
clickpoints Documentation

Release 1.1

Richard Gerum, Sebastian Richter

May 15, 2017

Contents

1	Installation	3
1.1	Windows	3
1.2	Linux	4
1.3	Mac	4
2	General	5
2.1	Zooming, Panning, Rotating	6
2.2	Jumping frames	6
2.3	Interfaces	6
3	Tutorial	7
3.1	Opening Files and Folders	7
3.2	Using ConfigFiles	8
3.3	Manual Tracking	9
3.4	Taking Measurements	12
4	Modules	15
4.1	Timeline	15
4.2	GammaCorrection	17
4.3	VideoExporter	19
4.4	Annotations	19
4.5	Marker	20
4.6	Mask	26
4.7	Info Hud	28
5	Add-ons	31
5.1	Tracking	31
5.2	Drift Correction	32
5.3	Cell Detector	32
5.4	Grab Plot Data	32
6	Examples	35
6.1	Count Penguins	35
6.2	Flourescence intensities in plant roots	36
6.3	Supervised Tracking of Fiducial Markers in Magnetic Tweezer Measurements	38
6.4	Using ClickPoints for Visualizing Simulation Results	39

7	Database API	41
7.1	Database Models	41
7.2	DataFile	46
8	Add-on API	65
8.1	GetCommandLineArgs	65
8.2	Commands	66
9	Citing ClickPoints	69
10	Note	71
	Bibliography	73



Click Points is a program written in the Python programming language, which serves on the one hand as an image viewer and on the other hand as an data display and annotation tool. Every frame can be annotated by a description, marked points/tracks, or marked areas (paint brush). This helps to view image data, do manual evaluation of data, help to create semi-automatic evaluation or display the results of automatic image evaluation.

CHAPTER 1

Installation

ClickPoints can be installed in different ways, you can choose the one which is the most comfortable for you and the operating system you are using.

Windows

Installer

We provide an installer for ClickPoints on Windows 64bit platforms.

Download: [ClickPoints Installer](#)

Additionally, you need an installation of Python with the packages needed by ClickPoints. For convenience we provide a WinPython installation with all the packages already installed. We recommend to use ClickPoints with this Python installation.

Download: [WinPython for ClickPoints Installer](#)

Note: We recommend that you install ClickPoints to C:\Software\ClickPoints, as the Program Files folder requires to always provide admin privileges when modifying any files.

Cutting Edge Version (Mercurial)

If you prefer to have the latest version with the latest features and bugfixes, you can grab the tip version from our mercurial repository.

```
hg clone https://bitbucket.org/fabry_biophysics/clickpoints
```

For adding clickpoints to the right click menu in the file explorer, execute the `install_clickpoints.bat` in the installation folder.

Warning: If you don't use our provided Python installation, you need to install the required packages. Missing packages will throw an ImportError, e.g. `ImportError: No module named peewee`. This means this package is missing and has to be installed. While most packages can be easily installed using pip, unfortunately some packages don't install out of the box with `pip install PACKAGENAME` but wheel files for these packages be obtained for Windows from Christoph Gohlke's [Unofficial Windows Binaries](#) or if you are on Linux from your distributions package repositories, e.g. using `sudo apt install python-PACKAGENAME`

Linux

Download the [Cutting Edge Version \(Mercurial\)](#) and run the `install_bat.py`, which will create a command line command `clickpoints` and add ClickPoints to the menu for right clicking on folders/images in the file browser (e.g. nautilus or dolphin).

Mac

Not yet supported. ClickPoints hasn't been tried on Mac yet and ClickPoints won't be added to the Finder right click menu yet. If you have python already installed, you can refer to the installation from [Cutting Edge Version \(Mercurial\)](#) to try to get it working yourself.


CHAPTER 2


General

Once ClickPoints has been installed it can be started directly from the Start Menu/Program launcher or by running the `ClickPoints.bat`, or respectively `ClickPoints` in Linux.

Attention: If the `ClickPoints.bat` file isn't present in the ClickPoints directory the `install.bat.py` script needs to be executed first.

This will open ClickPoints with an empty project.

Images can be added to the project by using .

The project can be saved by clicking on .

ClickPoints can also be used to directly open images, videos or folder by right clicking on them, which will open an unsaved project which already contains some images. This way ClickPoints functions as an image viewing tool.

ClickPoints can be opened with various files as target:

- an **image**, loading all images in the folder of the target image.
- a **video** file, loading only this video.
- a **folder**, loading all image and video files of the folder and its sub folders, which are concatenated to one single image stream.
- a previously saved `.cdb` **ClickPoints Project** file, loading the project as it was saved.

Pressing `ESC` closes ClickPoints.

To easily access marker, masks, track or other information, stored in the `.cdb` ClickPoints Project file, we provide a python based API

Attention: If you plan to evaluate your data set or continue working on the same data set you must save the project - otherwise all changes will be lost upon closing the program. If a project was saved, all changes are saved automatically upon frame change or by pressing S

Zooming, Panning, Rotating

ClickPoints opens with a display of the current image fit to the window. The display can be

- zoomed, using the mouse wheel
- panned, holding down the right mouse button
- rotated using R.

To fit the image into the window press F and switch to full screen mode by pressing W.

Note: Default rotation on startup or and rotation steps with each press of R can be defined in the `ConfigClickPoints.txt` with the entries `rotation =` and `rotation_steps =`.

Jumping frames

ClickPoints provides various options to change the current frame.

- The keys `Left` and `Right` go to the previous or next frame.
- The keys `Home` and `End` jump to the first or last frame.
- Click or Drag & Drop the timeline slider

Key pairs on the numpad allow for jumps of speciefied

- Numpad 2, Numpad 3: `-/+ 1`
- Numpad 5, Numpad 6: `-/+ 10`
- Numpad 8, Numpad 9: `-/+ 100`
- Numpad /, Numpad *: `-/+ 1000`

Be sure to have the numpad activated, or the keys won't work.

Note: The step size of the jump keys can be redefined by the `jumps = variable` in the `ConfigClickPoints.txt`

For continuous playback of frames see timeline module.

Interfaces

The interfaces for Marker, Mask and GammaCorretion can be shown/hidden pressing F2.

The recipes section contains some basic usage examples on how to get started using ClickPoints and explains different ways for different tasks.

Opening Files and Folders

ClickPoints was designed with multiple usage keys in mind, and therefore provides multiple ways to open files.

Attention: Opening a set of files for the first time can take some time to extract time and meta information from the filename, TIFF or EXIF header. For large collections of files it is recommended to save the collection as a project and use the `.cdb` file for starting ClickPoints. Saving time as no file system search is necessary and all meta information is already stored in the `.cdb`

via Interface

ClickPoints can be started empty by using a desktop link or calling `ClickPoints.bat` from CMD (Windows), or respectively `ClickPoints` from a terminal (Linux).

Images can be added by using **the folder button**.

via Context Menu

A fast and comfortable way to open files and folders with ClickPoints is the context menu.

ClickPoints can be opened with various files as target:

- an **image**, loading all images in the folder of the target image.
- a **video** file, loading only this video.

- a **folder**, loading all image and video files of the folder and its sub folders, which are concatenated to one single image stream.
- a previously saved `.cdb` **ClickPoints Project** file, loading the project as it was saved.

via Commandline Parameter

ClickPoints can be run directly from the commandline, e.g. to open the files in the current or a specific folder

```
ClickPoints "C:\Images"
```

or

```
python ClickPoints.py -srcfile="C:\Images"
```

Note: To use the short version of calling ClickPoints without the path, you have to add ClickPoints base path to the systems or users PATH variable (Windows) or create an alias (Linux).

via .txt File

Furthermore it is possible to supply a text file where each line contains the path to an image or video file. This is useful e.g. to open a fixed set of files, a list of files extract by another application or a database interface.

```
ClickPoints "sample.txt"
```

Listing 3.1: sample.txt

```
1  20120919_colonydensity.gif
   ↪      # relativ path (to txt file)
2  C:\Users\Desktop\images\20160601-141408_GE4000.jpg      # absolut path
3  \\192.168.0.99\2014\20140323\03\20140323-030151_31n2.JPG  # network path
```

Note: It is possible to open files over the network e.g. via samba shares. On Linux systems it is necessary do mount the network drive first!

Using ConfigFiles

The config file contains parameters to adjust ClickPoints to the users needs, adjusts default behaviour or configure key bindings. A GUI is available to set parameters during runtime, the ConfigFile is used to set default behaviour.

Scope

Config files in ClickPoints are designed to optimize the work flow. Upon opening a file the path structure is searched for the first occurrence of a valid config file. Thereby allowing the user to specify default for files grouped in one location.

If no config file is found, the default config values as set in click points base path are used (green). A config file located in the path “research” (blue) will overwrite these values and is used for files opened in child paths. Allowing the user

to define a preferred setup. In addition we can add a second config file lower in the path tree to specify a specific setup for all files that were stored under “Experiment_3”. This can contain a set of default marker names, which features to use or which add-ons to include.

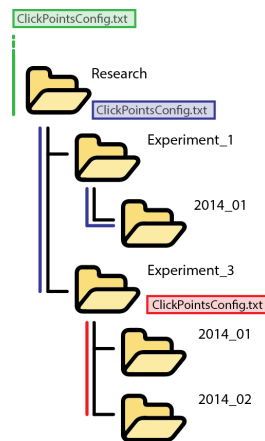


Fig. 3.1: Scope of ConfigFiles

Note: A graphical user interface to view and change config values is available too.

Manual Tracking

This tutorial gives a shot introduction how to get started manually labeling your own tracks, for a quick evaluation or ground truths for the evaluation of automated algorithms.

Getting started

1. Open the image sequence or video(s) in ClickPoints

For example: right click on the folder containing your images and select ClickPoints on the context menu

2. Save the project


Marked results and correlated images must be stored some where, there for the project has to be named

and saved. Click on the save button  and select a storage location and file name.

Note: Reference to images and video is stored relative as long as the files reside parallel or below in the path tree. If the files reside above or on a different branch, drive, or network location, the absolute path is stored.

3. Define Marker types

Before we can get started we have to specify a marker type. Marker types are like classes of objects, e.g. we might use a class for birds and another one for ships. Every marker type can have multiple tracks.

To open the marker menu either press `F2` or click on the Marker button  to switch to edit mode (Fig. A). Then right click onto the marker list to open the marker menu (Fig. B). You can reuse the default marker or create a new marker by selecting `+ add type`. Choose a name and color for your new marker type and make sure to set the type to `TYPE_track`. Confirm your changes by pressing `save`. To add more tracking types select `+ add type` and repeat the procedure.

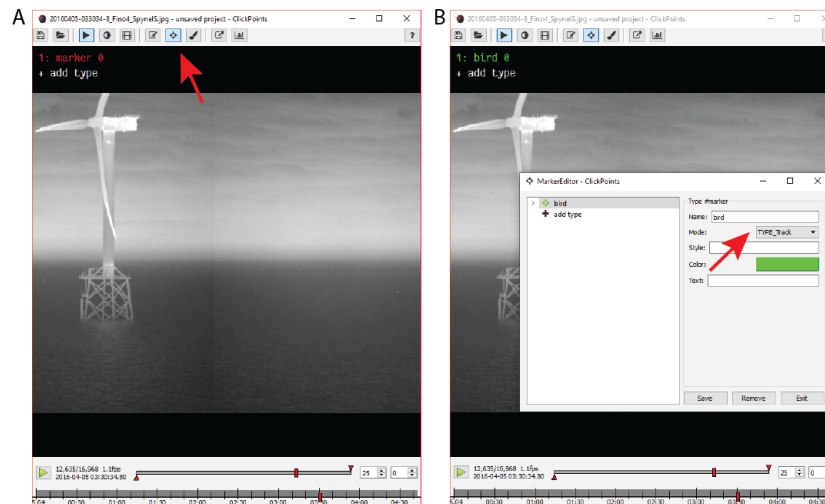



Fig. 3.2: Figure 1 | Defining a marker for tracking

4. Navigating the dataset

- Navigating the current frame:
 - right mouse button (hold) - to pan the image
 - mouse wheel - zoom the image
 - F - fit to view
 - W - full screen mode
 - H - hide time line
 - See General
- Navigating the dataset:
 - left & right cursor keys to go one frame forward and backward
 - Jump a specified set of frames with the numbad keys. See Jumping Frames
 - Use the frame and time navigation slider to by clicking or dragging the cursor to the desired position.
 - Jump to a specific frame by clicking on the frame counter and entering the desired frame number

- Press  to play the dataset with the specified frame rate or as fast as feasible.

Note: Due to the sequential compression of videos, traversing a video backwards is computationally expensive. ClickPoints provides a buffer so that the last N frames are stored and can be retrieved without any further computational cost. The default buffer size can be specified in the config.

Warning: Be careful not to reserve too much RAM for the frame buffer as it will drastically reduce performance!

5. Basic Tracking Procedure

The setup steps are completed, we can begin to mark some tracks.

1. Activate the type of marker you want to use by clicking on the label “bird” or press the associated number key.
2. Set the first marker by clicking on the image.
3. Switch to the next frame using the `right` cursor key.
4. The track now shows up with reduced opacity, indicating there is no marker for the current frame.
5. Upon dragging the marker (left click & hold) to the current position (release) a line indicates the connection to the last position. The track shows up with full opacity again.
6. If a frame is skipped, the marker can be dragged as usual but no connecting line will appear. Indicating a fragmentation of the track.
7. To create a second track, repeat step 1.
8. Markers are automatically save upon frame change or by pressing the `S` key.

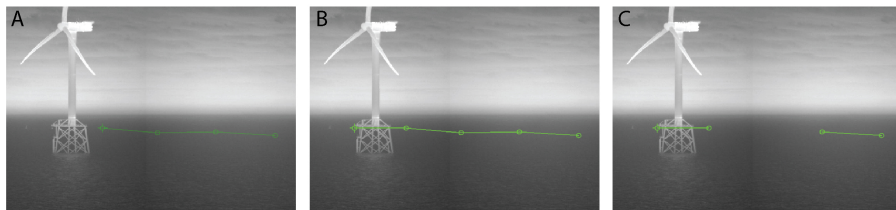


Fig. 3.3: Figure 2 | Track States

A - Track without update in current frame B - Track with update in current frame C - Track with missing marker

6. “Connect-nearest” Tracking Mode

For low density tracks ClickPoints provides the “connect nearest” mode. Clicking on the image will automatically connect the new marker to the closest Track in the last frame. Speeding up tracking for low track density scenes. The dragging of markers is still support and is usefull for intersecting tracks.

To activate “connect nearest” mode, set the config parameter `tracking_connect_nearest = True`.

See ConfigFiles for more details.

7. Important Controls

A list of useful controls for labeling tracks. Connect-nearest mode extends the list of default controls

- **default** left click - create new track (default mode) ctrl + left click - remove marker right click - open marker menu, see XXXXX
- **connect-nearest mode** left click - place marker, autoconnect to nearest track alt + left click - create new track shift + “left click” - place marker & load next frame

8. Advances Options

- Use SmartText to display additional information

See SmartText

Example: Display Track IDs

- open the marker menu
- navigate to “bird” marker type
- edit the text field by inserting

```
$track_id
```

All current markers of the type `bird` now display their internal track ID

- Use Styles to modify the display of markers and tracks

See Marker Styles

Example: Change track point display

- open the marker menu
- navigate to “bird” marker type
- edit the style field by inserting

```
{"track-line-style": "dash", "track-point-shape": "none"}
```

All tracks of the type `bird` now are displayed with dashed lines and without track points

Taking Measurements

How to perform quick measurements with ClickPoints and SmartText Markers



Fig. 3.4: Figure 3 | Tracks with SmartText ID



Fig. 3.5: Figure 3 | Tracks with modified style

CHAPTER 4

Modules

The modules are the different functions of the ClickPoints program. They can be accessed by the icons in the upper panel.




Fig. 4.1: The icon panel where all modules can be accessed.

Timeline

ClickPoints provides two timelines for navigation, a frame based and a timestamp based timeline. The frame based timeline is used by default, the timestamp timeline can be activated if time information of the displayed files is available. Time information extraction is implemented for the filename, the EXIF or TIFF headers.

Frame Timeline

The timeline is an interface at the bottom of the screen which displays range of currently loaded frames and allows for navigation through these frames. It can be displayed by clicking on .

To start/stop playback use the playback button at the left of the timeline or press `Space`. The label next to it displays which frame is currently displayed and how many frames the frame list has in total. The time bar has one slider to denote the currently selected frame and two triangular marker to select start and end frame of the playback. The keys `b` and `n` set the start/end marker to the current frame. The two tick boxes at the right contain the current frame rate and the number of frames to skip during playback between each frame. To go directly to a desired frame simply click on the frame display (left) and enter the frame number.

Each frame which has selected marker or masks is marked with a green tick mark (see Marker and Mask) and each frame marked with an annotation (see Annotations) is marked with a red tick. To jump to the next annotated frame press `Ctrl+Left` or `Ctrl+Right`.



Fig. 4.2: Frame Timeline example showing tick marks for marker and annotations.

