MIKAIA Plug-in your own AI - API Documentation

Table of contents

Introduction

The **Plug-in your own AI** App provides a way for connecting custom Python scripts into **MIKAIA**. They will appear as separate Apps in the App Center. The communication between the plugin and MIKAIA is done via a REST API. MIKAIA opens a server and makes the Slide Service available. Plugins can use it to retrieve metadata or pixels of the currently opened slide and generate annotations. A more detailed description of the concept and architecture is given in the separate **MIKAIA Plug-in your own AI - Programmer's Manual**.

This document lists all API endpoints. The **mikaia_api** module is the central module of the **mikaia_plugin_api** package.

License

mikaia_api

The **mikaia_api** module is the central module of the **mikaia_plugin_api** package.
It provides classes and functions to communicate with the **MIKAIA** Slide Service.

Class SlideService

Implements a MIKAIA SlideService client.

Initialization

```
SlideService(slidePath)
```

- **slidePath: string**
  SlideService root path.
  If the **MIKAIA** Software invokes a python script, the SlideService root path is passed as the first argument to the script.

Python code examples

```
from mikaia_plugin_api import mikaia_api

# The SlideService root path is passed as the first argument to the script
slideServicePath = sys.argv[1]

# Create an SlideService interface object to access MIKAIA SlideService
ss = mikaia_api.SlideService(slideServicePath)

# Request slide info from MIKAIA SlideService
slideInfo = ss.getSlideInfo()
print(slideInfo)
...
```

Method SlideService.getSlideInfo

```
SlideService.getSlideInfo(log = False)
```

Returns basic informations about the slide or image that is just loaded in the **MIKAIA** Software.

Parameters

- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

Returns a mikaia_api.SlideInfo object.

Python code examples

```
# Request slide info from MIKAIA SlideService
slideInfo = ss.getSlideInfo()
print(slideInfo)
```

Method SlideService.getUserParameters

```
SlideService.getUserParameters(log = False)
```

Returns user parameters defined in the **MIKAIA** Software.

MIKAIA provides a text input field where users can enter additional parameters in the format **Key = Value**. This function returns these user parameters as a dictionary of key/value pairs.

Parameters

- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

Returns the user parameters as a dictionary of key/value pairs.

Python code examples

```
# Request optional user parameters from MIKAIA SlideService
userParameters = ss.getUserParameters()
print(userParameters)
```

Method SlideService.sendProgress

```
SlideService.sendProgress(progress_0to1, progress_amount = 0, progress_message =
"", log = False)
```

Sends a progress notification to the **MIKAIA** Software to update its progress bar.
Progress can be reported as a normalized value between 0.0(0%) and 1.0(100%) or as a total amount of progress if a normalized value cannot be provided. The optional progress message is displayed in the progress bar of the **MIKAIA** Software.

Parameters

- **progress_0to1: float**
  A normalized progress value between 0.0(0%) and 1.0(100%).
- **progress_amount: int**
  A total progress amount. Should be used if a normalized value cannot be provided.
- **progress_msg: string**
  Optional progress message displayed in the progress bar of the **MIKAIA** Software.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

None.

Python code examples

```
# Send progress notification to MIKAIA SlideService
ss.sendProgress(0.05, 0, "Loading TensorFlow model...")
```

Method SlideService.sendMessage

```
SlideService.sendMessage(message, log = False)
```

Sends a message to the **MIKAIA** Software.
MIKAIA writes this message to its script execution log file, but does not display it in the GUI.

Parameters

- **message: string**
  Message to be logged in **MIKAIA**'s script execution log file.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

None.

Python code examples

```
# Send a log message to MIKAIA SlideService
ss.sendMessage("Tile size: {} x {} pixels".format(tile_width, tile_height))
```

Method SlideService.getThumbnail

```
SlideService.getThumbnail(max_width = 512, max_height = 512, log = False)
```

Get thumbnail image of the slide.

Parameters

- **max_width: boolean**
  maximum width of the thumbnail image in pixels.
- **max_height: boolean**
  maximum height of the thumbnail image in pixels.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

Returns the thumbnail image as (PIL-)image.

