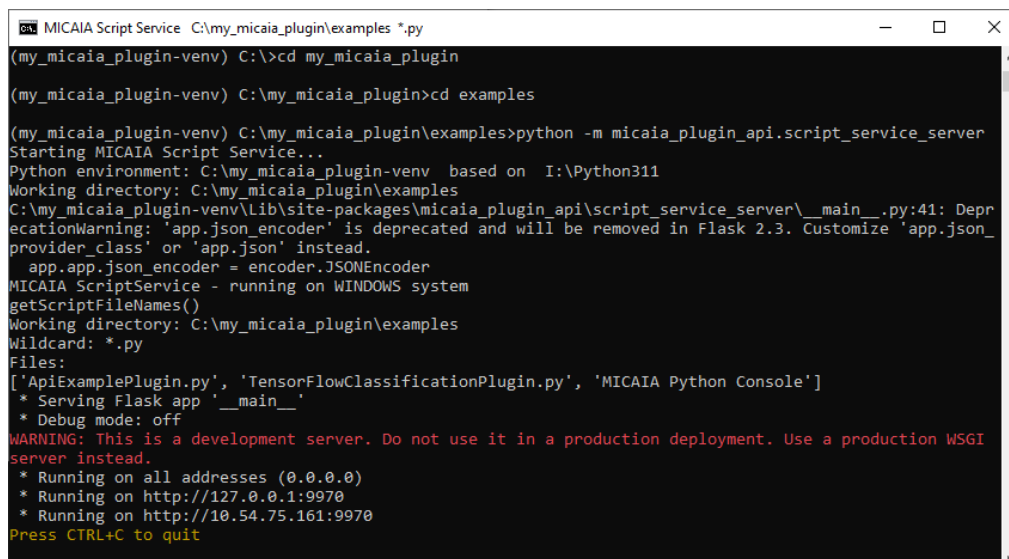
Now that the virtual Python environment is prepared, the Script Service can be started:

- 5) Now launch the Script Service.

The directory from where the Script Service is started will become the workspace that is scanned for \*.py plugins.

```
cd C:\my_mikaia_plugin\examples
python -m mikaia_plugin_api.script_service_server
```

The Script Service will detect the Python plugins “ApiExamplePlugin.py” and “TensorFlowClassificationPlugin.py” and additionally always list the built-in special **MIKAIA Python Console**



```

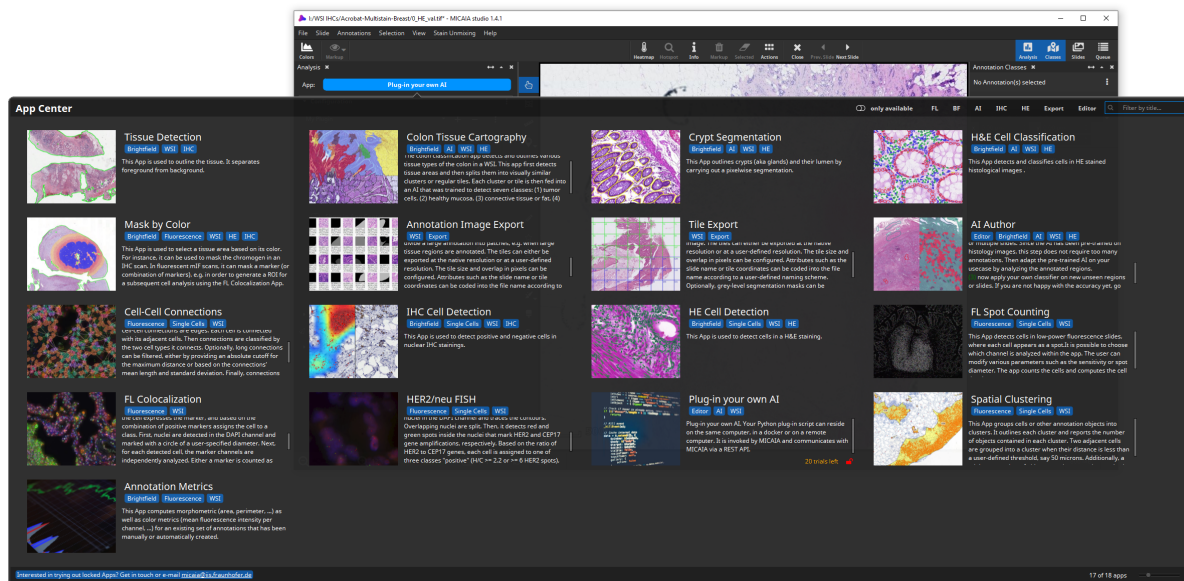
MIKAIA Script Service C:\my_mikaia_plugin\examples *.py
(my_mikaia_plugin-venv) C:\>cd my_mikaia_plugin
(my_mikaia_plugin-venv) C:\my_mikaia_plugin>cd examples
(my_mikaia_plugin-venv) C:\my_mikaia_plugin\examples>python -m mikaia_plugin_api.script_service_server
Starting MIKAIA Script Service...
Python environment: C:\my_mikaia_plugin-venv based on I:\Python311
Working directory: C:\my_mikaia_plugin\examples
C:\my_mikaia_plugin-venv\Lib\site-packages\mikaia_plugin_api\script_service_server\_main_.py:41: DeprecationWarning: 'app.json_encoder' is deprecated and will be removed in Flask 2.3. Customize 'app.json_provider_class' or 'app.json' instead.
  app.app.json_encoder = encoder.JSONEncoder
MIKAIA ScriptService - running on WINDOWS system
getScriptFileNames()
Working directory: C:\my_mikaia_plugin\examples
Wildcard: *.py
Files:
['ApiExamplePlugin.py', 'TensorFlowClassificationPlugin.py', 'MIKAIA Python Console']
* Serving Flask app '_main_'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:9970
* Running on http://10.54.75.161:9970
Press CTRL+C to quit