Config Parameter

- **fps** = (int, value ≥ 0) if not 0 overwrite the frame rate of the video
- **play_start** = (float)
 - > 1 : at which frame to start playback at what
 - $0 > \text{value} < 1$: fraction of the video to start playback
- **play_end** =
 - > 1 : at which frame to start playback at what
 - $0 > \text{value} < 1$: fraction of the video to start playback
- **playing** = (bool) whether to start playback at the program start
- **timeline_hide** = (bool) whether to hide the timeline at the program start

Keys

- H - hide control elements
- Space - run/pause
- Ctrl + Left - previous image with marker or annotation
- Ctrl + Right - next image with marker or annotation

Date Timeline

The date timeline displays the timestamps of the loaded data set. To navigate to desired time point simply drag the current position marker or click on the point on the date timeline. The timeline can be panned and zoomed by holding

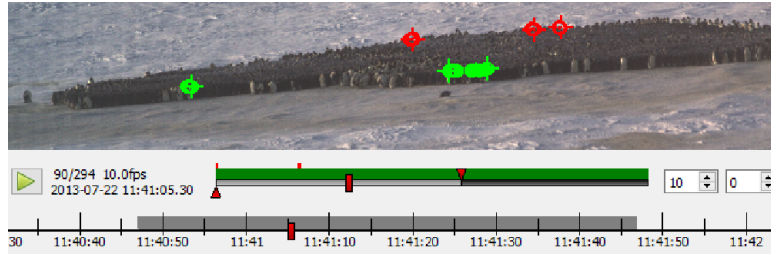


Fig. 4.3: Date Timeline example

the left mouse button (pan) and the mouse wheel (zoom). It aims to make it easier to get an idea of the time distribution of the data set, to find sections of missing data and facilitate navigation by a more meaningful metric than frames.

The extraction of timestamps by filename is fast than by EXIF. If you plan to repeatedly open files, without using a `.cdb` to store the time stamps, renaming them once might be beneficial. A list of timestamp search strings can be specified in the config file as shown in the code example below. As the search will be canceled after the first match it is necessary to order the the search strings by decreasing complexity.

Recommended naming template: `%Y%m%d-%H%M%S-%f_Location_Camera.type`


Config Parameter

- **datetimeline_show** = (bool) enable or disable the date timeline by setting this value to True or False
- **timestamp_formats** = (list of strings) list of match strings for images, with decreasing complexity
- **timestamp_formats2** = (list of strings)

list of match strings for videos with 2 timestamps, with decreasing complexity

```
# default values:
# for image formats with 1 timestamp
timestamp_formats = [r'%Y%m%d-%H%M%S-%f',      # e.g. 20160531-120011-2   with
↪fraction of second
                    r'%Y%m%d-%H%M%S']          # e.g. 20160531-120011
# for video formats with 2 timestamps (start & end)
timestamp_formats2 = [r'%Y%m%d-%H%M%S_%Y%m%d-%H%M%S']
```

GammaCorrection

The gamma correction is a slider box in the right bottom corner which allows to change the brightness and gamma of the currently displayed image. It can be opened by clicking on .

The box in the bottom right corner shows the current gamma and brightness adjustment. Moving a slider changes the display of the currently selected region in the images. The background of the box displays a histogram of brightness values of the current image region and a red line denoting the histogram transform given by the gamma and brightness adjustment. Pressing update the key G sets the currently visible region of the image as the active region for the adjustment. Especially for large images it increases performance significantly if only a portion of the image is adjusted. A click on reset resets gamma and brightness adjustments.



Fig. 4.4: An example gamma correction.

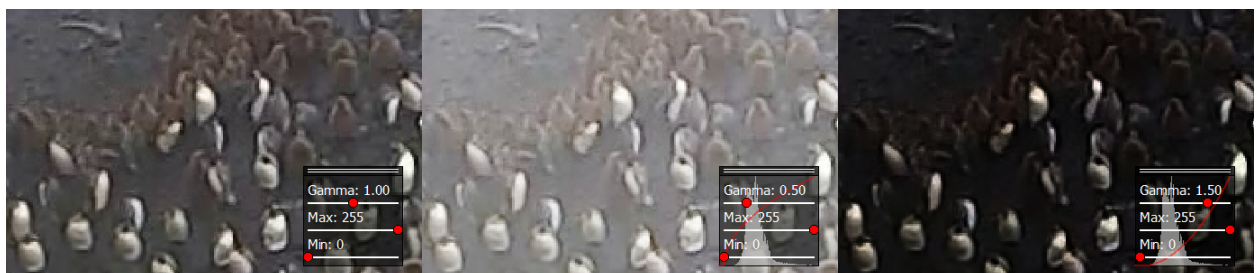


Fig. 4.5: The same image for different gamma values or 1, 0.5 and 1.5.

Gamma

The gamma value changes how bright and dark regions of the images are treated. A low gamma value (<1) brightens the dark regions up while leaving the bright regions as they are. A high gamma value (>1) darkens the dark regions of the image while leaving the bright regions as they are.

Brightness



Fig. 4.6: The same image for different brightness values, where once the lower and once the upper range was adjusted.


The brightness can be adjusted by selecting the Max and Min values. Increasing the Min value darkens the image by setting the Min value (and everything below) to zero intensity. Decreasing the Max value brightens the image by setting the Max value (and everything above) to maximum intensity.

Keys

- G: update rect

VideoExporter

The video exporter allows for the export of parts of the currently loaded images as a video, image sequence or gif file.


It can be opened using the  or by pressing z. A dialog will open, which allows to select an output filename for a video, an image sequence (which has to contain a %d number placeholder) or a gif file. Frames are exported starting from the start marker in the timeline to the end marker in the timeline. The framerate is also taken from the timeline. Images are cropped according to the current visible image part in the main window.

Keys

- Z: Export Video

Annotations

Annotations are text comments which can include a rating and tags, which is attached to a frame. To annotate a frame

or edit the annotation of a frame press A or  and fill in the information in the dialog. The frame will be marked with a red tick in the timeline. To get a list of all annotated frames press Y. In this list clicking an annotation results in a jump to the frame of the annotation.

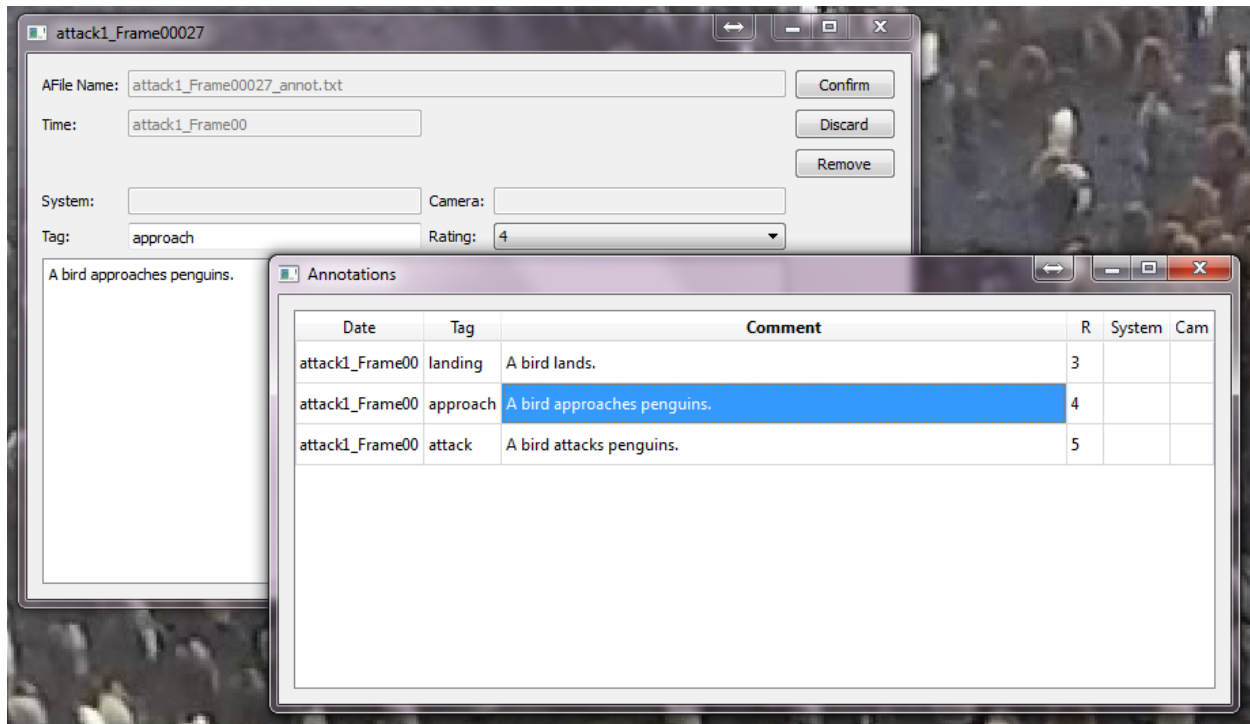


Fig. 4.7: An example of both the annotation editor and the annitation overview window.

Keys

- A: add/edit annotation
- Y: show annotation overview

Marker

Marker are added to a frame to refer to pixel positions. Marker can have different types to mark different objects. They can also be used in tracking mode to recognize an object over different frames.



The marker editor can be opened by clicking on .

The list of available markers is displayed at the top left corner. A marker type can be selected either by clicking on its name or by pressing the corresponding number key. A left click in the image places a new marker of the currently selected type. Existing markers can be dragged with the left mouse button and deleted by clicking on them while holding the control key.

To save the markers press S or change to the next image, which automatically saves the current markers.

Marker types

A right click on any marker or type opens the Marker Editor window. There types can be created, modified or deleted. Marker types have a name, which is displayed in the HUD, a color and a mode.

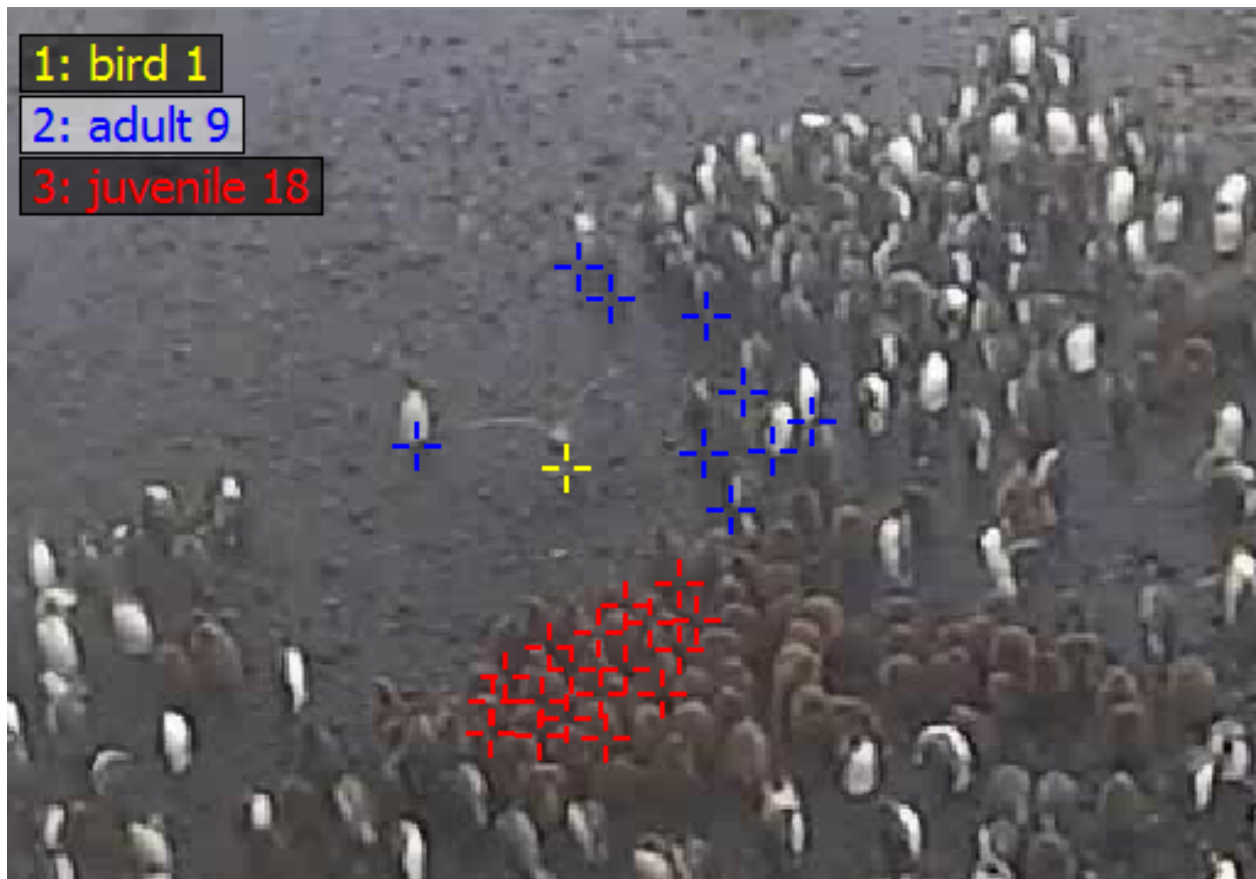


Fig. 4.8: An example image showing three different marker types and some markers placed on the image.



Fig. 4.9: Different marker type modes.

TYPE_Normal results in single markers. TYPE_Rect joins every two consecutive markers as a rectangle. TYPE_Line joins every two consecutive markers as a line. TYPE_Track specifies that this markers should use tracking mode (see section Tracking Mode).

Marker display

Pressing T toggles between three different marker displays. If the smallest size is selected, the markers can't be moved. This makes it easier to work with a lot of markers on a small area.



Fig. 4.10: The same marker in different size configurations.

Tracking mode

Often objects which occur in one image also occur in another image (e.g. the images are part of a video). Then it is necessary to make a connection between the object in the first image and the object in the second image. Therefore ClickPoints features a tracking mode, where markers can be associated between images. It can be enabled using the TYPE_Track for a marker type. The following images displays the difference between normal mode (left) and tracking mode (right):



Fig. 4.11: The same marker in normal mode (left) and in tracking mode (right). The track always displays all previous positions connected with a line, when they are from two consecutive images.

To start a track, mark the object in the first image. Then switch to the next image and the marker from the first image will still be displayed but only half transparent. To add a second point to the track grab the marker and move it to the new position of the object. Continue this process through the images where you want to track the object. If the object didn't move from the last frame or isn't visible, an image can be left out, which results in a gap in the track. To remove a point from the track, click it while holding control.

Marker Editor

The Marker Editor is used to manage marker types. New marker types can be created, existing ones can be modified or deleted.

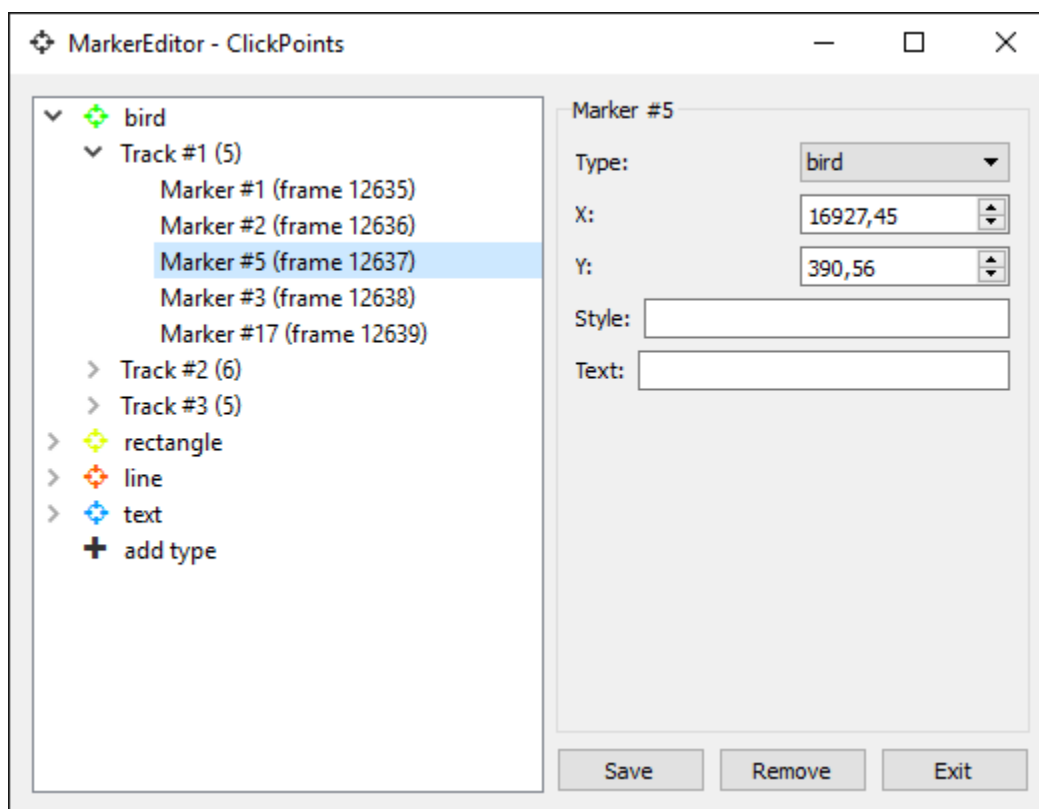



Fig. 4.12: The Marker Editor used to create and change marker types, navigate to tracks and marks and delete marker, tracks and types



Creating Marker Types To create a new marker type open the marker editor via  or right click on the marker display or a marker. Select the `+add type` field, enter a name, set the marker mode to marker, line, rectangle or track and choose a color. Further modifications can be achieved via the text and style field, for more details see the following sections.