Python code examples

```
# Request thumbnail image from MIKAIA SlideService and display it
thumbnail = ss.getThumbnail(800, 600, False)
thumbnail.show()
```

Method SlideService.getROI

```
SlideService.getROI(x_um, y_um, w_um, h_um, px_width_um, px_height_um = 0,
px_format = 'BGR', channel_idx = -1, log = False)
```

Get a rectangular ROI of the slide as (PIL-)image.

Parameters

- **x_um, y_um: float**
  location of the ROI(top left corner, in microns(um)).
- **w_um, h_um: float**
  width and height of the ROI in microns(um).
- **px_width_um, px_height_um: float**
  desired pixel resolution of the returned ROI image in microns(um)/pixel.
- **px_format: string**
  desired pixel format of the ROI image(one of: 'BGR', 'RGB' or 'Gray').
  Default value is 'RGB'.
- **channel_idx: int**
  index of a pixel data channel. Default is -1(use all channels).
  Useful to extract a certain pixel data channel from a fluorescence slide.
  Information about available pixel data channels is listed in the **channels** array of the **mikaia_api.SlideInfo** class.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

Returns the ROI image as (PIL-)image.

Python code examples

```
# Request a ROI as grayscale image from MIKAIA SlideService and display it
roi = ss.getROI(15000.5, 6000, 4000.0, 3000.0, 4.0, 4.0, 'Gray')
roi.show()
```

Method SlideService.getNativeROI

```
SlideService.getNativeROI(x_um, y_um, w_px, h_px, px_format = 'BGR', channel_idx =
-1, log = False)
```

Get a rectangular ROI of the slide as (PIL-)image.
The returned image has the native slide pixel resolution(as returned from the **getSlideInfo()** method

Parameters

- **x_um, y_um: float**
  location of the ROI image(top left corner, in microns(um)).
- **w_px, h_px: int**
  width and height of the ROI image in pixels.
- **px_format: string**
  desired pixel format of the ROI image(one of: 'BGR', 'RGB' or 'Gray').
  Default value is 'RGB'.
- **channel_idx: int**
  index of a pixel data channel. Default is -1(use all channels).
  Useful to extract a certain pixel data channel from a fluorescence slide.
  Information about available pixel data channels are listed in the **channels** array of the **mikaia_api.SlideInfo** class.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

Returns the ROI image as (PIL-)image.

Python code examples

```
# Request a ROI in native resolution from MIKAIA SlideService and display it
native_roi = ss.getNativeROI(17500.0, 7000, 1200, 1000, 'RGB', -1, False)
native_roi_.show()
```

Method SlideService.getAnnotations

```
SlideService.getAnnotations(shape_type = "", class_name = "", log = False)
```

Get a list of annotation items from the MIKAIA SlideService.

Parameters

- **shape_type: string**
  Shape filter. If provided, only annotations of the specified shape type are returned.
- **class_name: string**
  Class name filter. If provided, only annotations which belong to the specified annotation class are returned.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

Array of mikaia_api.Annotation objects.

Python code examples

```
# request all annotations from MIKAIA SlideService and print them
anno_list = ss.getAnnotations()
print(f"{len(anno_list)} annotations received:")
for item in anno_list:
    print(item)

# request all rectangle annotations from MIKAIA SlideService
anno_list = ss.getAnnotations('Rectangle')

# request all rectangle annotations of annotation class 'ROI' from MIKAIA
SlideService
anno_list = ss.getAnnotations('Rectangle', 'ROI')
```

Method SlideService.addAnnotation

```
SlideService.addAnnotation(shape_type, coordinates, holes = [], class_name = "",
log = False)
```

Creates an mikaia_api.Annotation instance from given parameters and adds it to the slide.