```

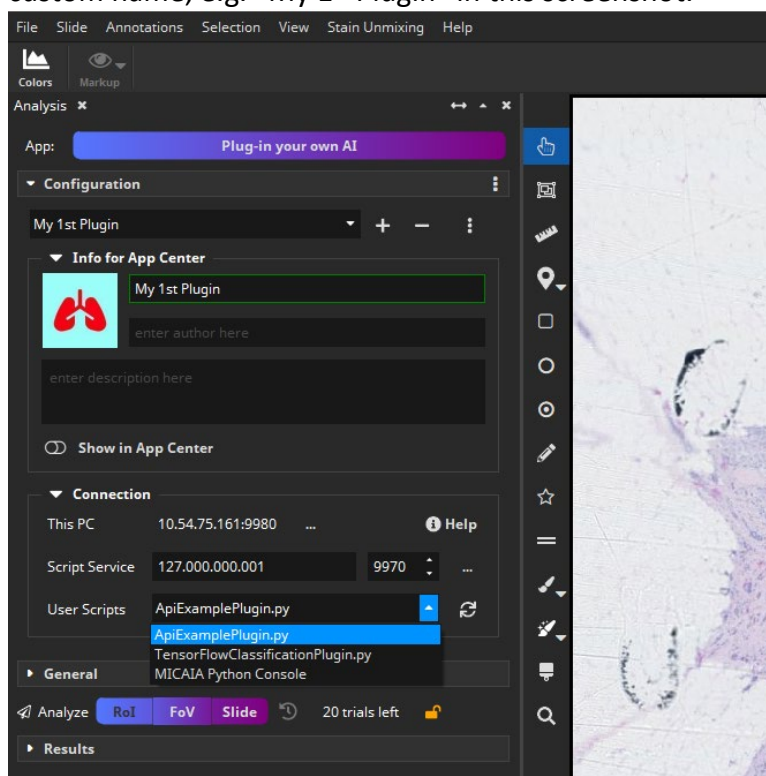
The console window is now occupied with the Script Service. It must stay open for the plugin to work.