Editing Marker Types To edit a marker type, simply select the type from the menu, changes the desired values and save the changes by pressing `Save`

Note: It is NOT possible to change marker types as long as marker objects of this type exist. E.g. you can't make lines out of regular markers as they don't have a second point.

Navigation The editor can also be used to navigate. Selecting a marker will bring you to the frame the marker is placed in. By clicking on the arrow in front of the type name the marker or track overview unfolds. Selecting a marker of a track will bring you to the frame it is placed in.

Deleting Types, Tracks and Markers Types, tracks and markers can be removed by selecting the object in the tree and pressing the `Remove` button. By removing a marker type all markers and tracks of this type are removed, removing a track will remove all markers of this track.

Warning: There is no undo button!

Marker Style Definitions

Style definitions can provide additional features to change the appearance of marker. They are inherited from the marker type to the track and from the track to the marker itself. If no track is present the marker inherits its style directly from the type. This allows to define type, track and marker specific styles.

Styles can be set using the Marker Editor (right click on any marker or type).

The styles use the JSON format for data storage. The following fields can be used:

- **Marker Color - "`color`":** `"#FF0000"` Defines the color of the marker in hex format. Color can also be a `matplotlib` colormap followed optionally by a number (e.g. `jet(30)`), then that many colors (default 100) are extracted from the color map and used for the marker/tracks to color every marker/track differently.
- **Marker Shape - "`shape`":** `"cross"` Defines the shape of the marker.
values: `cross` (default), `circle`, `rect`
- **Marker Line Width - "`line-width`":** `1` Defines the line width of the markers symbol (e.g. width of the circle). Ignored if a filled symbol (e.g. the cross) is used.
- **Marker Scale - "`scale`":** `1` Scaling of the marker.
- **Marker Transform - "`transform`":** `"screen"` If the marker should have a fixed size with respect to the screen or the image.
values: `screen` (default), `image`
- **Track Line Style - "`track-line-style`":** `"solid"` The style of the line used to display the track history.
values: `solid` (default), `dash`, `dot`, `dashdot`, `dashdotdot`
- **Track Line Width - "`track-line-width`":** `2` The line width of the line used to display the track history.

- **Track Gap Line Style – "track-gap-line-style": dash** The style of the line used to display gaps in the track history.
values: solid, dash (default), dot, dashdot, dashdotdot
- **Track Gap Line Width – "track-gap-line-width": 2** The line width of the line used to display gaps in the track history.
- **Track Marker Shape - "track-point-shape": "circle"** The marker shape used to display the track history.
values: circle (default), rect, cross, none
- **Track Marker Scale - "track-point-scale": 1** The scaling of markers used to display the track history.

Style Examples:

```
{ "color": "jet(30)" } # style for providing a marker type with 30 different colors
{ "track-line-style": "dash", "track-point-shape": "none" } # change the track style
```

Marker Text & SmartText

The text field allows to attach text to marker, line, rectangle and track objects. Text properties are inherited from the marker type to the track and from the track to the marker itself. If no track is present the marker inherits its text directly from the type. This allows to define type, track and marker specific texts.

Text can be set using the Marker Editor (right click on any marker or type).

ClickPoints provides a SmartText feature, enabling the display of self updating text in to display pre defined values. SmartText keyword always start with a \$ character. The keywords are depending on the type for marker, as explained in the following overview:

General

/n insert a new line

\$marker_id inserts the id of the marker, line or rectangle object

\$x_pos inserts the x position of the marker, first marker of a line or top left marker of a rectangle

\$y_pos inserts the y position of the marker, first marker of a line or top left marker of a rectangle

Line

\$length inserts the length of the line in pixel with 2 decimals.

Rectangle

\$area inserts the area of the rectangle in pixel with 2 decimals.

Track

\$track_id inserts the track id of the track.

Text Examples:

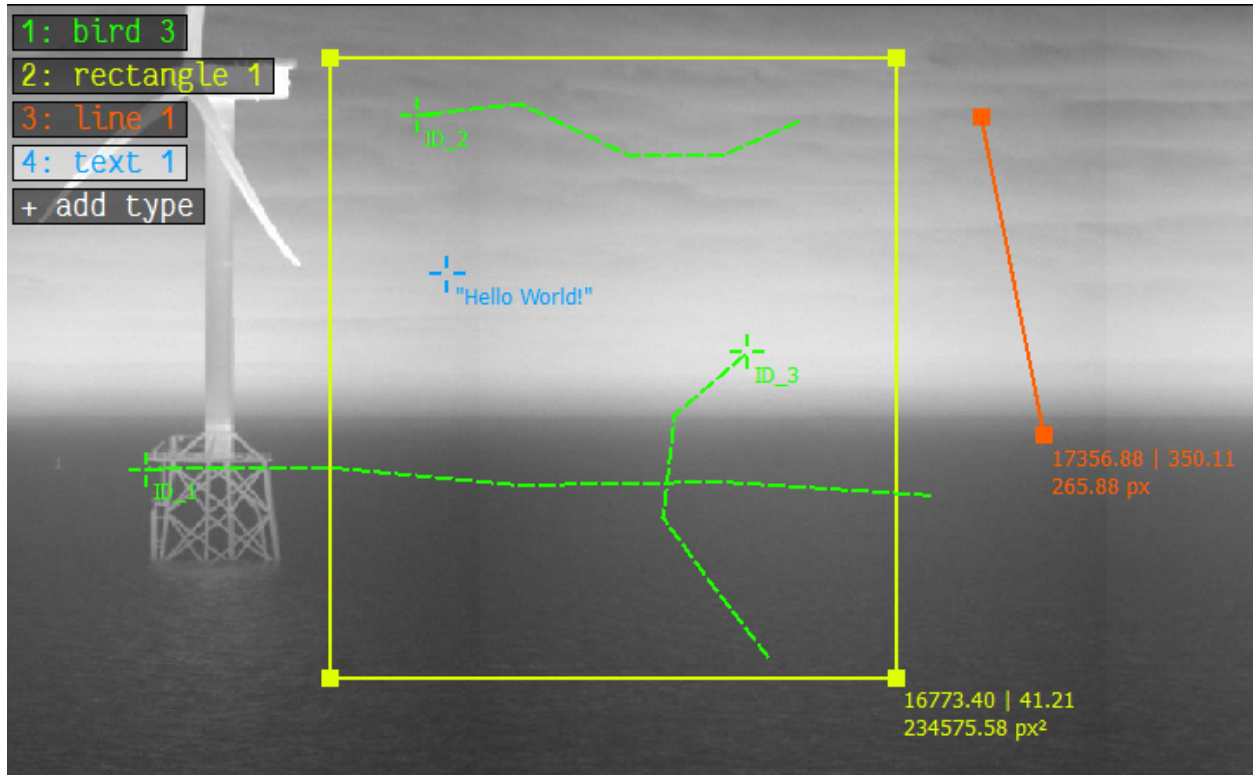
```
# regular Text
Marker: "Hello World!" # shows the text Hello World!

# SmartText
Track: "ID_$track_id" # shows the track ID
```

```

Line: "$x_pos | $y_pos \n$length px"           # shows the x & y coordinate and
↪length
Rect: "ID_$marker_id\n$x_pos | $y_pos \n$area px" # shows the object_id, its x & y
↪coordinate and area

```



Mask

A mask can be painted to mark regions in the image with a paint brush. The mask editor can be opened by clicking on



A list of available mask colors is displayed in the top right corner. Switching to paint mode can be done using the key P, pressing it again switches back to marker mode. Colors can be selected by clicking on its name or pressing the corresponding number key. Holding the left mouse button down draws a line in the mask using the selected color. To save the mask press S or change to the next image, which automatically saves the current mask. The mask type delete acts as an eraser and allows to remove labeled regions.

Define colors

A right click on a color name opens the mask editor menu, which allows the creation, modification and deletion of mask types. Every mask type consists of a name and a color.

Brush size

The brush radius can be de- and increased using the keys – and +.

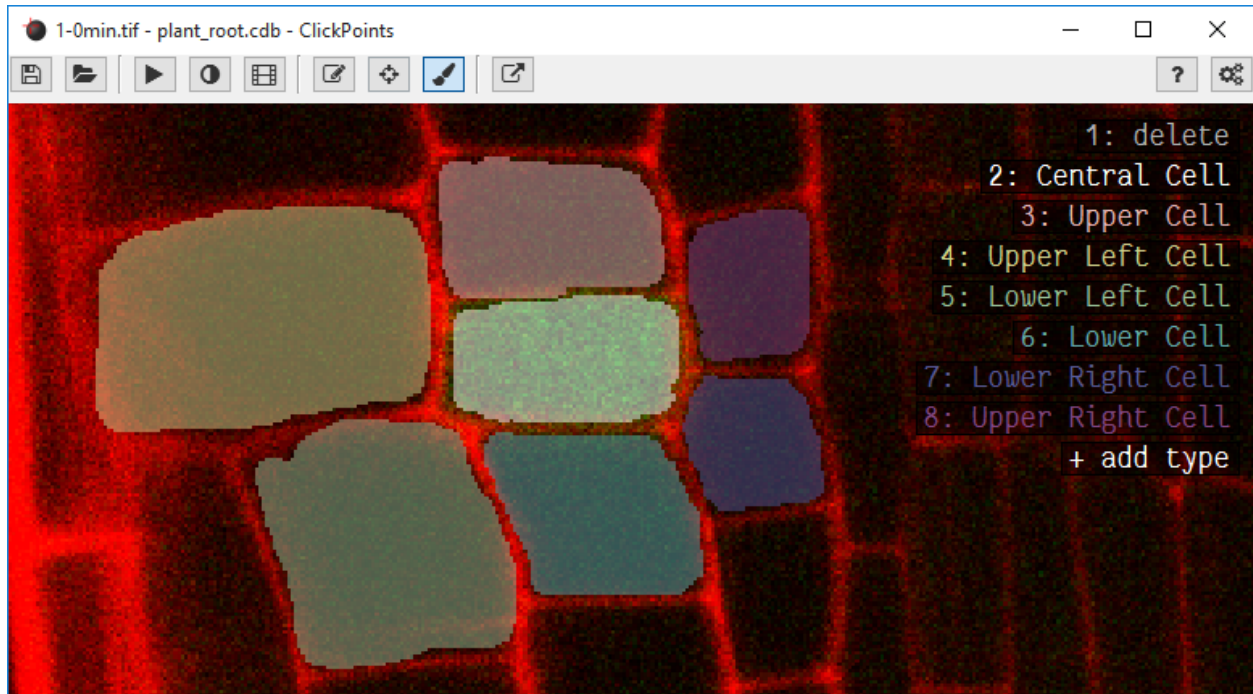


Fig. 4.13: An image where 7 regions have been marked with different masks.

Color picker

The color can alternatively to selection via number buttons or a click on the names be selected by using `K` to select the color which is currently below the cursor.

Mask transparency

The transparency of the mask can be adjusted with the keys `I` and `O`.

Mask update

Updating masks can be slow if the images are very large. To enable fast painting of large masks, ClickPoints can disable the automatic updates of the mask by disabling the option `Auto Mask Update`. If automatic updates are disabled the key `M` redraws the currently displayed mask.

Config Parameter

- `auto_mask_update` = whether to update the mask display after each stroke or manually by key press
- `draw_types` = `[[0, [255, 0, 0]]` specifies what categories to use for mask drawing. Every category is an array with two entries: index and color.

Keys

- 0-9: change brush type

- K: pick color of brush
- -: decrease brush radius
- +: increase brush radius
- O: increase mask transparency
- I: decrease mask transparency
- M: redraw the mask

Info Hud

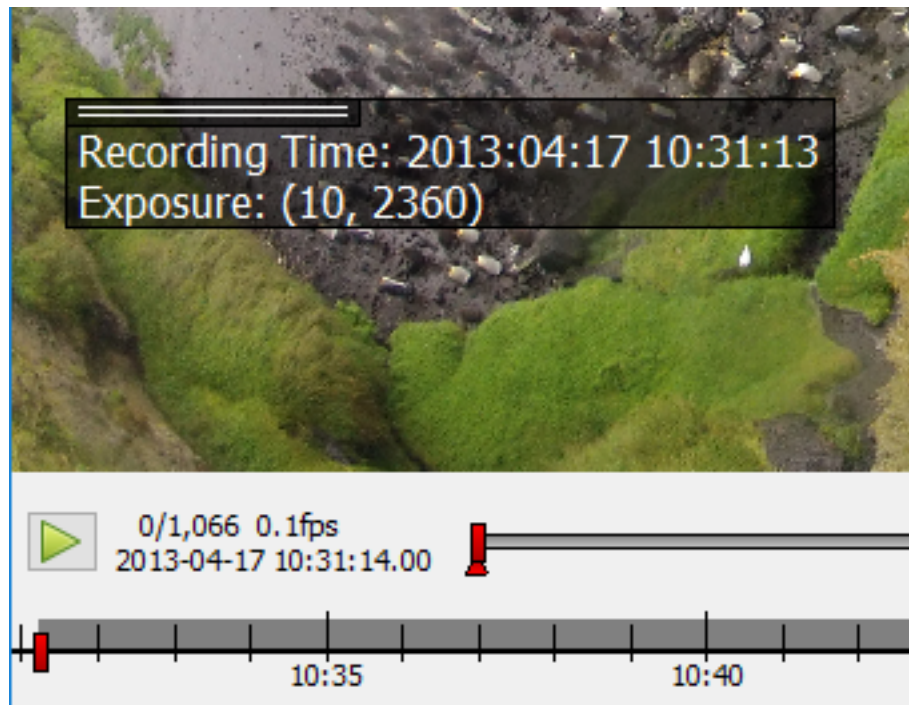


Fig. 4.14: Example of the info hud displaying time and exposure exif data from a jpg file.

This info hud can display additional information for each image. Information can be obtained from the filename, jpeg exif information or tiff metadata or be provided by an external script.

The text can be set using the options dialog. Placeholders for additional information are written with curly brackets {}. The keyword from the source (regex, exif or meta) is followed by the name of the information in brackets [], e.g. {exif[rating]}. If the text is set to @script the info hud can be filled using an external script. Use \n to start a new line.

To extract data from the filename a regular expression with named fields has to be provided.

Examples

Data from filename

```
file: "penguins_5min.jpg"

Info Text: "Animal: {regex[animal]} Time: {regex[time]}"
Filename Regex: '(?P<animal>.+?[^_])_(?P<time>.+?)min'

Output: "Animal: penguin Time: 5"
```

Data from exif

```
file: "P1000236.jpg"

Info Text: "Recording Time: {exif[DateTime]} Exposure: {exif[ExposureTime]}"

Output: "Recording Time: 2016:09:13 10:31:13 Exposure: (10, 2360)"
```

The keys can be any field of the jpeg exif header as e.g. shown at <http://www.exiv2.org/tags.html>

Data from meta

```
file: "20160913_134103.tif"

Info Text: "Magnification: {meta[magnification]} PixelSize: {meta[pixelsize]}"

Output: "Magnification: 10 PixelSize: 6.45"
```

The values presented in the meta field of tiff files varies by the tiff writer. ClickPoints can only access tiff meta data written in the json format in the tiff meta header field, as done by the `tifffile` python package.