Parameters

- **shape_type: string**
  Shape type of the annotation. Supported values: 'Point', 'Line', 'Rectangle', 'Ellipse', 'Polygon', 'PathWithHoles'.
- **coordinates: List**
  Python list of 2D coordinates. The content depends on the shape type:
  'Point': Exactly one Point [[x1, y1]]
  'Line': Start point and end point [[x1, y1], [x2, y2]]
  'Rectangle': top left and bottom right coordinate [[x1, y1], [x2, y2]]
  'Ellipse': top left and bottom right coordinate of the bounding box [[x1, y1], [x2, y2]]
  'Polygon': polygon points [[x1, y1], [x2, y2], ... , [xn, yn]]
- **holes: List**
  Python list of 2D coordinate Lists. Holes are only relevant for shape type 'PathWithHoles':\
- **class_name: string**
  Optional name of a annotation class to which the annotation should be assigned.
  If a class_name is provided and such a annotation class doesn't exist, a new annotation class with this name will be created automatically.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

Created mikaia_api.Annotation object.

Python code examples

```
# create some annotations objects
rect_anno = ss.addAnnotation('Rectangle', [[1000.0, 2000.0], [6000.5, 4500.99]])
ellipse_anno = ss.addAnnotation('Ellipse', [[1000.0, 10000.0], [6000.5,
14500.99]])
poly_anno = ss.addAnnotation('Polygon', [[10000.0, 5000.5], [12000.3, 3000.5],
[15000.5, 8500.99], [13000.3, 5500.5], [10500.3, 7000.5]])

# create a 'PathWithHoles' annotation
face_outline = [[0.0, 0.0], [100.0, 0.0], [90.0, 100.0], [10.0, 100.0]]
left_eye = [[15.0, 12.0], [27.0, 12.0], [27.0, 24.0], [15.0, 24.0]]
right_eye = [[73.0, 12.0], [85.0, 12.0], [85.0, 24.0], [73.0, 24.0]]
nose = [[50.0, 18.0], [58.0, 43.0], [42.0, 43.0]]
```

```
mouth = [[20.0, 55.0], [85.0, 55.0], [75.0, 85.0], [60.0, 90.0], [45.0, 90.0],
[30.0, 85.0]]
holes = [left_eye, right_eye, nose, mouth]
face_anno = ss.addAnnotation('PathWithHoles', face_outline, holes)
```

Method SlideService.addAnnotations

```
SlideService.addAnnotations(annotation_list, log = False)
```

Add one or more mikaia_api.Annotation objects to the slide. **Note:** Use method SlideService.createAnnotation to create instances of mikaia_api.Annotation.

Parameters

- **annotation_list: List[mikaia_api.Annotation]**
  List of mikaia_api.Annotation objects.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value: List[mikaia_api.Annotation]

List of mikaia_api.Annotation objects.

Python code examples

```
# create some annotations and add them to the slide
rect_anno = ss.createAnnotation('Rectangle', [[1000.0, 2000.0], [6000.5,
4500.99]])
ellipse_anno = ss.createAnnotation('Ellipse', [[1000.0, 10000.0], [6000.5,
14500.99]])
poly_anno = ss.createAnnotation('Polygon', [[10000.0, 5000.5], [12000.3, 3000.5],
[15000.5, 8500.99], [13000.3, 5500.5], [10500.3, 7000.5]])
anno_list = ss.addAnnotations([rect_anno, ellipse_anno, poly_anno])
```

Method SlideService.updateAnnotation

```
SlideService.updateAnnotation(annotation, log = False)
```

Updates the content of an already existing mikaia_api.Annotation object. Currently only the 'className' is supported by this update operation.

Parameters

- **annotation: mikaia_api.Annotation**
  mikaia_api.Annotation object to update.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

True on success, **False** otherwise.

Python code examples

```
# request all annotation objects of annotation class 'Class One' and change their
associated class to 'Class Two'
anno_list = ss.getAnnotations("", "Class One")
for anno in anno_list:
  anno.className = 'Class Two'
  ss.updateAnnotations(anno)
```

Method SlideService.getAnnotationClasses

```
SlideService.getAnnotationClasses(log = False)
```

Get a list of all mikaia_api.AnnotationClass items of the slide.

Parameters

- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

Array of mikaia_api.AnnotationClass objects.

Python code examples

```
# Request all specified annotation classes and print them
anno_class_list = ss.getAnnotationClasses()
[print(item) for item in anno_class_list]
```

Method SlideService.addAnnotationClass

```
SlideService.addAnnotationClass(class_name, description = "", line_width_px = -1,
line_color = "", fill_color = "", opacity = 1.0, log = False)
```

Add a mikaia_api.AnnotationClass object to the slide.