## 4. Adding a Plugin to the MIKAI® App Center


- 1) Open MIKAI®
- 2) Load any slide
- 3) Open the “Analysis” sidebar by clicking the **Analysis** button in the main toolbar on the far right
- 4) In the App Center, select the **Plug-in your own AI App**

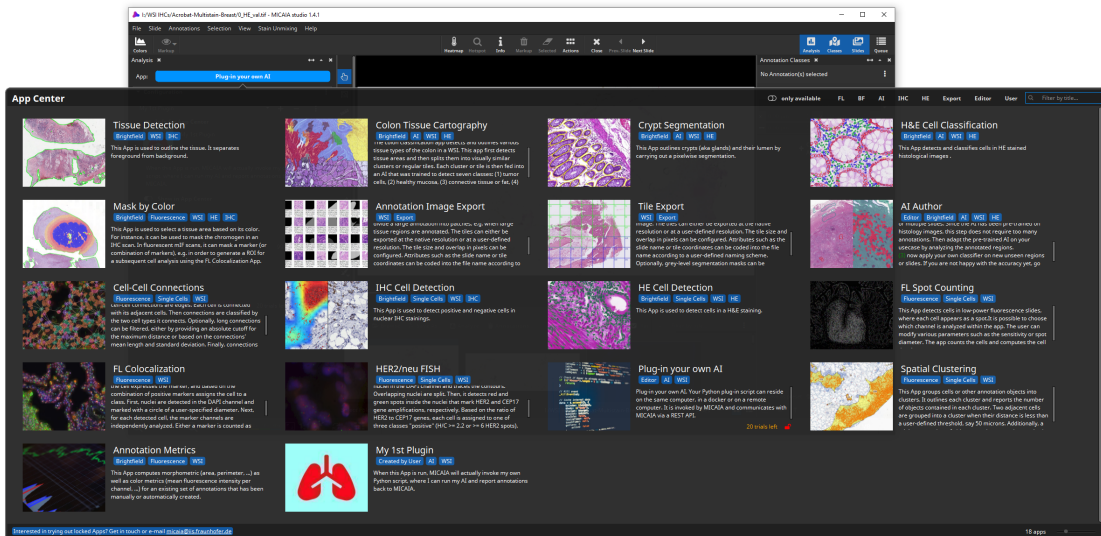


- 5) Press the + button to add a new plugin. A dialog will pop up, where you can enter a custom name, e.g. “My 1<sup>st</sup> Plugin” in this screenshot.





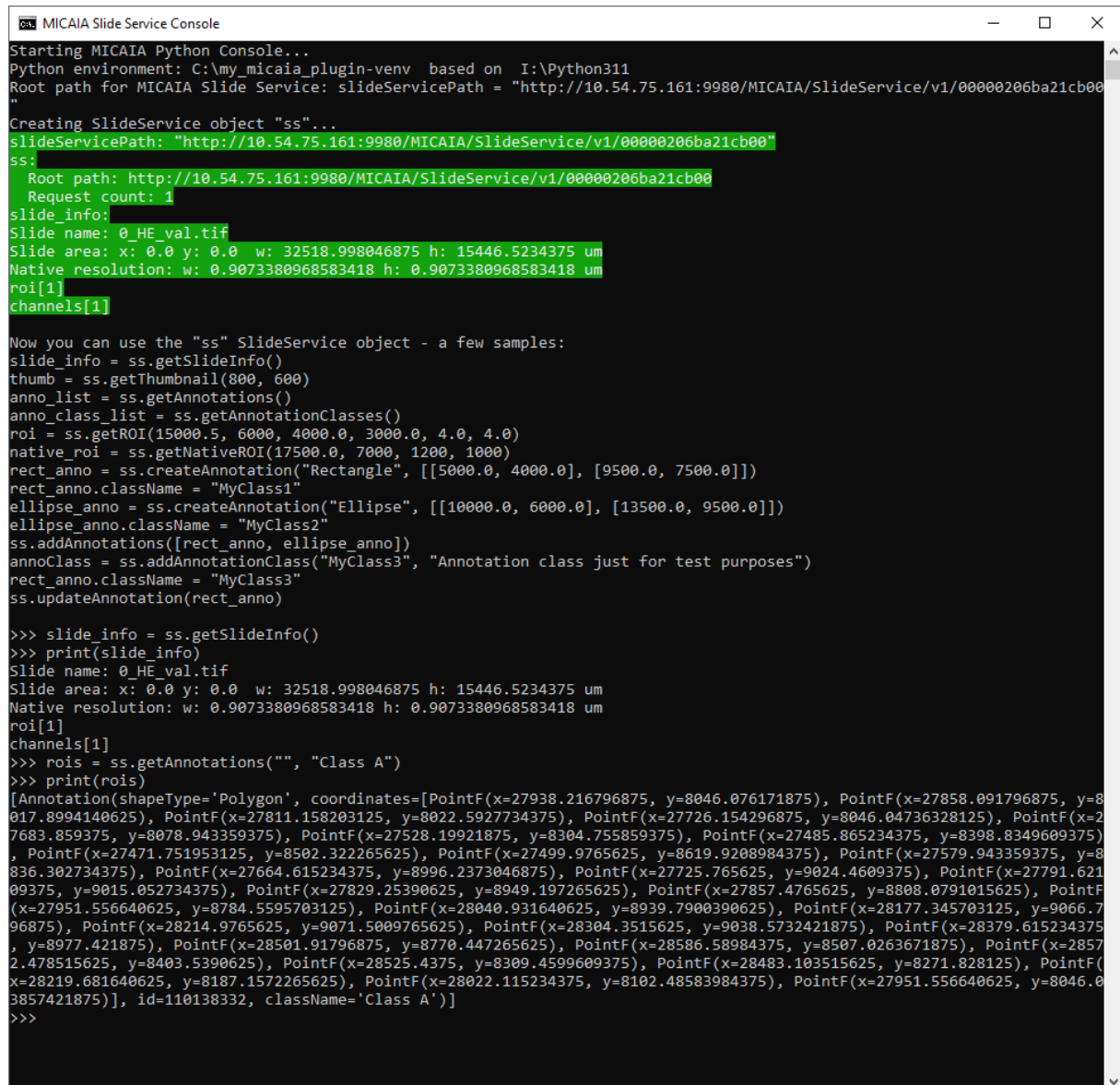
- 6) Then enter the IP address and port of the Script Service that is already running. If the Script Service was started on the same computer as MIKAIA, then the default values 127.0.0.1 and port 9970 are correct.
- 7) Press the  button to retrieve a list of available Python scripts from the Script Service. The list of discovered scripts will then be added to the **User Scripts** combo box. Additionally, the built-it **MIKAIA Python Console** is always available. Select your plugin.
- 8) In the Info for App Center section, enter a description text, select an icon or image and then toggle the Show in App Center switch. A new App is now available in the App Center:



## 5. MIKAIA Python Console

When the MIKAIA Python Console is selected in the User Scripts combo box and an analysis is subsequently started, then, instead of invoking a Python script, a Python terminal windows is opened.

In the terminal, it is possible to use the client-side Python commands and test the Slide Service API:



```

MICAIASlideService Console
Starting MIKAIA Python Console...
Python environment: C:\my_micaia_plugin-venv based on I:\Python311
Root path for MIKAIA Slide Service: slideServicePath = "http://10.54.75.161:9980/MICAIA/SlideService/v1/00000206ba21cb00"

Creating SlideService object "ss"...
slideServicePath: "http://10.54.75.161:9980/MICAIA/SlideService/v1/00000206ba21cb00"
ss:
  Root path: http://10.54.75.161:9980/MICAIA/SlideService/v1/00000206ba21cb00
  Request count: 1
slide_info:
  Slide name: 0_HE_val.tif
  Slide area: x: 0.0 y: 0.0 w: 32518.998046875 h: 15446.5234375 um
  Native resolution: w: 0.9073380968583418 h: 0.9073380968583418 um
roi[1]
channels[1]

Now you can use the "ss" SlideService object - a few samples:
slide_info = ss.getSlideInfo()
thumb = ss.getThumbnail(800, 600)
anno_list = ss.getAnnotations()
anno_class_list = ss.getAnnotationClasses()
roi = ss.getROI(15000.5, 6000, 4000.0, 3000.0, 4.0, 4.0)
native_roi = ss.getNativeROI(17500.0, 7000, 1200, 1000)
rect_anno = ss.createAnnotation("Rectangle", [[5000.0, 4000.0], [9500.0, 7500.0]])
rect_anno.className = "MyClass1"
ellipse_anno = ss.createAnnotation("Ellipse", [[10000.0, 6000.0], [13500.0, 9500.0]])
ellipse_anno.className = "MyClass2"
ss.addAnnotations([rect_anno, ellipse_anno])
annoClass = ss.addAnnotationClass("MyClass3", "Annotation class just for test purposes")
rect_anno.className = "MyClass3"
ss.updateAnnotation(rect_anno)

>>> slide_info = ss.getSlideInfo()
>>> print(slide_info)
Slide name: 0_HE_val.tif
Slide area: x: 0.0 y: 0.0 w: 32518.998046875 h: 15446.5234375 um
Native resolution: w: 0.9073380968583418 h: 0.9073380968583418 um
roi[1]
channels[1]
>>> rois = ss.getAnnotations("", "Class A")
>>> print(rois)
[Annotation(shapeType='Polygon', coordinates=[PointF(x=27938.216796875, y=8046.076171875), PointF(x=27858.091796875, y=8017.8994140625), PointF(x=27811.158203125, y=8022.5927734375), PointF(x=27726.154296875, y=8046.04736328125), PointF(x=27683.859375, y=8078.943359375), PointF(x=27528.19921875, y=8304.755859375), PointF(x=27485.865234375, y=8398.8349609375), PointF(x=27471.751953125, y=8502.322265625), PointF(x=27499.9765625, y=8619.9208984375), PointF(x=27579.943359375, y=8836.302734375), PointF(x=27664.615234375, y=8996.2373046875), PointF(x=27725.765625, y=9024.4609375), PointF(x=27791.62109375, y=9015.052734375), PointF(x=27829.25390625, y=8949.197265625), PointF(x=27857.4765625, y=8808.0791015625), PointF(x=27951.556640625, y=8784.5595703125), PointF(x=28040.931640625, y=8939.7900390625), PointF(x=28177.345703125, y=9066.796875), PointF(x=28214.9765625, y=9071.5009765625), PointF(x=28304.3515625, y=9038.5732421875), PointF(x=28379.615234375, y=8977.421875), PointF(x=28501.91796875, y=8770.447265625), PointF(x=28586.58984375, y=8507.0263671875), PointF(x=28572.478515625, y=8403.5390625), PointF(x=28525.4375, y=8300.4599609375), PointF(x=28483.103515625, y=8271.828125), PointF(x=28219.681640625, y=8187.1572265625), PointF(x=28022.115234375, y=8102.48583984375), PointF(x=27951.556640625, y=8046.03857421875)], id=110138332, className='Class A')]
>>>

```

The MIKAIA Python Console serves as an easy means to try out the client-side Python API and interact with MIKAIA.

## 6. Example Plugins

The examples subfolder in the zip contains two example plugins:

### 6.1 ApiExamplePlugin.py

This script does not perform a meaningful image analysis, but rather shows how to use most of the client-side API calls. The user is prompted to press the “Enter” key after every command.

### 6.2 TensorFlowClassificationPlugin.py

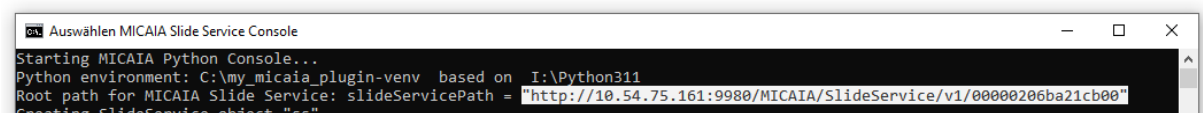
This script loads a TensorFlow CNN model that classifies image patches. It was trained on H&E stained colon sections to recognize multiple tissue classes, including. “tumor”, “epithelium”, inflammation”, etc.. The model was not trained exhaustively but serves its purpose for this example.

The model is contained in the zip archive in the subfolder examples\zoo\.