Data from script

```
Info Text: "@script"
```


and a script file listening to the `PreLoadImageEvent` should set the text with `com.updateHUD`. This script should be started via the script launcher and could look like this:

```
1 from __future__ import print_function, division
2 import os
3 import numpy as np
4 import socket
5 import select
6
7 import clickpoints
8
9 start_frame, database, port = clickpoints.GetCommandLineArgs()
10 com = clickpoints.Commands(port, catch_terminate_signal=True)
11
12 def displayMetaInfo(ans):
13     # print('in function:', ans)
14     command, fullname, framenr = ans[0].split(' ', 2)
15     fpath, fname = os.path.split(fullname)
16     com.updateHUD(framenr+" "+fullname)
```

```
17
18 # input
19 HOST = "localhost"
20 PORT = port
21 BROADCAST_PORT = PORT + 1
22
23 # broadcast socket to listen to
24 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
25 sock.setblocking(0)
26 sock.bind(('127.0.0.1', BROADCAST_PORT))
27
28 last_img_nr = -1
29 # main loop
30 while True:
31     ready_to_read, ready_to_write, in_error = select.select([sock], [], [], 0)
32
33     # wait for incoming signal
34     if ready_to_read:
35         ans = sock.recvfrom(1024)
36
37         # split information
38         img_nr = np.int(ans[0].split()[2])
39
40         if ans[0].startswith('PreLoadImageEvent') and img_nr != last_img_nr:
41             # print("nr is:", img_nr)
42             displayMetaInfo(ans)
43             last_img_nr = img_nr
44
45         # annoying buffer part
46         # read out and thereby delete all remaining entries
47         last_message = ""
48         messages_pending = False
49         ready_to_read, ready_to_write, in_error = select.select([sock], [], [], 0)
50         if ready_to_read:
51             messages_pending = True
52             while messages_pending:
53                 ready_to_read, ready_to_write, in_error = select.select([sock],
54 ↪ [], [], 0)
55                 # clear incoming buffer
56                 if ready_to_read:
57                     tmp = sock.recvfrom(1024)
58                     # print('message pending', tmp)
59                     if tmp[0].startswith('PreLoadImageEvent'):
60                         last_message = tmp
61                         # print('lastmsg:', last_message)
62                 else:
63                     messages_pending = False
64                     # make sure last message is displayed
65                     if not last_message == ans and not last_message == '' and img_
66 ↪ nr != last_img_nr:
67                         print("reached this")
68                         displayMetaInfo(last_message)
69                         last_message = ''
70                         last_img_nr = img_nr
```

Add-ons

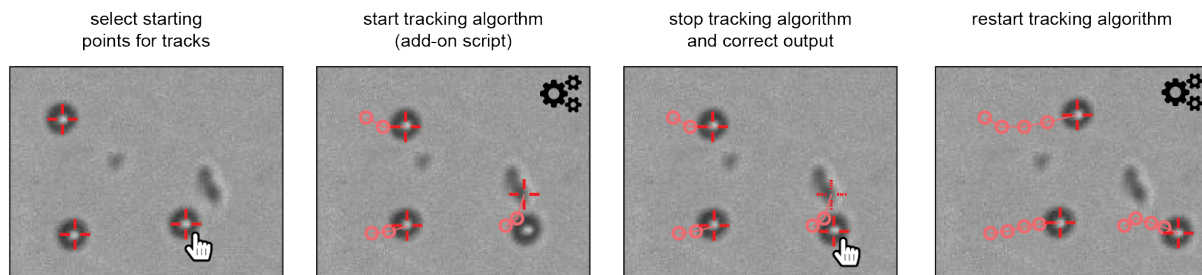
Add-ons are helpful scripts which are not part of the main ClickPoints program, but can be loaded on demand to do some evaluation task.

They can be loaded by clicking on  and loading a the .py of the add-on. ClickPoints already comes with a couple of add-ons, but it is easy to add your own or extend existing ones.


Each addon will be assigned to a key from F12 downwards (F12, F11, F10 and so on). Hitting this key will start the addon with access to the current project database and the current ClickPoints instances. Hitting this key again will stop the addon again.

To configure ClickPoints to already have scripts loaded on startup, you can define them in the `ConfigClickPoints.txt` file as `launch_scripts =`.

Tracking



This add-on takes markers in one image and tries to find the corresponding image parts in the subsequent images.

To use it, open a ClickPoints session and add the add-on `Track.py` by clicking on .

Create a marker type with mode `TYPE_Track`. Mark every object which should be tracked with a marker of this type. Then hit `F12` (or the button you assigned the `Track.py` to) and watch the objects to be tracked. You can at any point hit the key again to stop the tracking. If the tracker has made errors, you can move the marker by hand and restart the tracking from the new position.


The algorithm uses the position using a sparse iterative Lucas-Kanade optical flow algorithm [1].

Attention: If the markers are not in a `TYPE_Tracking` type, they are not tracked by `Track.py`. Also marker which already have been tracked are only tracked again, if they were moved in ClickPoints.

References

Drift Correction


This add-on takes a region in the image and tries to find it in every image. The offset saved for every image to correct for drift in the video.

To use it, open a ClickPoints session and add the add-on `DriftCorrection.py` by clicking on .

When you first start the script a marker type named `drift_rect` is created. Use this type to select a region in the image which remains stable over the course of the video. Start the drift correction script by using `F12` (or the key the script is connected to). The drift correction can be stopped and restarted at any time using the key again.

Cell Detector

This add-on is designed to take a microscope image of fluorescently labeled cell nuclei and find the coordinates of every cell.


To use it, open a ClickPoints session and add the add-on `CellDetector.py` by clicking on .

Start the cell detector script by using `F12` (or the key the script is connected to). All found cell nuclei will be labeled with a marker.

Attention: The Cell Detector won't work for cells which are too densely clustered. But ClickPoints allows you to review and adjust the results if some cells were not detected.

Grab Plot Data

This add-on helps to retrieve data from plots.

To use it, open a ClickPoints session and add the add-on `GrabPlotData.py` by clicking on .

Sometimes it is useful to extract data from plotted results found in publications to compare them with own results or simulations. ClickPoints therefore provides the add-on “GrabPlotData”. It uses three marker types. The types “`x_axis`” and “`y_axis`” should be used to mark the beginning and end of the x and y axis of the plot. Markers should be assigned a text containing the value which is associated with this point on the axis. These axis markers are used

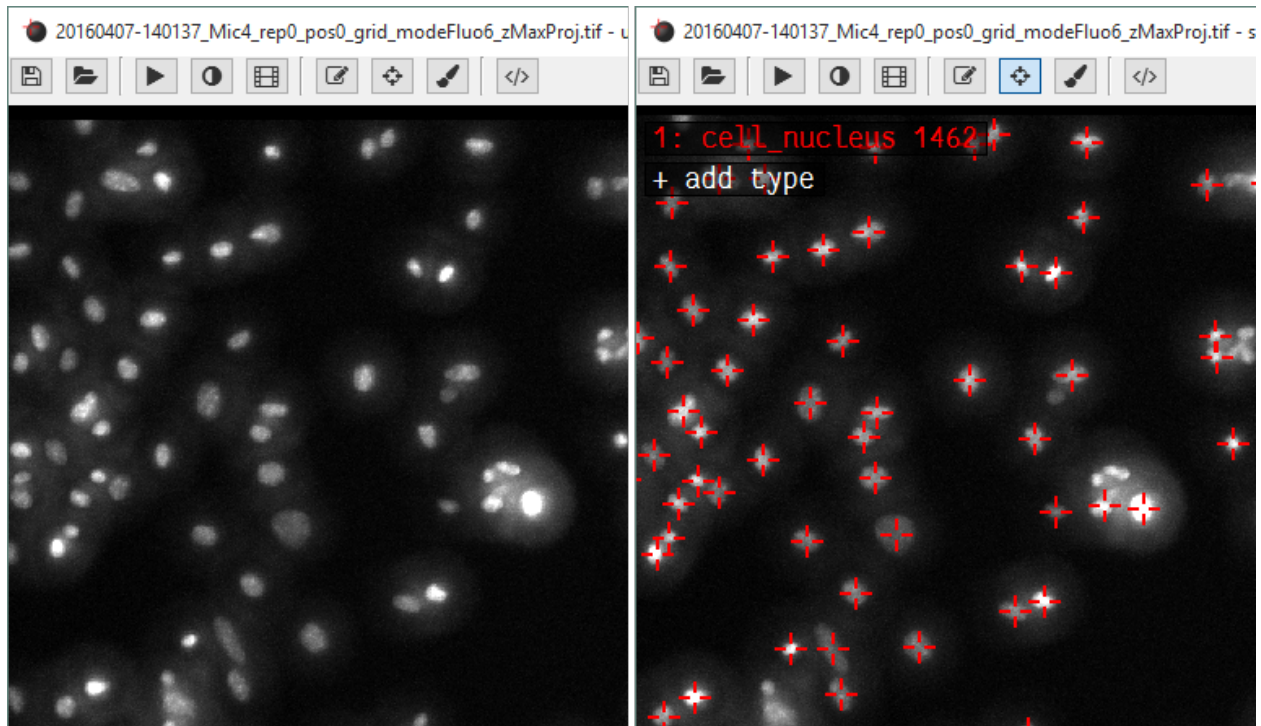


Fig. 5.1: An image of cell nuclei before and after executing the Cell Detector addon.

to remap the pixel coordinates of the “data” markers to the values provided by the axis. These remapped values are stored in a “.txt” file that has the same name as the image.

Attention: This can only be used if the axe is not scaled logarithmically. Only linear axes are supported.

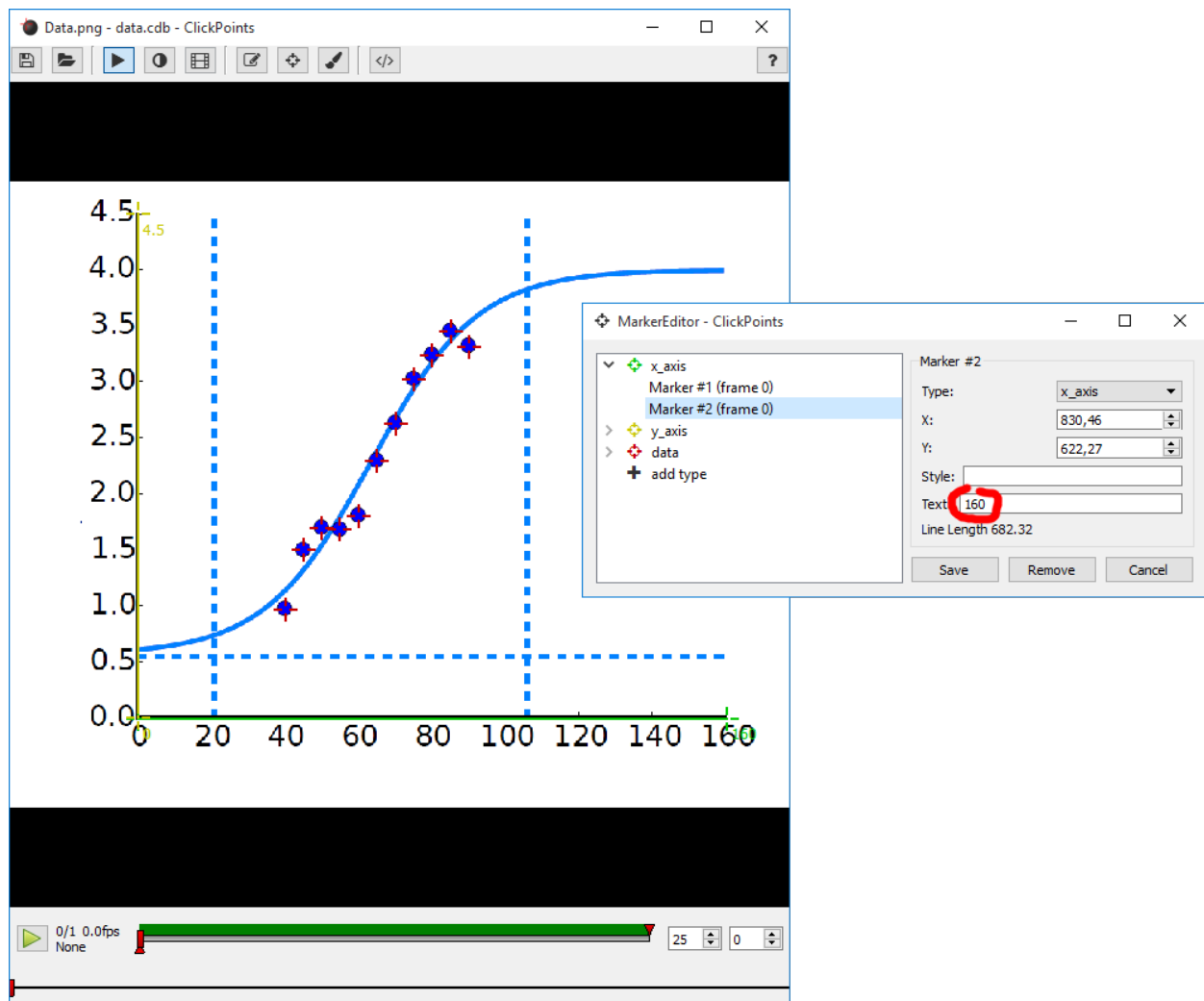


Fig. 5.2: The two axis are marked with the corresponding markers and the data points with the data markers. The start and end points of the axis are assigned a text containing the corresponding axis value.

Examples

The examples provide some usage examples of ClickPoints to demonstrate its various functionalities and how data can be processed with ClickPoints and later easily evaluated with ClickPoints.

To keep the download size of ClickPoints down, the examples are kept in a separate repository. They can be downloaded [here](#).

Count Penguins

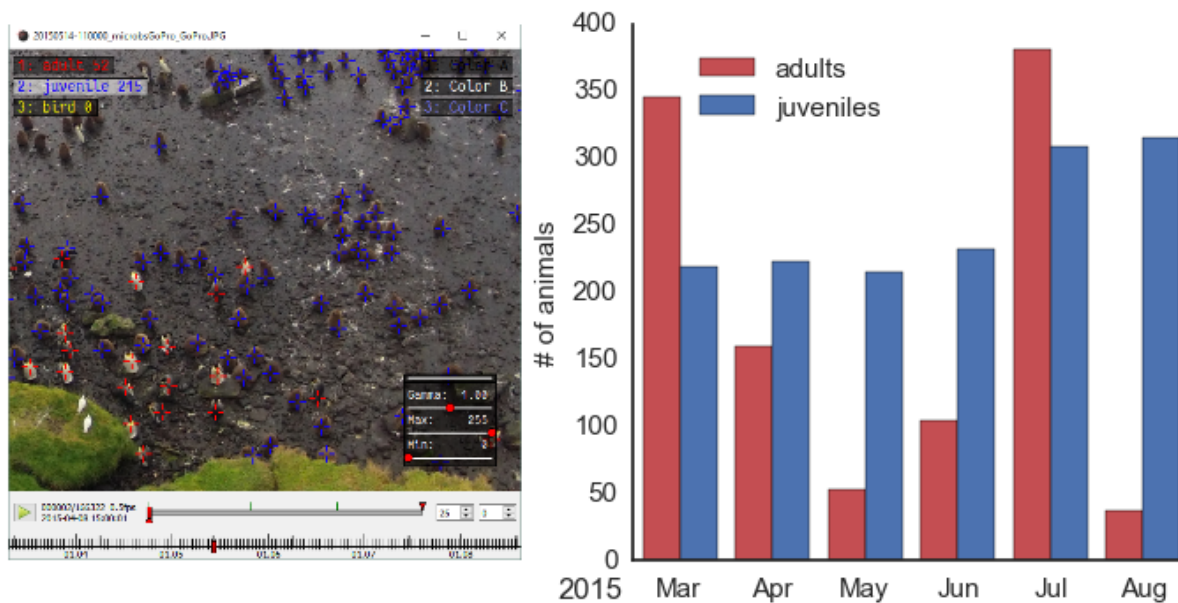


Fig. 6.1: Left: image of clickpoints to count penguins. Right: number of penguins counted.

In the example, we show how the ClickPoints can be used to count penguins animals in an image.

The example contains some images recorded with a GoPro Hero 2 camera, located at the Baie du Marin King penguin colony on Possession Island of the Crozet Archipelago [1]. Two marker types were added in ClickPoints to count the adult and juvenile animals.

The the counts can be evaluated using a small script:

```

1 import matplotlib.pyplot as plt
2 import clickpoints
3
4 # open database
5 db = clickpoints.DataFile("count.cdb")
6
7 # iterate over images
8 for index, image in enumerate(db.getImages()):
9     # get count of adults in current image
10    marker = db.getMarkers(image=image, type="adult")
11    plt.bar(index, marker.count(), color='b', width=0.3)
12
13    # get count of juveniles in current image
14    marker = db.getMarkers(image=image, type="juvenile")
15    plt.bar(index+0.3, marker.count(), color='r', width=0.3)
16
17 # display the plot
18 plt.show()

```

References

Flourescence intensities in plant roots

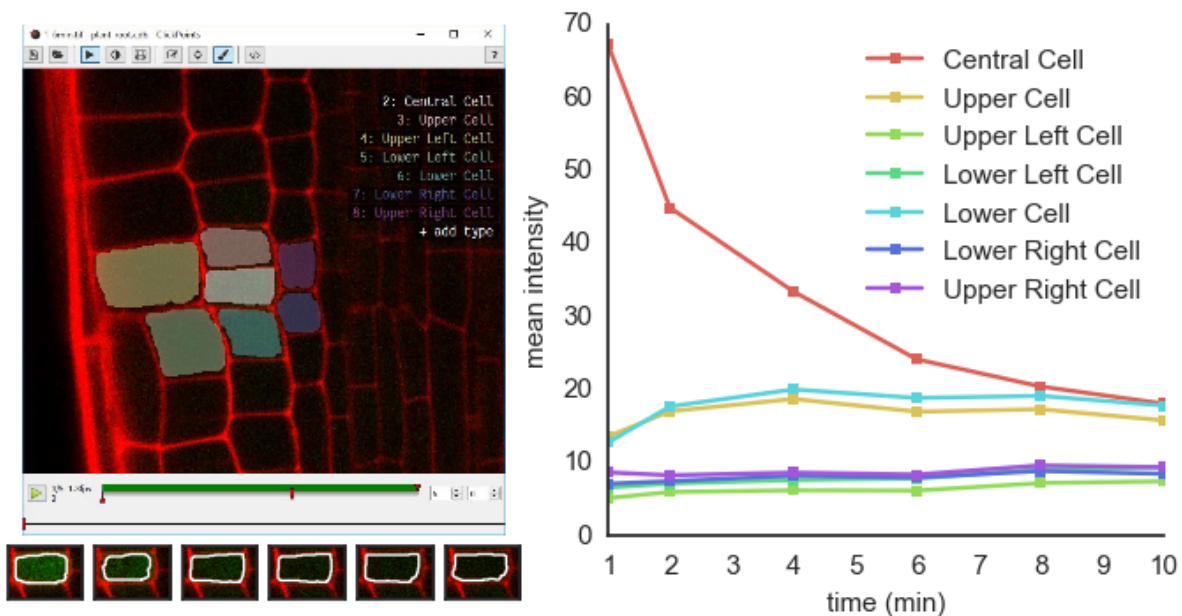


Fig. 6.2: Left: image of a plant root in ClickPoints. Right: fluorescence intensities of the cells over time.