Parameters

- **class_name: string**
  name of the annotation class .
- **description: string**
  Optional description of the annotation class.
- **line_width_px: int**
  Optional outline width(in screen pixels) for annotations associated with this annotation class.
- **line_color: string**
  Optional outline color for annotations associated with this annotation class.
  RGB color definition as hexadecimal HTML '#AARRGGBB' color string(e.g. '#ffc280de').
- **fill_color: string**
  Optional fill color for annotations associated with this annotation class.
  RGB color definition as hexadecimal HTML '#AARRGGBB' color string(e.g. '#ffa260be').
- **opacity: float**
  Optional opacity for annotations associated with this annotation class(0.0(fully transparent) to 1.0(fully opaque)).
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

The created mikaia_api.AnnotationClass instance.

Python code examples

```
# create a new annotation class object without explicit style properties and add
it to the slide.
# The style properties are applied implicitely by the MIKAIA application.
annoClass = ss.addAnnotationClass('Test-Class', 'Annotation class just for test
purposes')
print(anno_class)

# create a new annotation class object with explicit style properties and add it
to the slide.
annoClass = ss.addAnnotationClass('Unclassified', 'Unclassified annotations', 3,
'#ff808080de', '#ffb0b0b0', 0.33)
print(anno_class)
```

Method SlideService.addAnnotationClasses

```
SlideService.addAnnotationClasses(annotation_class_list, log = False)
```

Add one or more mikaia_api.AnnotationClass objects to the slide.
Note: Use method SlideService.createAnnotationClass to create instances of mikaia_api.AnnotationClass.

Parameters

- **annotation_class_list: List[mikaia_api.AnnotationClass]**
  List of mikaia_api.AnnotationClass objects.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value: List[mikaia_api.AnnotationClass]

List of added mikaia_api.AnnotationClass objects.

Python code examples

```
# create some annotation classes and add them to the slide
annoClass0 = ss.createAnnotationClass('Unclassified', 'Unclassified annotations',
2, '#ff909090')
annoClass1 = ss.createAnnotationClass('Class One', 'Annotations of class One', 5,
'#FFAABB00')
annoClass2 = ss.createAnnotationClass('Class Two', 'Annotations of class Two', 3,
'#ff00bbaa', '#ff009080', 0.2 )
ss.addAnnotationClasses([annoClass0, annoClass1, annoClass2])
```

Method SlideService.updateAnnotationClass

```
SlideService.updateAnnotation(annotation, log = False)
```

Updates the content of an already existing mikaia_api.AnnotationClass object.
Currently the following attributes of class mikaia_api.AnnotationClass are supported:

- classDescription
- outlineWidth
- outlineColor
- fillColor
- opacity

Parameters

- **annotation: mikaia_api.AnnotationClass**
  mikaia_api.AnnotationClass object to update.
- **log: boolean**
  If set to 'True', additional output(for debugging) will be generated.

Return value

True on success, **False** otherwise.

Python code examples

```
# Change line width and line color of annotation class 'Class One'
annoClassList = ss.getAnnotationClasses()
for annoClass in annoClassList:
  if annoClass.className == 'Class One':
    annoClass.outlineWidth = 3
    annoClass.outlineColor = '#ffaa00bb'
    ss.updateAnnotationClass(annoClass)
```

Method SlideService.createAnnotation

```
SlideService.createAnnotation(shape_type, coordinates, holes = [], class_name =
"")
```

Creates a mikaia_api.Annotation instance from given parameters.

Parameters

- **shape_type: string**
  Shape type of the annotation. Supported values: 'Point', 'Line', 'Rectangle', 'Ellipse', 'Polygon', 'PathWithHoles'.
- **coordinates: List**
  Python list of 2D coordinates. The content depends on the shape type:
  'Point': Exactly one Point [[x1, y1]]
  'Line': Start point and end point [[x1, y1], [x2, y2]]
  'Rectangle': top left and bottom right coordinate [[x1, y1], [x2, y2]]
  'Ellipse': top left and bottom right coordinate of the bounding box [[x1, y1], [x2, y2]]
  'Polygon': polygon points [[x1, y1], [x2, y2], ... , [xn, yn]]
  'PathWithHoles': outline path as polygon points [[x1, y1], [x2, y2], ... , [xn, yn]]
- **holes: List**
  Python list of 2D coordinate Lists. Holes are only relevant for shape type 'PathWithHoles':
- **class_name: string**
  Name of the associated annotation class(empty if there is no annotation class associated).