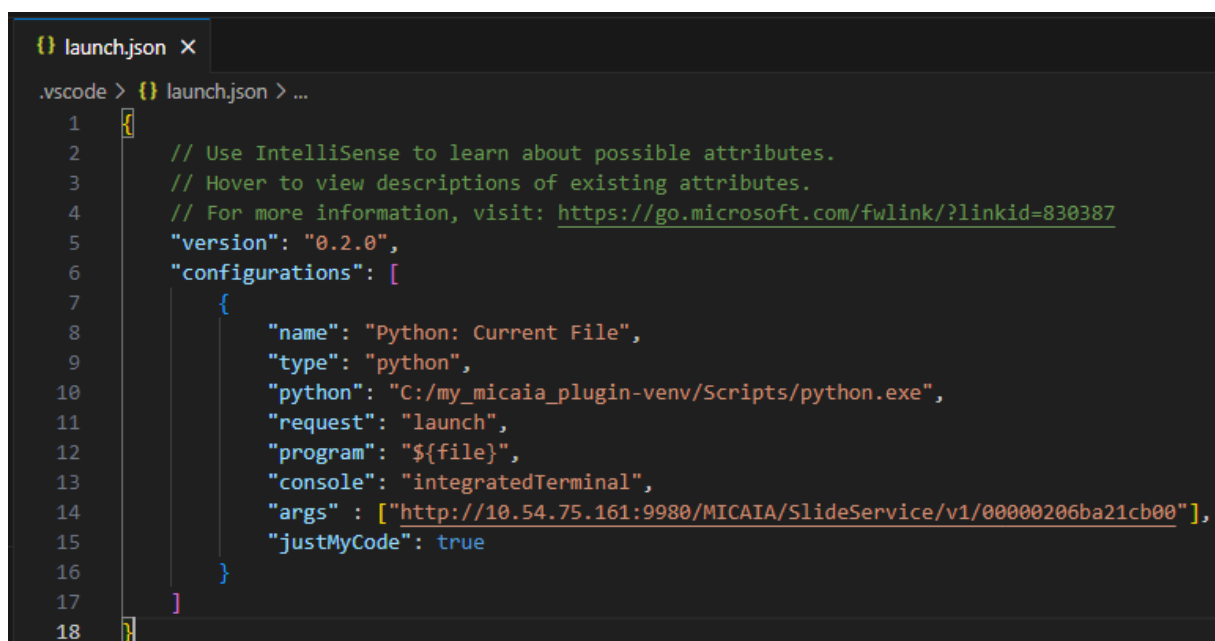
## 7. How to debug a Plugin

In the regular workflow, the Script Service launches the Python script and so debugging is not straight forward.

One way to attach a debugger is to configure MIKAIA to open the MIKAIA Python Console instead of the plugin. The programmer can then copy the URL incl. the session ID from the Python Console window and launch their plugin manually in their favorite IDE, passing the URL as the first command line argument.



In Visual Studio Code this can be done by first creating a launch.json configuration:





In this example, two lines were manually added:

```
"python": "C:/my_mikaia_plugin-venv/Scripts/python.exe",
```

tells VS Code to use the previously set up virtual environment (see chapter 2).

```
"args" : ["http://10.54.75.161:9980/MIKAIA/SlideService/v1/00000206ba21cb00"],
```

tells VS Code to pass the URL and session ID that was copied from the MIKAIA Python Console to the script as a command line argument.

## 8. Instantiating client-side Python API

A plugin will typically run the following commands initially

- 1) Retrieve the SlideService URL and session ID

```
import sys
from mikaia_plugin_api import mikaia_api

#[...]
def main():
    slideServiceUrl = sys.argv[1]
```

- 2) Create a SlideService client object

```
mikaia = mikaia_api.SlideService(slideServiceUrl)
```

- 3) Query infos on the currently opened slide and the ROIs that shall be analyzed

```
slideInfo = mikaia.getSlideInfo()
rois = slideInfo.roi # if empty, analyze entire slide
```

If the user selects to analyze the current field of view, it will be passed as a single ROI.

## 9. Analyzing a ROI, FoV or the entire Slide

The user starts an analysis by pressing either the ROI, FoV or Slide button.

<b>ROI</b>	Analyze all currently selected annotations
<b>FoV</b>	Analyze the currently visible region of the slide
<b>Slide</b>	Analyze the entire slide (usually the Tissue Detection is carried out first)

The plugin is expected to comply with this convention. Which mode the user selected can be queried from the SlideInfo object.

```
slideInfo = mikaia.getSlideInfo()
rois = slideInfo.roi # if empty, analyze entire slide
```

## 10. Full API documentation

The API documentation is available in a separate document  
**MIKAIA plug-in-your-own-AI API Documentation.pdf**

## 11. Contact Information

For more information in general and contact information please visit our web page  
[www.mikaia.ai](http://www.mikaia.ai)



MIKAIA® is developed by Fraunhofer IIS.  
With questions regarding custom developments,  
please get in touch via  
[mikaia@iis.fraunhofer.de](mailto:mikaia@iis.fraunhofer.de).

Further information is available at  
<https://www.iis.fraunhofer.de/digitalpathology>