In the example, we show how the mask painting feature of ClickPoints can be used to evaluate fluorescence intensities in microscope recordings.

Images of an *Arabidopsis thaliana* root tip, obtained using a two-photon confocal microscope [1], recorded at 1 min time intervals are used. The plant roots expressed a photoactivatable green fluorescent protein, which after activation with a UV pulse diffuses from the activated cells to the neighbouring cells.

For each time step a mask is painted to cover each cell in each time step.

The fluorescence intensities be evaluated using a small script:

```

1  from __future__ import division, print_function
2  import re
3  import numpy as np
4  from matplotlib import pyplot as plt
5
6  # connect to ClickPoints database
7  # database filename is supplied as command line argument when started from ClickPoints
8  import clickpoints
9  start_frame, database, port = clickpoints.GetCommandLineArgs()
10 db = clickpoints.DataFile(database)
11 com = clickpoints.Commands(port, catch_terminate_signal=True)
12
13 # get images and mask_types
14 images = db.getImages()
15 mask_types = db.getMaskTypes()
16
17 # regular expression to get time from filename
18 regex = re.compile(r".*(?P<experiment>\d*)-(?P<time>\d*)min")
19
20 # initialize arrays for times and intensities
21 times = []
22 intensities = []
23
24 # iterate over all images
25 for image in images:
26     print("Image", image.filename)
27     # get time from filename
28     time = float(regex.match(image.filename).groupdict()["time"])
29     times.append(time)
30
31     # get mask and green channel of image
32     mask = image.mask.data
33     green_channel = image.data[:, :, 1]
34
35     # sum the pixel intensities for every channel
36     intensities.append([np.mean(green_channel[mask == mask_type.index]) for mask_type in mask_types])
37
38 # convert lists to numpy arrays
39 intensities = np.array(intensities).T
40 times = np.array(times)
41
42 # iterate over cells
43 for mask_type, cell_int in zip(mask_types, intensities):
44     plt.plot(times, cell_int, "-s", label=mask_type.name)
45
46 # add legend and labels
47 plt.legend()

```

```

48 plt.xlabel("time (min)")
49 plt.ylabel("mean intensity")
50 # display the plot
51 plt.show()

```

References

Supervised Tracking of Fiducial Markers in Magnetic Tweezer Measurements

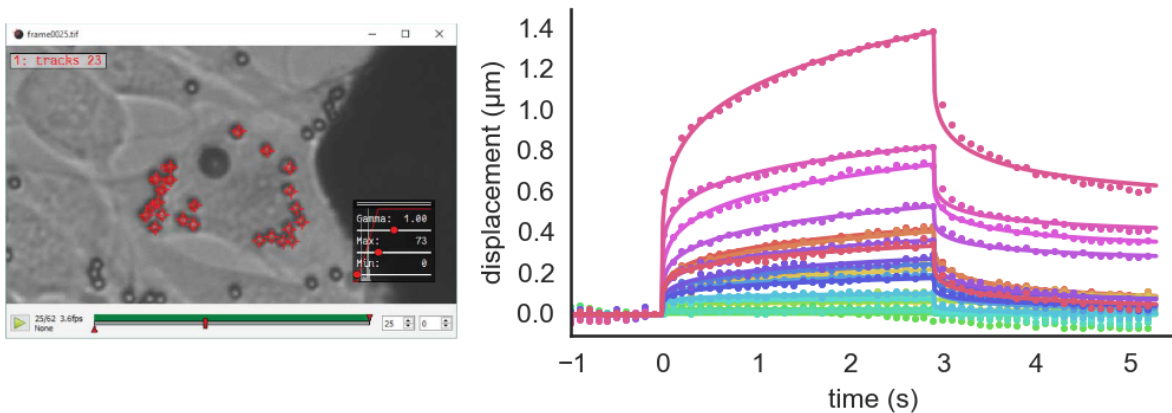


Fig. 6.3: Left: the image of beads on cells loaded in ClickPoints. Right: displacement of beads.

In the example, we show how the ClickPoints addon `Track.py` can be used to track objects in an image and how the resulting tracks can later on be used to calculate displacements. [\[1\]](#)

The data we show in this example are measurements of a magnetic tweezer, which uses a magnetic field to apply forces on cells. The cell is additionally tagged with non magnetic beads, which are used as fiducial markers.

The images can be opened with ClickPoints and every small bead (the fiducial markers) is marked with a marker of type `tracks`. Then the `Track.py` addon is started to find the position of these beads in the subsequent images.

The tracks can then be evaluated using a small script:

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # connect to ClickPoints database
5  # database filename is supplied as command line argument when started from ClickPoints
6  import clickpoints
7  start_frame, database, port = clickpoints.GetCommandLineArgs()
8  db = clickpoints.DataFile(database)
9
10 # get all tracks
11 tracks = db.getTracks()
12
13 # iterate over all tracks
14 for track in tracks:
15     # get the points

```

```

16 points = track.points_corrected
17 # calculate the distance to the first point
18 distance = np.linalg.norm(points[:, :] - points[0, :], axis=1)
19 # plot the displacement
20 plt.plot(track.frames, distance, "-o")
21
22 # show the plot
23 plt.xlabel("# frame")
24 plt.ylabel("displacement (pixel)")
25 plt.show()

```

References

Using ClickPoints for Visualizing Simulation Results

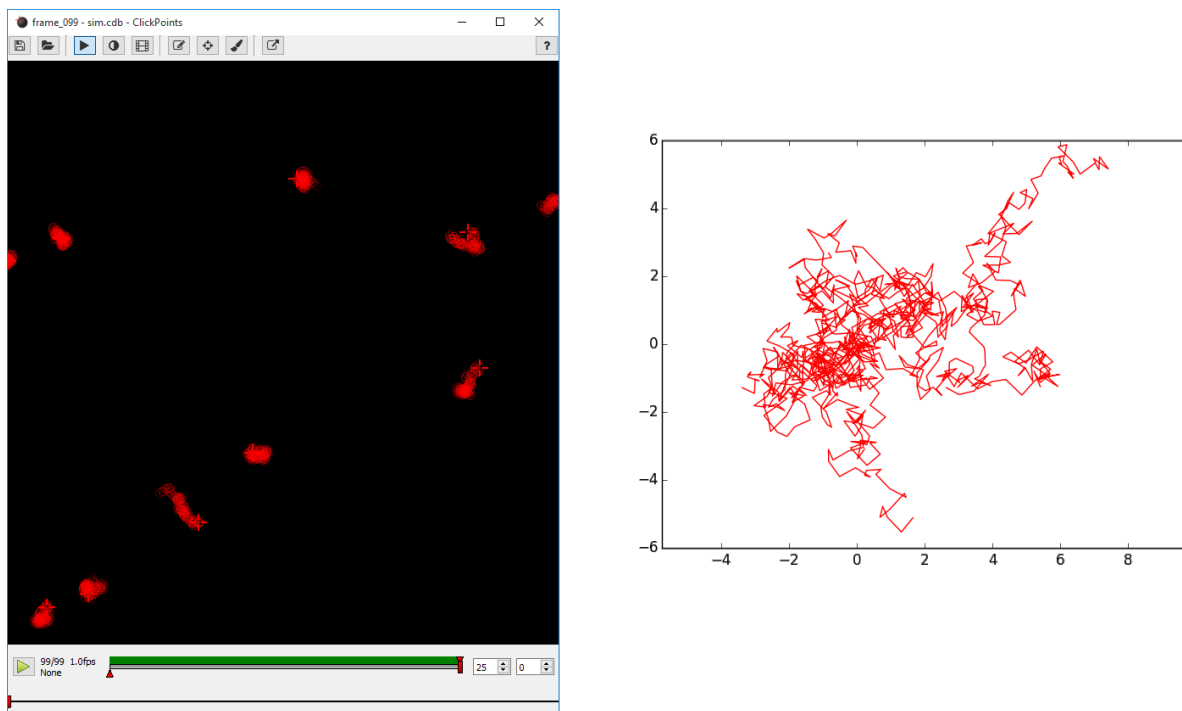


Fig. 6.4: Left: Tracks of the random walk simulation in ClickPoints. Right: Tracks plotted all starting from (0, 0).

Here we show how ClickPoints can be apart from viewing and analyzing images also be used to store simulation results in a ClickPoints Project file. This has the advantages that the simulation can later be viewed in ClickPoints, with all the features of playback, zooming and panning. Also the coordinates of the objects used in the simulation can later be accessed through the ClickPoints Project file.

This simple example simulates the movement of 10 object which follow a random walk.

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import clickpoints
4 import io
5

```

```
6  # Simulation parameters
7  N = 10
8  size = 100
9  size = 100
10 frame_count = 100
11
12 # create new database
13 db = clickpoints.DataFile("sim.cdb", "w")
14
15 # Create a new marker type
16 type_point = db.setMarkerType("point", "#FF0000", mode=db.TYPE_Track)
17
18 # Create track instances
19 tracks = [db.setTrack(type_point) for i in range(N)]
20
21 # Create initial positions
22 points = np.random.rand(N, 2)*size
23
24 # iterate
25 for i in range(frame_count):
26     print(i)
27     # Create a new frame
28     image = db.setImage("frame_%03d" % i, width=size, height=size)
29
30     # Move the positions
31     points += np.random.rand(N, 2)-0.5
32
33     # Save the new positions
34     db.setMarkers(image=image, x=points[:, 0], y=points[:, 1], track=tracks)
35
36 # plot the results
37 for track in tracks:
38     plt.plot(track.points[:, 0], track.points[:, 1], '-')
39 plt.xlim(0, size)
40 plt.ylim(size, 0)
41 plt.show()
```

CHAPTER 7

Database API

ClickPoints comes with a powerful API which enables access from within python to ClickPoints Projects which are stored in a `.cdb` ClickPoints SQLite database.

To get started reading and writing to a database use:

```
1 import clickpoints
2 db = clickpoints.DataFile("project.cdb")
```

This will open an existing project file called `project.cdb`.

Note: The Examples section demonstrates the use of the API with various examples and provides a good starting point to write custom evaluations.

Attention: To be able to use the API, the clickpoints package has to be installed! If a `ImportError: No module named clickpoints` error is raised, you have to install the package first. Go to `clickpointspackage` in your `clickpoints` directory and execute `python setup.py develop` there.

Database Models

The `.cdb` file consists of multiple SQL tables in which it stores its information. Each table is represented in the API as a peewee model. Users which are not familiar can use the API without any knowledge of peewee, as the API provides all functions necessary to access the data. For each table a `get` (retrieve entries), `set` (add and change entries) and `delete` (remove entries) function is provided. Functions with a plural name always work on multiple entries at once and all arguments can be provided as single values or arrays if multiple entries should be affected.

The tables are: *Meta*, *Path*, *Image*, *Offset*, *Track*, *MarkerType*, *Marker*, *Line*, *Rectangle*, *Mask*, *MaskType*, *Annotation*, *Tag*, *TagAssociation*.

class **Meta**

Stores key value pairs containing meta information for the ClickPoints project.

Attributes:

- **key** (*str, unique*) - the key
- **value** (*str*) - the value for the key

class **Path**

Stores a path. Referenced by each image entry.

See also: `getPath()`, `getPaths()`, `setPath()`, `deletePaths()`.

Attributes:

- **path** (*str, unique*) - the path
- **images** (*list of Image*) - the images with this path.

class **Image**

Stores an image.

See also: `getImage()`, `getImages()`, `getImageIterator()`, `setImage()`, `deleteImages()`.

Attributes:

- **filename** (*str, unique*) - the name of the file.
- **ext** (*str*) - the extension of the file.
- **frame** (*int*) - the frame of the file (0 for images, ≥ 0 for images from videos).
- **external_id** (*int*) - the id of the file entry of a corresponding external database. Only used when ClickPoints is started from an external database.
- **timestamp** (*datetime*) - the timestamp associated to the image.
- **sort_index** (*int, unique*) - the index of the image. The number shown in ClickPoints next to the time line.
- **width** (*int*) - None if it has not be set, otherwise the width of the image.
- **height** (*int*) - None if it has not be set, otherwise the height of the image.
- **path** (*Path*) - the linked path entry containing the path to the image.
- **offset** (*Offset*) - the linked offset entry containing the offsets stored for this image.
- **markers** (*list of Marker*) - a list of marker entries for this image.
- **lines** (*list of Line*) - a list of line entries for this image.
- **rectangles** (*list of Rectangle*) - a list of rectangle entries for this image.
- **mask** (*Mask*) - the mask entry associated with the image.
- **data** (*array*) - the image data as a numpy array. Data will be loaded on demand and cached.
- **data8** (*array, uint8*) - the image data converted to unsigned 8 bit integers.
- **getShape()** (*list*) - a list containing height and width of the image. If they are not stored in the database yet, the image data has to be loaded.

class **Offset**

Offsets associated with an image.

Attributes:

- **image** (*Image*) - the associated image entry.
- **x** (*int*) - the x offset
- **y** (*int*) - the y offset

class **Track**

A track containing multiple markers.

See also: *getTrack()*, *getTracks()*, *setTrack()*, *deleteTracks()*.

Attributes:

- **style** (*str*) - the style for this track.
- **text** (*str*) - an additional text associated with this track. It is displayed next to the markers of this track in ClickPoints.
- **hidden** (*bool*) - whether the track should be displayed in ClickPoints.
- **points** (*array*) - an Nx2 array containing the x and y coordinates of the associated markers.
- **points_corrected** (*array*) - an Nx2 array containing the x and y coordinates of the associated markers corrected by the offsets of the images.
- **markers** (*list of Marker*) - a list containing all the associated markers.
- **times** (*list of datetime*) - a list containing the timestamps for the images of the associated markers.
- **frames** (*list of int*) - a list containing all the frame numbers for the images of the associated markers.
- **image_ids** (*list of int*) - a list containing all the ids for the images of the associated markers.

class **MarkerType**

A marker type.

See also: *getMarkerTypes()*, *getMarkerType()*, *setMarkerType()*, *deleteMarkerTypes()*.

Attributes:

- **name** (*str, unique*) - the name of the marker type.
- **color** (*str*) - the color of the marker in HTML format, e.g. #FF0000 (red).
- **mode** (*int*) - the mode, has to be either: TYPE_Normal, TYPE_Rect, TYPE_Line or TYPE_Track
- **style** (*str*) - the style of the marker type.
- **text** (*str*) - an additional text associated with the marker type. It is displayed next to the markers of this type in ClickPoints.
- **hidden** (*bool*) - whether the markers of this type should be displayed in ClickPoints.
- **markers** (*list of Marker*) - a list containing all markers of this type. Only for TYPE_Normal and TYPE_Track.
- **lines** (*list of Line*) - a list containing all lines of this type. Only for TYPE_Line.
- **markers** (*list of Rectangle*) - a list containing all rectangles of this type. Only for TYPE_Rect.

class **Marker**

A marker.

See also: *getMarker()*, *getMarkers()*, *setMarker()*, *setMarkers()*, *deleteMarkers()*.

Attributes:

- **image** (*Image*) - the image entry associated with this marker.

- **x** (*int*) - the x coordinate of the marker.
- **y** (*int*) - the y coordinate of the marker.
- **type** (*MarkerType*) - the marker type.
- **processed** (*bool*) - a flag that is set to 0 if the marker is manually moved in ClickPoints, it can be set from an add-on if the add-on has already processed this marker.
- **style** (*str*) - the style definition of the marker.
- **text** (*str*) - an additional text associated with the marker. It is displayed next to the marker in ClickPoints.
- **track** (*Track*) - the track entry the marker belongs to. Only for TYPE_Track.
- **correctedXY()** (*array*) - the marker position corrected by the offset of the image.
- **pos()** (*array*) - an array containing the coordinates of the marker: [x, y].

class Line

A line.

See also: *getLine()*, *getLines()*, *setLine()*, *setLines()*, *deleteLines()*.

Attributes:

- **image** (*Image*) - the image entry associated with this line.
- **x1** (*int*) - the first x coordinate of the line.
- **y1** (*int*) - the first y coordinate of the line.
- **x2** (*int*) - the second x coordinate of the line.
- **y2** (*int*) - the second y coordinate of the line.
- **type** (*MarkerType*) - the marker type.
- **processed** (*bool*) - a flag that is set to 0 if the line is manually moved in ClickPoints, it can be set from an add-on if the add-on has already processed this line.
- **style** (*str*) - the style definition of the line.
- **text** (*str*) - an additional text associated with the line. It is displayed next to the line in ClickPoints.
- **correctedXY()** (*array*) - the line positions corrected by the offset of the image.
- **pos()** (*array*) - an array containing the coordinates of the line: [x, y].
- **length** (*float*) - the length of the line in pixel.

class Rectangle

A rectangle.