Return value

Created mikaia_api.Annotation object.

Python code examples

```
# create some naitive annotations objects
rect_anno = ss.createAnnotation('Rectangle', [[1000.0, 2000.0], [6000.5,
4500.99]])
ellipse_anno = ss.createAnnotation('Ellipse', [[1000.0, 10000.0], [6000.5,
14500.99]])
poly_anno = ss.createAnnotation('Polygon', [[10000.0, 5000.5], [12000.3, 3000.5],
[15000.5, 8500.99], [13000.3, 5500.5], [10500.3, 7000.5]])

# create a 'PathWithHoles' annotation
face_outline = [[0.0, 0.0], [100.0, 0.0], [90.0, 100.0], [10.0, 100.0]]
left_eye = [[15.0, 12.0], [27.0, 12.0], [27.0, 24.0], [15.0, 24.0]]
right_eye = [[73.0, 12.0], [85.0, 12.0], [85.0, 24.0], [73.0, 24.0]]
nose = [[50.0, 18.0], [58.0, 43.0], [42.0, 43.0]]
mouth = [[20.0, 55.0], [85.0, 55.0], [75.0, 85.0], [60.0, 90.0], [45.0, 90.0],
[30.0, 85.0]]
```

```
holes = [left_eye, right_eye, nose, mouth]
face_anno = ss.createAnnotation('PathWithHoles', face_outline, holes)
```

Method SlideService.createAnnotationClass

```
SlideService.createAnnotationClass(class_name, description = ""))
```

Creates a mikaia_api.AnnotationClass instance from given parameters.

Parameters

- **class_name: string**
  Name of the annotation class.
- **description: string**
  Description of the annotation class(optional).
- **line_width_px: int**
  Optional outline width(in screen pixels) for annotations associated with this annotation class.
- **line_color: string**
  Optional outline color for annotations associated with this annotation class.
  RGB color definition as hexadecimal HTML '#AARRGGBB' string(e.g. '#ffc280de').
- **fill_color: string**
  Optional fill color for annotations associated with this annotation class(optional).
  RGB color definition as hexadecimal HTML '#AARRGGBB' string(e.g. '#ffa260be').
- **opacity: float**
  Optional opacity for annotations associated with this annotation class(0.0(fully transparent) to 1.0(fully opaque)).

Return value

Created mikaia_api.AnnotationClass object.

Python code examples

```
# create some annotation classes and add them to the slide
annoClass0 = ss.createAnnotationClass('Unclassified', 'Unclassified annotations',
2, '#ff909090')
annoClass1 = ss.createAnnotationClass('Class One', 'Annotations of class One', 5,
'#FFAABB00')
annoClass2 = ss.createAnnotationClass('Class Two', 'Annotations of class Two', 3,
'#ff00bbaa', '#ff009080', 0.2 )
ss.addAnnotationClasses([annoClass0, annoClass1, annoClass2])
```

Class PointF

2D point with floating point coordinates.

Class variables

- **x: float**
  x-coordinate in microns(um).
- **y: float**
  y-coordinate in microns(um).

Python code examples

```
# Create 2D point object and print it
pt = mikaia_api.PointF(12.0, 5.5)
print(pt)
```

Class SizeF

2D size with floating point dimensions.

Class variables

- **width: float**
  Width in microns(um).
- **height: float**
  Height in microns(um).

Python code examples

```
# Create 2D size object and print it
size = mikaia_api.SizeF(100.0, 50.0)
print(size)
```

Class RectF

2D rectangle with floating point coordinates/dimensions.

Class variables

- **x: float**
  x-coordinate of the top left rectangle corner in microns(um).
- **y: float**
  y-coordinate of the top left rectangle corner in microns(um).
- **width: float**
  Rectangle width in microns(um).
- **height: float**
  Rectangle height in microns(um).