See also: *getRectangle()*, *getRectangles()*, *setRectangle()*, *setRectangles()*, *deleteRectangles()*.

Attributes:

- **image** (*Image*) - the image entry associated with this rectangle.
- **x** (*int*) - the x coordinate of the rectangle.
- **y** (*int*) - the y coordinate of the rectangle.
- **width** (*int*) - the width of the rectangle.
- **height** (*int*) - the height of the rectangle.

- **type** (*MarkerType*) - the marker type.
- **processed** (*bool*) - a flag that is set to 0 if the rectangle is manually moved in ClickPoints, it can be set from an add-on if the add-on has already processed this line.
- **style** (*str*) - the style definition of the rectangle.
- **text** (*str*) - an additional text associated with the rectangle. It is displayed next to the rectangle in ClickPoints.
- **correctedXY()** (*array*) - the rectangle positions corrected by the offset of the image.
- **pos()** (*array*) - an array containing the coordinates of the rectangle: [x, y].
- **slice_x()** (*slice*) - a slice object to use the rectangle to cut out a region of an image
- **slice_y()** (*slice*) - a slice object to use the rectangle to cut out a region of an image
- **area()** (*float*) - the area of the rectangle

class Mask

A mask entry.

See also: *getMask()*, *getMasks()*, *setMask()*, *deleteMasks()*.

Attributes:

- **image** (*Image*) - the image entry associated with this marker.
- **data** (*array*) - the mask image as a numpy array. Mask types are stored by their index value.

class MaskType

A mask type.

See also: *getMaskType()*, *getMaskTypes()*, *setMaskType()*, *deleteMaskTypes()*.

Attributes:

- **name** (*str*) - the name of the mask type.
- **color** (*str*) - the color of the mask type in HTML format, e.g. #FF0000 (red).
- **index** (*int*) - the integer value used to represent this type in the mask.

class Annotation

An annotation.

See also: *getAnnotation()*, *getAnnotations()*, *setAnnotation()*, *deleteAnnotations()*.

Attributes:

- **image** (*Image*) - the image entry associated with this annotation.
- **timestamp** (*datetime*) - the timestamp of the image linked to the annotation.
- **comment** (*str*) - the text of the comment.
- **rating** (*int*) - the value added to the annotation as rating.
- **tags** (*list of Tag*) - the tags associated with this annotation.

class Tag

A tag for an *Annotation*.

See also: *getTag()*, *getTags()*, *setTag()*, *deleteTags()*.

Attributes:

- **name** (*str*) - the name of the tag.

- **annotations** (*list of [Annotation](#)*) - the annotations associated with this tag.

class TagAssociation

A link between a [Tag](#) and an [Annotation](#)

Attributes:

- **annotation** ([Annotation](#)) - the linked annotation.
- **tag** ([Tag](#)) - the linked tag.

DataFile

class clickpoints.DataFile (*database_filename=None, mode='r'*)

The DataFile class provides access to the .cdb file format in which ClickPoints stores the data for a project.

Parameters

- **database_filename** (*string*) – the filename to open
- **mode** (*string, optional*) – can be ‘r’ (default) to open an existing database and append data to it or ‘w’ to create a new database. If the mode is ‘w’ and the database already exists, it will be deleted and a new database will be created.

deleteAnnotations (*image=None, frame=None, filename=None, timestamp=None, comment=None, rating=None, id=None*)

Delete all [Annotation](#) entries with the given criteria.

See also: [getAnnotation\(\)](#), [getAnnotations\(\)](#), [setAnnotation\(\)](#).

Parameters

- **image** (*int, [Image](#), array_like, optional*) – the image/images for which the annotations should be retrieved. If omitted, frame numbers or filenames should be specified instead.
- **frame** (*int, array_like, optional*) – frame number/numbers of the images, which annotations should be returned. If omitted, images or filenames should be specified instead.
- **filename** (*string, array_like, optional*) – filename of the image/images, which annotations should be returned. If omitted, images or frame numbers should be specified instead.
- **timestamp** (*datetime, array_like, optional*) – timestamp/s of the annotations.
- **comment** (*string, array_like, optional*) – the comment/s of the annotations.
- **rating** (*int, array_like, optional*) – the rating/s of the annotations.
- **id** (*int, array_like, optional*) – id/ids of the annotations.

Returns **rows** – the number of affected rows.

Return type **int**

deleteImages (*filename=None, path=None, frame=None, external_id=None, timestamp=None, width=None, height=None, id=None*)

Delete all [Image](#) entries with the given criteria.

See also: [getImage\(\)](#), [getImages\(\)](#), [getImageIterator\(\)](#), [setImage\(\)](#).

Parameters

- **filename** (*string, array_like, optional*) – the filename/filenames of the image (including the extension)
- **path** (*string, int, [Path](#), array_like optional*) – the path string, id or entry of the image to insert
- **frame** (*int, array_like, optional*) – the number/numbers of frames the images have
- **external_id** (*int, array_like, optional*) – an external id/ids for the images. Only necessary if the annotation server is used
- **timestamp** (*datetime object, array_like, optional*) – the timestamp/timestamps of the images
- **width** (*int, array_like, optional*) – the width/widths of the images
- **height** (*int, optional*) – the height/heights of the images
- **id** (*int, array_like, optional*) – the id/ids of the images

Returns *rows* – the number of affected rows.

Return type *int*

deleteLines (*image=None, frame=None, filename=None, x1=None, y1=None, x2=None, y2=None, type=None, processed=None, text=None, id=None*)

Delete all [Line](#) entries with the given criteria.

See also: [getLine\(\)](#), [getLines\(\)](#), [setLine\(\)](#), [setLines\(\)](#).

Parameters

- **image** (*int, [Image](#), array_like, optional*) – the image/s of the lines.
- **frame** (*int, array_like, optional*) – the frame/s of the images of the lines.
- **filename** (*string, array_like, optional*) – the filename/s of the images of the lines.
- **x1** (*int, array_like, optional*) – the x coordinate/s of the start of the lines.
- **y1** (*int, array_like, optional*) – the y coordinate/s of the start of the lines.
- **x2** (*int, array_like, optional*) – the x coordinate/s of the end of the lines.
- **y2** (*int, array_like, optional*) – the y coordinate/s of the end of the lines.
- **type** (*string, [MarkerType](#), array_like, optional*) – the marker type/s (or name/s) of the lines.
- **processed** (*int, array_like, optional*) – the processed flag/s of the lines.
- **text** (*string, array_like, optional*) – the text/s of the lines.
- **id** (*int, array_like, optional*) – the id/s of the lines.

Returns *rows* – the number of affected rows.

Return type *int*

deleteMarkerTypes (*name=None, color=None, mode=None, text=None, hidden=None, id=None*)

Delete all [MarkerType](#) entries from the database, which match the given criteria.

See also: [getMarkerType\(\)](#), [getMarkerTypes\(\)](#), [setMarkerType\(\)](#).

Parameters

- **name** (*str, array_like, optional*) – the name of the type

- **color** (*str*, *array_like*, *optional*) – hex code string for rgb color of style “#00ff3f”
- **mode** (*int*, *array_like*, *optional*) – mode of the marker type (marker 0, rect 1, line 2, track 4)
- **text** (*str*, *array_like*, *optional*) – display text
- **hidden** (*bool*, *array_like*, *optional*) – whether the types should be displayed in ClickPoints
- **id** (*int*, *array_like*, *optional*) – id of the *MarkerType* object

Returns *entries* – nr of deleted entries

Return type *int*

deleteMarkers (*image=None*, *frame=None*, *filename=None*, *x=None*, *y=None*, *type=None*, *processed=None*, *track=None*, *text=None*, *id=None*)

Delete all *Marker* entries with the given criteria.

See also: *getMarker()*, *getMarkers()*, *setMarker()*, *setMarkers()*.

Parameters

- **image** (*int*, *Image*, *array_like*, *optional*) – the image/s of the markers.
- **frame** (*int*, *array_like*, *optional*) – the frame/s of the images of the markers.
- **filename** (*string*, *array_like*, *optional*) – the filename/s of the images of the markers.
- **x** (*int*, *array_like*, *optional*) – the x coordinate/s of the markers.
- **y** (*int*, *array_like*, *optional*) – the y coordinate/s of the markers.
- **type** (*string*, *MarkerType*, *array_like*, *optional*) – the marker type/s (or name/s) of the markers.
- **processed** (*int*, *array_like*, *optional*) – the processed flag/s of the markers.
- **track** (*int*, *Track*, *array_like*, *optional*) – the track id/s or instance/s of the markers.
- **text** (*string*, *array_like*, *optional*) – the text/s of the markers.
- **id** (*int*, *array_like*, *optional*) – the id/s of the markers.

Returns *rows* – the number of affected rows.

Return type *int*

deleteMaskTypes (*name=None*, *color=None*, *index=None*, *id=None*)

Delete all *MaskType* entries from the database, which match the given criteria.

See also: *getMaskType()*, *getMaskTypes()*, *setMaskType()*.

Parameters

- **name** (*string*, *array_like*, *optional*) – the name/names of the mask types.
- **color** (*string*, *array_like*, *optional*) – the color/colors of the mask types.
- **index** (*int*, *array_like*, *optional*) – the index/indices of the mask types, which is used for painting this mask types.
- **id** (*int*, *array_like*, *optional*) – the id/ids of the mask types.

deleteMasks (*image=None, frame=None, filename=None, id=None*)

Delete all *Mask* entries with the given criteria.

See also: *getMask()*, *getMasks()*, *setMask()*.

Parameters

- **image** (*int, Image, array_like, optional*) – the image/images for which the mask should be deleted. If omitted, frame numbers or filenames should be specified instead.
- **frame** (*int, array_like, optional*) – frame number/numbers of the images, which masks should be deleted. If omitted, images or filenames should be specified instead.
- **filename** (*string, array_like, optional*) – filename of the image/images, which masks should be deleted. If omitted, images or frame numbers should be specified instead.
- **id** (*int, array_like, optional*) – id/ids of the masks.

deletePaths (*path_string=None, base_path=None, id=None*)

Delete all *Path* entries with the given criteria.

See also: *getPath()*, *getPaths()*, *setPath()*

Parameters

- **path_string** (*string, optional*) – the string/s specifying the paths.
- **base_path** (*string, optional*) – return only paths starting with the *base_path* string.
- **id** (*int, optional*) – the id/s of the paths.

Returns *rows* – the number of affected rows.

Return type *int*

deleteRectangles (*image=None, frame=None, filename=None, x=None, y=None, width=None, height=None, type=None, processed=None, text=None, id=None*)

Delete all *Rectangle* entries with the given criteria.

See also: *getRectangle()*, *getRectangles()*, *setRectangle()*, *setRectangles()*.

Parameters

- **image** (*int, Image, array_like, optional*) – the image/s of the rectangles.
- **frame** (*int, array_like, optional*) – the frame/s of the images of the rectangles.
- **filename** (*string, array_like, optional*) – the filename/s of the images of the rectangles.
- **x** (*int, array_like, optional*) – the x coordinate/s of the upper left corner/s of the rectangles.
- **y** (*int, array_like, optional*) – the y coordinate/s of the upper left corner/s of the rectangles.
- **width** (*int, array_like, optional*) – the width/s of the rectangles.
- **height** (*int, array_like, optional*) – the height/s of the rectangles.
- **type** (*string, MarkerType, array_like, optional*) – the marker type/s (or name/s) of the rectangles.

- **processed** (*int, array_like, optional*) – the processed flag/s of the rectangles.
- **text** (*string, array_like, optional*) – the text/s of the rectangles.
- **id** (*int, array_like, optional*) – the id/s of the rectangles.

Returns *rows* – the number of affected rows.

Return type *int*

deleteTags (*name=None, id=None*)

Delete all *Tag* entries from the database, which match the given criteria. If no criteria a given, delete all.

See also: *getTag()*, *getTags()*, *setTag()*.

Parameters

- **name** (*string, array_like, optional*) – the name/names of the *Tag*.
- **id** (*int, array_like, optional*) – the id/ids of the *Tag*.

Returns *rows* – number of rows deleted

Return type *int*

deleteTracks (*type=None, text=None, hidden=None, id=None*)

Delete a single *Track* object specified by id or all *Track* object of an type

See also: *getTrack()*, *getTracks()*, *setTrack()*.

Parameters

- **type** (*MarkerType, str, array_like, optional*) – the marker type or name of the marker type
- **text** (*str, array_like, optional*) – the *Track* specific text entry
- **hidden** (*bool, array_like, optional*) – whether the tracks should be displayed in ClickPoints
- **id** (*int, array_like, array_like, optional*) – the *Track* ID

Returns *rows* – the number of affected rows.

Return type *int*

getAnnotation (*image=None, frame=None, filename=None, id=None, create=False*)

Get the *Annotation* entry for the given image frame number or filename.

See also: *getAnnotations()*, *setAnnotation()*, *deleteAnnotations()*.

Parameters

- **image** (*int, Image, optional*) – the image for which the annotation should be retrieved. If omitted, frame number or filename should be specified instead.
- **frame** (*int, optional*) – frame number of the image, which annotation should be returned. If omitted, image or filename should be specified instead.
- **filename** (*string, optional*) – filename of the image, which annotation should be returned. If omitted, image or frame number should be specified instead.
- **id** (*int, optional*) – id of the annotation entry.
- **create** (*bool, optional*) – whether the annotation should be created if it does not exist. (default: False)

Returns `annotation` – the desired `Annotation` entry.

Return type `Annotation`

getAnnotations (*image=None, frame=None, filename=None, timestamp=None, tag=None, comment=None, rating=None, id=None*)

Get all `Annotation` entries from the database, which match the given criteria. If no criteria a given, return all masks.

See also: `getAnnotation()`, `setAnnotation()`, `deleteAnnotations()`.

Parameters

- **image** (*int, Image, array_like, optional*) – the image/images for which the annotations should be retrieved. If omitted, frame numbers or filenames should be specified instead.
- **frame** (*int, array_like, optional*) – frame number/numbers of the images, which annotations should be returned. If omitted, images or filenames should be specified instead.
- **filename** (*string, array_like, optional*) – filename of the image/images, which annotations should be returned. If omitted, images or frame numbers should be specified instead.
- **timestamp** (*datetime, array_like, optional*) – timestamp/s of the annotations.
- **tag** (*string, array_like, optional*) – the tag/s of the annotations to load.
- **comment** (*string, array_like, optional*) – the comment/s of the annotations.
- **rating** (*int, array_like, optional*) – the rating/s of the annotations.
- **id** (*int, array_like, optional*) – id/ids of the annotations.

Returns `entries` – a query object containing all the matching `Annotation` entries in the database file.

Return type `Annotation`

getDbVersion()

Returns the version of the currently opened database file.

Returns `version` – the version of the database

Return type `string`

getImage (*frame=None, filename=None, id=None*)

Returns the `Image` entry with the given frame number.

See also: `getImages()`, `getImageIterator()`, `setImage()`, `deleteImages()`.

Parameters

- **frame** (*int, optional*) – the frame number of the desired image, as displayed in ClickPoints.
- **filename** (*string, optional*) – the filename of the desired image.
- **id** (*int, optional*) – the id of the image.

Returns `image` – the image entry.

Return type `Image`

getImageIterator (*start_frame=0, end_frame=None*)

Get an iterator to iterate over all *Image* entries starting from *start_frame*.

See also: *getImage()*, *getImages()*, *setImage()*, *deleteImages()*.

Parameters

- **start_frame** (*int, optional*) – start at the image with the number *start_frame*. Default is 0
- **end_frame** (*int, optional*) – the last frame of the iteration (excluded). Default is None, the iteration stops when no more images are present.

Returns *image_iterator* – an iterator object to iterate over *Image* entries.

Return type *iterator*

Examples

```
1 import clickpoints
2
3 # open the database "data.cdb"
4 db = clickpoints.DataFile("data.cdb")
5
6 # iterate over all images and print the filename
7 for image in db.getImageIterator():
8     print(image.filename)
```

getImages (*frame=None, filename=None, ext=None, external_id=None, timestamp=None, width=None, height=None, path=None, order_by='sort_index'*)

Get all *Image* entries sorted by sort index. For large databases *getImageIterator()*, should be used as it doesn't load all frames at once.

See also: *getImage()*, *getImageIterator()*, *setImage()*, *deleteImages()*.

Parameters

- **frame** (*int, array_like, optional*) – the frame number/s of the image/s as displayed in ClickPoints (*sort_index* in the database).
- **filename** (*string, array_like, optional*) – the filename/s of the image/s.
- **ext** (*string, array_like, optional*) – the extension/s of the image/s.
- **external_id** (*int, array_like, optional*) – the external id/s of the image/s.
- **timestamp** (*datetime, array_like, optional*) – the timestamp/s of the image/s.
- **width** (*int, array_like, optional*) – the width/s of the image/s.
- **height** (*int, array_like, optional*) – the height/s of the image/s.
- **path** (*int, Path, array_like, optional*) – the path/s (or path id/s) of the image/s.
- **order_by** (*string, optional*) – sort by either 'sort_index' (default) or 'timestamp'.

Returns *entries* – a query object containing all the *Image* entries in the database file.

Return type *array_like*

getLine (*id*)

Retrieve an *Line* object from the database.

See also: *getLines()*, *setLine()*, *setLines()*, *deleteLines()*.

Parameters *id* (*int*) – the id of the line

Returns *line* – the *Line* with the desired id or None.

Return type *Line*

getLines (*image=None*, *frame=None*, *filename=None*, *x1=None*, *y1=None*, *x2=None*, *y2=None*, *type=None*, *processed=None*, *text=None*, *id=None*)

Get all *Line* entries with the given criteria.

See also: *getLine()*, *setLine()*, *setLines()*, *deleteLines()*.

Parameters

- **image** (*int*, *Image*, *array_like*, *optional*) – the image/s of the lines.
- **frame** (*int*, *array_like*, *optional*) – the frame/s of the images of the lines.
- **filename** (*string*, *array_like*, *optional*) – the filename/s of the images of the lines.
- **x1** (*int*, *array_like*, *optional*) – the x coordinate/s of the lines start.
- **y1** (*int*, *array_like*, *optional*) – the y coordinate/s of the lines start.
- **x2** (*int*, *array_like*, *optional*) – the x coordinate/s of the lines end.
- **y2** (*int*, *array_like*, *optional*) – the y coordinate/s of the lines end.
- **type** (*string*, *MarkerType*, *array_like*, *optional*) – the marker type/s (or name/s) of the lines.
- **processed** (*int*, *array_like*, *optional*) – the processed flag/s of the lines.
- **text** (*string*, *array_like*, *optional*) – the text/s of the lines.
- **id** (*int*, *array_like*, *optional*) – the id/s of the lines.

Returns *entries* – a query object which contains all *Line* entries.

Return type *array_like*

getMarker (*id*)

Retrieve an *Marker* object from the database.

See also: *getMarkers()*, *setMarker()*, *setMarkers()*, *deleteMarkers()*.

Parameters *id* (*int*) – the id of the marker

Returns *marker* – the *Marker* with the desired id or None.

Return type *Marker*

getMarkerType (*name=None*, *id=None*)

Retrieve an *MarkerType* object from the database.

See also: *getMarkerTypes()*, *setMarkerType()*, *deleteMarkerTypes()*.

Parameters

- **name** (*str*, *optional*) – the name of the desired type
- **id** (*int*, *optional*) – id of the *MarkerType* object

Returns *entries* – the *MarkerType* with the desired name or None.

Return type `array_like`

getMarkerTypes (*name=None, color=None, mode=None, text=None, hidden=None, id=None*)

Retrieve all *MarkerType* objects in the database.

See also: *getMarkerType()*, *setMarkerType()*, *deleteMarkerTypes()*.

Parameters

- **name** (*str, array_like, optional*) – the name of the type
- **color** (*str, array_like, optional*) – hex code string for rgb color of style “#00ff3f”
- **mode** (*int, array_like, optional*) – mode of the marker type (marker 0, rect 1, line 2, track 4)
- **text** (*str, array_like, optional*) – display text
- **hidden** (*bool, array_like, optional*) – whether the types should be displayed in ClickPoints
- **id** (*int, array_like, optional*) – id of the *MarkerType* object

Returns **entries** – a query object which contains all *MarkerType* entries.

Return type `array_like`

getMarkers (*image=None, frame=None, filename=None, x=None, y=None, type=None, processed=None, track=None, text=None, id=None*)

Get all *Marker* entries with the given criteria.

See also: *getMarker()*, *getMarkers()*, *setMarker()*, *setMarkers()*, *deleteMarkers()*.

Parameters

- **image** (*int, Image, array_like, optional*) – the image/s of the markers.
- **frame** (*int, array_like, optional*) – the frame/s of the images of the markers.
- **filename** (*string, array_like, optional*) – the filename/s of the images of the markers.
- **x** (*int, array_like, optional*) – the x coordinate/s of the markers.
- **y** (*int, array_like, optional*) – the y coordinate/s of the markers.
- **type** (*string, MarkerType, array_like, optional*) – the marker type/s (or name/s) of the markers.
- **processed** (*int, array_like, optional*) – the processed flag/s of the markers.
- **track** (*int, Track, array_like, optional*) – the track id/s or instance/s of the markers.
- **text** (*string, array_like, optional*) – the text/s of the markers.
- **id** (*int, array_like, optional*) – the id/s of the markers.

Returns **entries** – a query object which contains all *Marker* entries.

Return type `array_like`

getMask (*image=None, frame=None, filename=None, id=None, create=False*)

Get the *Mask* entry for the given image frame number or filename.

See also: *getMasks()*, *setMask()*, *deleteMasks()*.

Parameters

- **image** (*int*, *Image*, *optional*) – the image for which the mask should be retrieved. If omitted, frame number or filename should be specified instead.
- **frame** (*int*, *optional*) – frame number of the image, which mask should be returned. If omitted, image or filename should be specified instead.
- **filename** (*string*, *optional*) – filename of the image, which mask should be returned. If omitted, image or frame number should be specified instead.
- **id** (*int*, *optional*) – id of the mask entry.
- **create** (*bool*, *optional*) – whether the mask should be created if it does not exist. (default: False)

Returns **mask** – the desired *Mask* entry.

Return type *Mask*

getMaskType (*name=None, color=None, index=None, id=None*)

Get a *MaskType* from the database.

See also: *getMaskTypes()*, *setMaskType()*, *deleteMaskTypes()*.

Parameters

- **name** (*string*, *optional*) – the name of the mask type.
- **color** (*string*, *optional*) – the color of the mask type.
- **index** (*int*, *optional*) – the index of the mask type, which is used for painting this mask type.
- **id** (*int*, *optional*) – the id of the mask type.

Returns **entries** – the created/requested *MaskType* entry.

Return type *MaskType*

getMaskTypes (*name=None, color=None, index=None, id=None*)

Get all *MaskType* entries from the database, which match the given criteria. If no criteria a given, return all mask types.

See also: *getMaskType()*, *setMaskType()*, *deleteMaskTypes()*.

Parameters

- **name** (*string*, *array_like*, *optional*) – the name/names of the mask types.
- **color** (*string*, *array_like*, *optional*) – the color/colors of the mask types.
- **index** (*int*, *array_like*, *optional*) – the index/indices of the mask types, which is used for painting this mask types.
- **id** (*int*, *array_like*, *optional*) – the id/ids of the mask types.

Returns **entries** – a query object containing all the matching *MaskType* entries in the database file.

Return type *array_like*

getMasks (*image=None, frame=None, filename=None, id=None, order_by='sort_index'*)

Get all *Mask* entries from the database, which match the given criteria. If no criteria a given, return all masks.

See also: *getMask()*, *setMask()*, *deleteMasks()*.

Parameters

- **image** (*int, Image, array_like, optional*) – the image/images for which the mask should be retrieved. If omitted, frame numbers or filenames should be specified instead.
- **frame** (*int, array_like, optional*) – frame number/numbers of the images, which masks should be returned. If omitted, images or filenames should be specified instead.
- **filename** (*string, array_like, optional*) – filename of the image/images, which masks should be returned. If omitted, images or frame numbers should be specified instead.
- **id** (*int, array_like, optional*) – id/ids of the masks.
- **order_by** (*string, optional*) – sorts the result according to sort paramter ('sort_index' or 'timestamp')

Returns **entries** – a query object containing all the matching *Mask* entries in the database file.

Return type *Mask*

getPath (*path_string=None, id=None, create=False*)

Get a *Path* entry from the database.

See also: *getPaths()*, *setPath()*, *deletePaths()*

Parameters

- **path_string** (*string, optional*) – the string specifying the path.
- **id** (*int, optional*) – the id of the path.
- **create** (*bool, optional*) – whether the path should be created if it does not exist. (default: False)

Returns **path** – the created/requested *Path* entry.

Return type *Path*

getPaths (*path_string=None, base_path=None, id=None*)

Get all *Path* entries from the database, which match the given criteria. If no criteria a given, return all paths.

See also: *getPath()*, *setPath()*, *deletePaths()*

Parameters

- **path_string** (*string, path_string, optional*) – the string/s specifying the path/s.
- **base_path** (*string, optional*) – return only paths starting with the base_path string.
- **id** (*int, array_like, optional*) – the id/s of the path/s.

Returns **entries** – a query object containing all the matching *Path* entries in the database file.

Return type *array_like*

getRectangle (*id*)

Retrieve an *Rectangle* object from the database.

See also: *getRectangles()*, *setRectangle()*, *setRectangles()*, *deleteRectangles()*.

Parameters **id** (*int*) – the id of the rectangle.

Returns `rectangle` – the *Rectangle* with the desired id or None.

Return type *Rectangle*

getRectangles (*image=None, frame=None, filename=None, x=None, y=None, width=None, height=None, type=None, processed=None, text=None, id=None*)

Get all *Rectangle* entries with the given criteria.

See also: *getRectangle()*, *setRectangle()*, *setRectangles()*, *deleteRectangles()*.

Parameters

- **image** (*int, Image, array_like, optional*) – the image/s of the rectangles.
- **frame** (*int, array_like, optional*) – the frame/s of the images of the rectangles.
- **filename** (*string, array_like, optional*) – the filename/s of the images of the rectangles.
- **x** (*int, array_like, optional*) – the x coordinate/s of the upper left corner/s of the rectangles.
- **y** (*int, array_like, optional*) – the y coordinate/s of the upper left corner/s of the rectangles.
- **width** (*int, array_like, optional*) – the width/s of the rectangles.
- **height** (*int, array_like, optional*) – the height/s of the rectangles.
- **type** (*string, MarkerType, array_like, optional*) – the marker type/s (or name/s) of the rectangles.
- **processed** (*int, array_like, optional*) – the processed flag/s of the rectangles.
- **text** (*string, array_like, optional*) – the text/s of the rectangles.
- **id** (*int, array_like, optional*) – the id/s of the rectangles.

Returns `entries` – a query object which contains all *Rectangle* entries.

Return type *array_like*

getTag (*name=None, id=None*)

Get a specific *Tag* entry by its name or database ID

See also: *getTags()*, *setTag()*, *deleteTags()*.

Parameters

- **name** (*str*) – name of the tag
- **id** (*int*) – id of *Tag* entry

Returns `entries` – requested object of class *Tag* or None

Return type *Tag*

getTags (*name=None, id=None*)

Get all *Tag* entries from the database, which match the given criteria. If no criteria a given, return all.

See also: *getTag()*, *setTag()*, *deleteTags()*.

Parameters

- **name** (*string, array_like, optional*) – the name/names of the *Tag*.
- **id** (*int, array_like, optional*) – the id/ids of the *Tag*.

Returns `entries` – a query object containing all the matching `Tag` entries in the database file.

Return type `array_like`

getTrack (*id*)

Get a specific `Track` entry by its database ID.

See also: `getTracks()`, `deleteTracks()`.

Parameters `id` (*int*) – id of the track

Returns `entries` – requested object of class `Track` or `None`

Return type `Track`

getTracks (*type=None, text=None, hidden=None, id=None*)

Get all `Track` entries, optional filter by type

See also: `getTrack()`, `setTrack()`, `deleteTracks()`.

Parameters

- **type** (*MarkerType*, *str*, *array_like*, *optional*) – the marker type/types or name of the marker type for the track.
- **text** (*str*, *array_like*, *optional*) – the `Track` specific text entry
- **hidden** (*bool*, *array_like*, *optional*) – whether the tracks should be displayed in ClickPoints
- **id** (*int*, *array_like*, *optional*) – the `Track` ID

Returns `entries` – a query object which contains the requested `Track`.

Return type `array_like`

max_sql_variables ()

Get the maximum number of arguments allowed in a query by the current sqlite3 implementation. Based on ‘this question ‘_

Returns inferred `SQLITE_MAX_VARIABLE_NUMBER`

Return type `int`

setAnnotation (*image=None, frame=None, filename=None, timestamp=None, comment=None, rating=None, id=None*)

Insert or update an `Annotation` object in the database.

See also: `getAnnotation()`, `getAnnotations()`, `deleteAnnotations()`.

Parameters

- **image** (*int*, *Image*, *optional*) – the image of the annotation.
- **frame** (*int*, *optional*) – the frame of the images of the annotation.
- **filename** (*string*, *optional*) – the filename of the image of the annotation.
- **timestamp** (*datetime*, *optional*) – the timestamp of the annotation.
- **comment** (*string*, *optional*) – the text of the annotation.
- **rating** (*int*, *optional*) – the rating of the annotation.
- **id** (*int*, *optional*) – the id of the annotation.

Returns `annotation` – the created or changed `Annotation` item.

Return type `Annotation`

setImage (*filename=None, path=None, frame=None, external_id=None, timestamp=None, width=None, height=None, id=None*)

Update or create new *Image* entry with the given parameters.

See also: *getImage()*, *getImages()*, *getImageIterator()*, *deleteImages()*.

Parameters

- **filename** (*string, optional*) – the filename of the image (including the extension)
- **path** (*string, int, Path, optional*) – the path string, id or entry of the image to insert
- **frame** (*int, optional*) – the frame number if the image is part of a video
- **external_id** (*int, optional*) – an external id for the image. Only necessary if the annotation server is used
- **timestamp** (*datetime object, optional*) – the timestamp of the image
- **width** (*int, optional*) – the width of the image
- **height** (*int, optional*) – the height of the image
- **id** (*int, optional*) – the id of the image

Returns *image* – the changed or created *Image* entry

Return type *Image*

setLine (*image=None, frame=None, filename=None, x1=None, y1=None, x2=None, y2=None, type=None, processed=None, style=None, text=None, id=None*)

Insert or update an *Line* object in the database.

See also: *getLine()*, *getLines()*, *setLines()*, *deleteLines()*.

Parameters

- **image** (*int, Image, optional*) – the image of the line.
- **frame** (*int, optional*) – the frame of the images of the line.
- **filename** (*string, optional*) – the filename of the image of the line.
- **x1** (*int, optional*) – the x coordinate of the start of the line.
- **y1** (*int, optional*) – the y coordinate of the start of the line.
- **x2** (*int, optional*) – the x coordinate of the end of the line.
- **y2** (*int, optional*) – the y coordinate of the end of the line.
- **type** (*string, MarkerType, optional*) – the marker type (or name) of the line.
- **processed** (*int, optional*) – the processed flag of the line.
- **text** (*string, optional*) – the text of the line.
- **id** (*int, optional*) – the id of the line.

Returns *line* – the created or changed *Line* item.

Return type *Line*

setLines (*image=None, frame=None, filename=None, x1=None, y1=None, x2=None, y2=None, type=None, processed=None, style=None, text=None, id=None*)

Insert or update multiple *Line* objects in the database.

See also: *getLine()*, *getLines()*, *setLine()*, *deleteLines()*.

Parameters

- **image** (*int*, *Image*, *array_like*, *optional*) – the image/s of the lines.
- **frame** (*int*, *array_like*, *optional*) – the frame/s of the images of the lines.
- **filename** (*string*, *array_like*, *optional*) – the filename/s of the images of the lines.
- **x1** (*int*, *array_like*, *optional*) – the x coordinate/s of the start of the lines.
- **y1** (*int*, *array_like*, *optional*) – the y coordinate/s of the start of the lines.
- **x2** (*int*, *array_like*, *optional*) – the x coordinate/s of the end of the lines.
- **y2** (*int*, *array_like*, *optional*) – the y coordinate/s of the end of the lines.
- **type** (*string*, *MarkerType*, *array_like*, *optional*) – the marker type/s (or name/s) of the lines.
- **processed** (*int*, *array_like*, *optional*) – the processed flag/s of the lines.
- **track** (*int*, *Track*, *array_like*, *optional*) – the track id/s or instance/s of the lines.
- **text** (*string*, *array_like*, *optional*) – the text/s of the lines.
- **id** (*int*, *array_like*, *optional*) – the id/s of the lines.

Returns *success* – it the inserting was successful.

Return type *bool*

setMarker (*image=None*, *frame=None*, *filename=None*, *x=None*, *y=None*, *type=None*, *processed=None*, *track=None*, *style=None*, *text=None*, *id=None*)

Insert or update an *Marker* object in the database.

See also: *getMarker()*, *getMarkers()*, *setMarkers()*, *deleteMarkers()*.

Parameters

- **image** (*int*, *Image*, *optional*) – the image of the marker.
- **frame** (*int*, *optional*) – the frame of the images of the marker.
- **filename** (*string*, *optional*) – the filename of the image of the marker.
- **x** (*int*, *optional*) – the x coordinate of the marker.
- **y** (*int*, *optional*) – the y coordinate of the marker.
- **type** (*string*, *MarkerType*, *optional*) – the marker type (or name) of the marker.
- **processed** (*int*, *optional*) – the processed flag of the marker.
- **track** (*int*, *Track*, *optional*) – the track id or instance of the marker.
- **text** (*string*, *optional*) – the text of the marker.
- **id** (*int*, *optional*) – the id of the marker.