Python code examples

```
# Create 2D rectangle object and print it
rect = mikaia_api.RectF(12.0, 5.5, 300.0, 150.0)
print(rect)
```

Class Annotation

Data class that represents an annotation object of the **MIKAIA** Software.
The following annotation types are supported: **Point**, **Line**, **Rectangle**, **Ellipse**, **Polygon**, **PathWithHoles**.
Use the SlideService.createAnnotation method to create annotation objects

Class variables

- **shapeType: string**
  Shape type of the annotation. Supported values: 'Point', 'Line', 'Rectangle', 'Ellipse', 'Polygon', 'PathWithHoles'.
- **coordinates: List[List[float]]**
  Coordinates of the annotation shape(outline and optional holes) as lists of 2D point coordinates.
  Each coordinates list is a flat array of 2D point coordinates(in microns(µm)) in the format [x1, y1, x2, y2, ... xn, yn].
  The 1st point list contains the coordinates of the outline contour.
  All further point lists describe the contours of holes lying inside the outline.
  Holes are only relevant for shapeType 'PathWithHoles'.
  All other shapes types consist of exactly one point list with the following content:
  'Point': [x, y] the point coordinates.
  'Line': [x1, y1, x2, y2]' start point and end point of the line.
  'Rectangle': [x1, y1, x2, y2]' top left and bottom right coordinates of the rectangle.
  'Ellipse': [x1, y1, x2, y2]' top left and bottom right coordinates of the bounding rectangle.
  'Polygon': [x1, y1, ... xn, yn]' coordinates of n polygon points.\
- **id: int**
  MIKAIA-ID of the annotation. Don't change the value.
- **className: string**
  Name of the associated annotation class(empty if there is no annotation class associated).

Python code examples

```
# Request all rectangle annotations associated with class 'User Annotation'
anno_list = ss.getAnnotations('Rectangle', 'User Annotation')
[print(item) for item in anno_list]

# Usage of method SlideService.createAnnotation() to create annotation objects
rect_anno = ss.createAnnotation('Rectangle', [[1000.0, 2000.0], [6000.5,
4500.99]])
ellipse_anno = ss.createAnnotation('Ellipse', [[1000.0, 10000.0], [6000.5,
14500.99]])
poly_anno = ss.createAnnotation('Polygon', [[10000.0, 5000.5], [12000.3, 3000.5],
[15000.5, 8500.99], [13000.3, 5500.5], [10500.3, 7000.5]])

# create a 'PathWithHoles' annotation
face_outline = [[0.0, 0.0], [100.0, 0.0], [90.0, 100.0], [10.0, 100.0]]
left_eye = [[15.0, 12.0], [27.0, 12.0], [27.0, 24.0], [15.0, 24.0]]
right_eye = [[73.0, 12.0], [85.0, 12.0], [85.0, 24.0], [73.0, 24.0]]
nose = [[50.0, 18.0], [58.0, 43.0], [42.0, 43.0]]
```

```
mouth = [[20.0, 55.0], [85.0, 55.0], [75.0, 85.0], [60.0, 90.0], [45.0, 90.0],
[30.0, 85.0]]
holes = [left_eye, right_eye, nose, mouth]
face_anno = ss.createAnnotation('PathWithHoles', face_outline, holes)
```

Method Annotation.toTuples

```
Annotation.toTuples()
```

Returns the annotation coordinates as lists of (x, y) tuples:
[[(x1, y1), (x2, y2), ... (xn, yn)], [(x1, y1), (x2, y2), ... (xm, ym)], ...

Parameters

- **None**

Return value

Annotation coordinates as lists of (x, y) tuples.

Python code examples

```
# Request annotations from MIKAIA SlideService and convert the annotation
coordinates
anno_list = ss.getAnnotations()
coordinates_as_tuples = anno_list[0].toTuples()
print(coordinates_as_tuples)
```

Method Annotation.toArrays

```
Annotation.toArrays()
```

Returns the annotation coordinates as lists of [x, y] arrays:
[[[x1, y1], [x2, y2], ... [xn, yn]], [[x1, y1], [x2, y2], ... [xm, ym]], ...]

Parameters

- **None**

Return value

Annotation coordinates as lists of [x, y] arrays.

Python code examples

```
# Request annotations from MIKAIA SlideService and convert the annotation
coordinates
anno_list = ss.getAnnotations()
coordinates_as_arrays = anno_list[0].toArrays()
print(coordinates_as_arrays)
```

Method Annotation.boundingRect()

```
SlideService.boundingRect(index = 0)
```

Returns the bounding rectangle of an annotation contour(outline contour or hole contour) as mikaia_api.RectF instance.

Parameters

- **index: int**
  index of the annotation contour(0: outline contour, > 0: hole contour).

Return value

Returns the bounding rectangle of the specified annotation contour.

Python code examples

```
# Get bounding rectangles of all annotation contours(outline contour or hole
contours)
boundingRects = []
for index in range(0, len(anno_item.coordinates)):
  boundingRects.append(anno_item.boundingRect(index))
```

Class AnnotationClass

Data class that represents an annotation class object of the **MIKAIA** Software.
Use the SlideService.createAnnotationClass method to create annotation class objects

Class variables

- **className: string**
  Name of the annotation class.
- **classDescription: string**
  Description of the annotation class(optional, may be empty).
- **line_width_px: int**
  Outline width(in screen pixels) for annotations associated with this annotation class.
- **line_color: string**
  Outline color for annotations associated with this annotation class.
  RGB color definition as hexadecimal HTML '#AARRGGBB' color string(e.g. '#ffc280de').
- **fill_color: string**
  Fill color for annotations associated with this annotation class.
  RGB color definition as hexadecimal HTML '#AARRGGBB' color string(e.g. '#ffa260be').
- **opacity: float**
  Opacity for annotations associated with this annotation class(0.0(fully transparent) to 1.0(fully opaque)).
- **id: int**
  MIKAIA-ID of the annotation class. Don't change the value.

Python code examples

```
# Request all specified annotation classes from MIKAIA SlideService and print them
anno_class_list = ss.getAnnotationClasses()
[print(item) for item in anno_class_list]

# create some annotation classes and add them to the slide
annoClass0 = ss.createAnnotationClass('Unclassified', 'Unclassified annotations',
2, '#ff909090')
annoClass1 = ss.createAnnotationClass('Class One', 'Annotations of class One', 5,
'#FFAABB00')
annoClass2 = ss.createAnnotationClass('Class Two', 'Annotations of class Two', 3,
'#ff009080', '#ff00bbaa', 0.5 )
ss.addAnnotationClasses([annoClass0, annoClass1, annoClass2, annoClassFace])
```

Class SlideInfo

Data class that covers some basic informations about the slide or image that is just loaded in the **MIKAIA** Software.

Class variables

- **name: string**
  Name of the loaded slide.
- **slideRect: mikaia_api.RectF**
  Location and size of the slide. The unit of these slide coordinates is always microns(um).
- **nativeResolution: SizeF**
  Native pixel resolution in microns(um).
  Defines pixel width and height of images of the highest resolution level of the slide.
- **roi: List[mikaia_api.Annotation]**
  List of region of interest(ROI) to analyze.
  By default it is the whole slide area, but it can be also a list of user defined annotations(rectangles or polygons).
- **roi: List[mikaia_api.ChannelInfo]**
  List of available pixel data channels of the slide.

Python code examples

```
# Request slide info from MIKAIA SlideService and print content
slideInfo = ss.getSlideInfo()
print(slideInfo)
print(slideInfo.roi[0])
print(slideInfo.channels[0])
...
```

Class ChannelInfo

Data class that covers some basic informations about a slide pixel data channel.

Class variables

- **name: string**
  Name of the pixel data channel(e.g. 'Brightfield', 'DAPI', 'Cy5', ...).
- **type: string**
  Type of the pixel data channel(One of: 'Brightfield', 'Fluorescence', 'Other', 'Unspecified').
- **index: int**
  Channel index.
  Use this index to select the pixel data channel when reading a ROI image from the slide.

Python code examples

```
# Request slide info from MIKAIA SlideService and show first entry of channels
list
slideInfo = ss.getSlideInfo()
print(slideInfo.channels[0])
...
```