Returns *marker* – the created or changed *Marker* item.

Return type *Marker*

setMarkerType (*name=None*, *color=None*, *mode=None*, *style=None*, *text=None*, *hidden=None*, *id=None*)

Insert or update an *MarkerType* object in the database.

See also: *getMarkerType()*, *getMarkerTypes()*, *deleteMarkerTypes()*.

Parameters

- **name** (*str*, *optional*) – the name of the type
- **color** (*str*, *optional*) – hex code string for rgb color of style “#00ff3f”
- **mode** (*int*, *optional*) – mode of the marker type (marker 0, rect 1, line 2, track 4)
- **style** (*str*, *optional*) – style string
- **text** (*str*, *optional*) – display text
- **hidden** (*bool*, *optional*) – whether the type should be displayed in ClickPoints
- **id** (*int*, *optional*) – id of the *MarkerType* object

Returns *entries* – the created *MarkerType* with the desired name or None.

Return type object

setMarkers (*image=None*, *frame=None*, *filename=None*, *x=None*, *y=None*, *type=None*, *processed=None*, *track=None*, *style=None*, *text=None*, *id=None*)

Insert or update multiple *Marker* objects in the database.

See also: *getMarker()*, *getMarkers()*, *setMarker()*, *deleteMarkers()*.

Parameters

- **image** (*int*, *Image*, *array_like*, *optional*) – the image/s of the markers.
- **frame** (*int*, *array_like*, *optional*) – the frame/s of the images of the markers.
- **filename** (*string*, *array_like*, *optional*) – the filename/s of the images of the markers.
- **x** (*int*, *array_like*, *optional*) – the x coordinate/s of the markers.
- **y** (*int*, *array_like*, *optional*) – the y coordinate/s of the markers.
- **type** (*string*, *MarkerType*, *array_like*, *optional*) – the marker type/s (or name/s) of the markers.
- **processed** (*int*, *array_like*, *optional*) – the processed flag/s of the markers.
- **track** (*int*, *Track*, *array_like*, *optional*) – the track id/s or instance/s of the markers.
- **text** (*string*, *array_like*, *optional*) – the text/s of the markers.
- **id** (*int*, *array_like*, *optional*) – the id/s of the markers.

Returns *success* – it the inserting was successful.

Return type bool

setMask (*image=None*, *frame=None*, *filename=None*, *data=None*, *id=None*)

Update or create new *Mask* entry with the given parameters.

See also: *getMask()*, *getMasks()*, *deleteMasks()*.

Parameters

- **image** (*int*, *Image*, *optional*) – the image for which the mask should be set. If omitted, frame number or filename should be specified instead.
- **frame** (*int*, *optional*) – frame number of the images, which masks should be set. If omitted, image or filename should be specified instead.

- **filename** (*string, optional*) – filename of the image, which masks should be set. If omitted, image or frame number should be specified instead.
- **data** (*ndarray, optional*) – the mask data of the mask to set. Must have the same dimensions as the corresponding image, but only one channel, and it should be using the data type uint8.
- **id** (*int, optional*) – id of the mask entry.

Returns **mask** – the changed or created *Mask* entry.

Return type *Mask*

setMaskType (*name=None, color=None, index=None, id=None*)

Update or create a new a *MaskType* entry with the given parameters.

See also: *getMaskType()*, *getMaskTypes()*, *setMaskType()*, *deleteMaskTypes()*.

Parameters

- **name** (*string, optional*) – the name of the mask type.
- **color** (*string, optional*) – the color of the mask type.
- **index** (*int, optional*) – the index of the mask type, which is used for painting this mask type.
- **id** (*int, optional*) – the id of the mask type.

Returns **entries** – the changed or created *MaskType* entry.

Return type *MaskType*

setPath (*path_string=None, id=None*)

Update or create a new *Path* entry with the given parameters.

See also: *getPath()*, *getPaths()*, *deletePaths()*

Parameters

- **path_string** (*string, optional*) – the string specifying the path.
- **id** (*int, optional*) – the id of the paths.

Returns **entries** – the changed or created *Path* entry.

Return type *Path*

setRectangle (*image=None, frame=None, filename=None, x=None, y=None, width=None, height=None, type=None, processed=None, style=None, text=None, id=None*)

Insert or update an *Rectangle* object in the database.

See also: *getRectangle()*, *getRectangles()*, *setRectangles()*, *deleteRectangles()*.

Parameters

- **image** (*int, Image, optional*) – the image of the rectangle.
- **frame** (*int, optional*) – the frame of the images of the rectangle.
- **filename** (*string, optional*) – the filename of the image of the rectangle.
- **x** (*int, optional*) – the x coordinate of the upper left corner of the rectangle.
- **y** (*int, optional*) – the y coordinate of the upper left of the rectangle.
- **width** (*int, optional*) – the width of the rectangle.

- **height** (*int*, *optional*) – the height of the rectangle.
- **type** (*string*, *MarkerType*, *optional*) – the marker type (or name) of the rectangle.
- **processed** (*int*, *optional*) – the processed flag of the rectangle.
- **text** (*string*, *optional*) – the text of the rectangle.
- **id** (*int*, *optional*) – the id of the rectangle.

Returns *rectangle* – the created or changed *Rectangle* item.

Return type *Rectangle*

setRectangles (*image=None*, *frame=None*, *filename=None*, *x=None*, *y=None*, *width=None*, *height=None*, *type=None*, *processed=None*, *style=None*, *text=None*, *id=None*)
 Insert or update multiple *Rectangle* objects in the database.

See also: *getRectangle()*, *getRectangles()*, *setRectangle()*, *deleteRectangles()*.

Parameters

- **image** (*int*, *Image*, *array_like*, *optional*) – the image/s of the rectangles.
- **frame** (*int*, *array_like*, *optional*) – the frame/s of the images of the rectangles.
- **filename** (*string*, *array_like*, *optional*) – the filename/s of the images of the rectangles.
- **x** (*int*, *array_like*, *optional*) – the x coordinate/s of the upper left corner/s of the rectangles.
- **y** (*int*, *array_like*, *optional*) – the y coordinate/s of the upper left corner/s of the rectangles.
- **width** (*int*, *array_like*, *optional*) – the width/s of the rectangles.
- **height** (*int*, *array_like*, *optional*) – the height/s of the rectangles.
- **type** (*string*, *MarkerType*, *array_like*, *optional*) – the marker type/s (or name/s) of the rectangles.
- **processed** (*int*, *array_like*, *optional*) – the processed flag/s of the rectangles.
- **track** (*int*, *Track*, *array_like*, *optional*) – the track id/s or instance/s of the rectangles.
- **text** (*string*, *array_like*, *optional*) – the text/s of the rectangles.
- **id** (*int*, *array_like*, *optional*) – the id/s of the rectangles.

Returns *success* – if the inserting was successful.

Return type *bool*

setTag (*name=None*, *id=None*)

Set a specific *Tag* entry by its name or database ID

See also: *getTag()*, *getTags()*, *deleteTags()*.

Parameters

- **name** (*str*) – name of the tag
- **id** (*int*) – id of *Tag* entry

Returns *entries* – object of class *Tag*

Return type *Tag*

setTrack (*type*, *style=None*, *text=None*, *hidden=None*, *id=None*, *uid=None*)

Insert or update a *Track* object.

See also: *getTrack()*, *getTracks()*, *deleteTracks()*.

Parameters

- **type** (*MarkerType*, str) – the marker type or name of the marker type for the track.
- **style** – the *Track* specific style entry
- **text** – the *Track* specific text entry
- **hidden** – whether the track should be displayed in ClickPoints
- **id** (*int*, *array_like*) – the *Track* ID

Returns *track* – a new *Track* object

Return type track object

CHAPTER 8

Add-on API

ClickPoints allows to easily write add-on scripts. They are called from ClickPoints with command line arguments specifying which database to use, at which frame to start and how to communicate with ClickPoints.

The add-on script should start as follows:

```
1 import clickpoints
2 start_frame, database, port = clickpoints.GetCommandLineArgs()
3 db = clickpoints.DataFile(database)
4 com = clickpoints.Commands(port, catch_terminate_signal=True)
```

This will retrieve `start_frame`, `database` and `port` from the command line arguments the script was started with. When executing the script through the add-on interface, ClickPoints will provide these values. These can then be used to open the ClickPoints project file and establish a connection to the ClickPoints instance.

Note: The Addons section demonstrates how the add-ons can be used and may serve as a good starting point to write custom add-ons.

Attention: To be able to use the API, the clickpoints package has to be installed! If a `ImportError: No module named clickpoints` error is raised, you have to install the package first. Go to `clickpointspackage` in your clickpoints directory and execute `python setup.py develop` there.

GetCommandLineArgs

`clickpoints.GetCommandLineArgs()`

Parse the command line arguments for the information provided by ClickPoints, if the script is invoked from within ClickPoints. The arguments are `-start_frame` `-database` and `-port`.

Returns

- **start_frame** (*int*) – the frame ClickPoints was in when invoking the script. Probably the evaluation should start here
- **database** (*string*) – the filename of the database where the current ClickPoints project is stored. Should be used with `clickpoints.DataFile`
- **port** (*int*) – the port of the socket connection to communicate with the ClickPoints instance. Should be used with `clickpoints.Commands`

Commands

class `clickpoints.Commands` (*port=None, catch_terminate_signal=False*)

The `Commands` class provides an interface for external scripts to communicate with a currently open ClickPoints instance. Communication is done using socket connection. ClickPoints provides the port number for this connection when calling an external script. Use `clickpoints.GetCommandLineArgs` to obtain the port number.

Parameters

- **port** (*int, optional*) – the port for the socket connection to communicate with ClickPoints. If it is not provided, a dummy connection is used with doesn't pass any commands. This behaviour is provided to enable external scripts to run with and without a ClickPoints instance.
- **catch_terminate_signal** (*bool, optional*) – whether a terminate signal from ClickPoints should directly terminate the script (default) or if only the `terminate_signal` flag should be set. This flag can later on be queried with `HasTerminateSignal()`

CatchTerminateSignal ()

Catch the terminate signal when ClickPoints wants to close the script execution. When called at the beginning of the script, the signal is cached and its status can be queried with `HasTerminateSignal`. This can be used for a gentle program termination, where the current progress loop can be finished before stopping the program execution.

GetImage (*value*)

Get the currently in ClickPoints displayed image.

Returns

- **image** (*ndarray*) – the image data.
- **image_id** (*int*) – the image id in the database.
- **image_frame** (*int*) – which frame is used if the image is from a video file. 0 if the source is an image file.

HasTerminateSignal ()

Whether or not the program has received a terminate signal from ClickPoints. Can only be used if `CatchTerminateSignal` was called before.

Returns `terminate_signal` – True if ClickPoints has sent a terminate signal.

Return type `bool`

JumpFrames (*value*)

Let ClickPoints jump the given amount of frames.

Parameters **value** (*int*) – the amount of frame which ClickPoints should jump.

JumpFramesWait (*value*)

Let ClickPoints jump the given amount of frames and wait for it to complete.

Parameters **value** (*int*) – the amount of frame which ClickPoints should jump.

JumpToFrame (*value*)

Let ClickPoints jump to the given frame.

Parameters **value** (*int*) – the frame to which ClickPoints should jump.

JumpToFrameWait (*value*)

Let ClickPoints jump to the given frame and wait for it to complete.

Parameters **value** (*int*) – the frame to which ClickPoints should jump.

ReloadMarker (*frame=None*)

Reloads the marker from the given frame in ClickPoints.

Parameters **frame** (*int*) – the frame which ClickPoints should reload.

ReloadMask ()

Reloads the current mask file in ClickPoints.

ReloadTypes ()

Reloads the marker types.

log (**args*)

Print to the ClickPoints console.

Parameters ***args** (*string*) – multiple strings to print

updateHUD (*value*)

CHAPTER 9

Citing ClickPoints

If you use ClickPoints for academic research, you are highly encouraged (though not required) to cite the following paper:

- Gerum, R., Richter, S., Fabry, B. and Zitterbart, D.P. (2016), “[ClickPoints: an expandable toolbox for scientific image annotation and analysis](#)”. *Methods Ecol Evol.* doi:10.1111/2041-210X.12702

ClickPoints is developed primarily by academics, and so citations matter a lot to us. Citing ClickPoints also increases it's exposure and potential user (and developer) base, which is to the benefit of all users of ClickPoints. Thanks in advance!

CHAPTER 10

Note

If you encounter any bugs or unexpected behaviour, you are encouraged to report a bug in our Bitbucket [bugtracker](#).

Bibliography

- [1] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.
- [1] Celine Le Bohec. Programme 137 ‘ecophy-antavia’ of the french polar institute paul-emile victor (ipev).
- [1] Nadjia Gerlitz. Dronpa. 2016.
- [1] Navid Bonakdar, Richard Gerum, Michael Kuhn, Marina Spörrer, Anna Lippert, Werner Schneider, Katerina E Aifantis, and Ben Fabry. Mechanical plasticity of cells. *Nature Materials*, 2016.

A

Annotation (built-in class), 45

C

CatchTerminateSignal() (clickpoints.Commands method), 66

Commands (class in clickpoints), 66

D

DataFile (class in clickpoints), 46

deleteAnnotations() (clickpoints.DataFile method), 46

deleteImages() (clickpoints.DataFile method), 46

deleteLines() (clickpoints.DataFile method), 47

deleteMarkers() (clickpoints.DataFile method), 48

deleteMarkerTypes() (clickpoints.DataFile method), 47

deleteMasks() (clickpoints.DataFile method), 48

deleteMaskTypes() (clickpoints.DataFile method), 48

deletePaths() (clickpoints.DataFile method), 49

deleteRectangles() (clickpoints.DataFile method), 49

deleteTags() (clickpoints.DataFile method), 50

deleteTracks() (clickpoints.DataFile method), 50

G

getAnnotation() (clickpoints.DataFile method), 50

getAnnotations() (clickpoints.DataFile method), 51

GetCommandLineArgs() (in module clickpoints), 65

getDbVersion() (clickpoints.DataFile method), 51

getImage() (clickpoints.Commands method), 66

getImage() (clickpoints.DataFile method), 51

getImageIterator() (clickpoints.DataFile method), 51

getImages() (clickpoints.DataFile method), 52

getLine() (clickpoints.DataFile method), 52

getLines() (clickpoints.DataFile method), 53

getMarker() (clickpoints.DataFile method), 53

getMarkers() (clickpoints.DataFile method), 54

getMarkerType() (clickpoints.DataFile method), 53

getMarkerTypes() (clickpoints.DataFile method), 54

getMask() (clickpoints.DataFile method), 54

getMasks() (clickpoints.DataFile method), 55

getMaskType() (clickpoints.DataFile method), 55

getMaskTypes() (clickpoints.DataFile method), 55

getPath() (clickpoints.DataFile method), 56

getPaths() (clickpoints.DataFile method), 56

getRectangle() (clickpoints.DataFile method), 56

getRectangles() (clickpoints.DataFile method), 57

getTag() (clickpoints.DataFile method), 57

getTags() (clickpoints.DataFile method), 57

getTrack() (clickpoints.DataFile method), 58

getTracks() (clickpoints.DataFile method), 58

H

HasTerminateSignal() (clickpoints.Commands method), 66

I

Image (built-in class), 42

J

JumpFrames() (clickpoints.Commands method), 66

JumpFramesWait() (clickpoints.Commands method), 66

JumpToFrame() (clickpoints.Commands method), 67

JumpToFrameWait() (clickpoints.Commands method), 67

L

Line (built-in class), 44

log() (clickpoints.Commands method), 67

M

Marker (built-in class), 43

MarkerType (built-in class), 43

Mask (built-in class), 45

MaskType (built-in class), 45

max_sql_variables() (clickpoints.DataFile method), 58

Meta (built-in class), 41

O

Offset (built-in class), 42

P

Path (built-in class), [42](#)

R

Rectangle (built-in class), [44](#)

ReloadMarker() (clickpoints.Commands method), [67](#)

ReloadMask() (clickpoints.Commands method), [67](#)

ReloadTypes() (clickpoints.Commands method), [67](#)

S

setAnnotation() (clickpoints.DataFile method), [58](#)

setImage() (clickpoints.DataFile method), [58](#)

setLine() (clickpoints.DataFile method), [59](#)

setLines() (clickpoints.DataFile method), [59](#)

setMarker() (clickpoints.DataFile method), [60](#)

setMarkers() (clickpoints.DataFile method), [61](#)

setMarkerType() (clickpoints.DataFile method), [60](#)

setMask() (clickpoints.DataFile method), [61](#)

setMaskType() (clickpoints.DataFile method), [62](#)

setPath() (clickpoints.DataFile method), [62](#)

setRectangle() (clickpoints.DataFile method), [62](#)

setRectangles() (clickpoints.DataFile method), [63](#)

setTag() (clickpoints.DataFile method), [63](#)

setTrack() (clickpoints.DataFile method), [64](#)

T

Tag (built-in class), [45](#)

TagAssociation (built-in class), [46](#)

Track (built-in class), [43](#)

U

updateHUD() (clickpoints.Commands method), [67](#)