# The EyeLink Plugin for OpenSesame

# User Manual

Version 0.6

Last updated on May 31, 2021

## Licensing Info

The EyeLink plugin for OpenSesame was developed to help users to control and to communicate with the EyeLink trackers. The current release does not contain any eye-event triggers (e.g. wait for a fixation of 200 ms in a pre-defined Interest Area to initiate the presentation of stimulus), but it has most of the frequently used functions and configuration options of the tracker. We encourage users to download the version released on the SR Support website, which implements the tracking practice and options recommended by SR Research, but the user is allowed to modify the source code to meet their experimental needs. Please see the licensing info below.

# 1 Installation

This plugin has been tested on Windows 10, macOS 11.3.1 and Ubuntu 20.04[1], but should work for any operating systems supported by OpenSesame.

1) Download and install the latest version of the EyeLink Developers Kit from the SR Support website, under Downloads: EyeLink Developers Kit / API (https://www.sr-support.com/thread-13.html). It includes the EyeLink API, the OpenSesame plugin, and example OpenSesame experiments illustrating how to use the plugin to interface with EyeLink systems.

2) After installation of the EyeLink Developers Kit, locate the OpenSesame folder containing the EyeLink plugin and examples:

   Windows: C:\Program Files (x86)\SR Research\EyeLink\SampleExperiments\Python\examples\OpenSesame

   macOS: /Applications/Eyelink/SampleExperiments/Python/examples/OpenSesame

   Ubuntu: /usr/share/EyeLink/SampleExperiments/Python/examples/OpenSesame

3) Copy all the folders in the openSesame_plugin folder (i.e. *el_CamSetup*, *el_Connect*, …, *el_StopRecording* folders) to:

   Windows: C:\Program Files (x86)\OpenSesame\share\opensesame_plugins\

   macOS: /Applications/OpenSesame.app/Contents/Resources/share/opensesame_plugins/

   Ubuntu: /usr/share/opensesame_plugins/

   Note, on Ubuntu to ensure you have permissions to paste these folders, you can use the terminal to copy the files using the following command:

   ```
   sudo cp -r /usr/share/EyeLink/SampleExperiments/Python/examples/OpenSesame/openSesame _plugin/el*
   /usr/share/opensesame_plugins/
   ```

4) OpenSesame needs the PyLink library to communicate with the tracker. PyLink is a Python wrapper of the EyeLink API (part of the EyeLink Developers Kit). To install the PyLink library, please do the following:

[1]For Ubuntu 20.04 don't exit the experiment through the ESCAPE key as this will not properly close the connection with the Host PC. This will cause a connection error and require a reboot of both Host and Display PC.

Windows:

Different versions of the PyLink library, for different versions of Python can be found in the following folder after the EyeLink Developers Kit for Windows has been installed:

```
C:\Program Files (x86)\SR Research\EyeLink\SampleExperiments\Python\64
```

There are multiple subfolders in this folder with two-digit names (e.g., 2.7, 3.6, 3.7). The two digits in the folder name indicate the Python version it was built against (e.g., the subfolder 3.7 contains the PyLink library for Python 3.7). When you launch OpenSesame, you can see the version of Python that it is using in the Console. Please copy the “pylink” folder from the corresponding version folder and paste it to the following folder:

```
C:\Program Files (x86)\OpenSesame\Lib\site-packages
```

macOS:

Different versions of the PyLink library, for different versions of Python can be found in the following folder after the EyeLink Developers Kit for macOS has been installed:

```
/Applications/EyeLink/SampleExperiments/Python
```

There are multiple subfolders in this folder with two-digit names (e.g., 2.7, 3.6, 3.7). The two digits in the folder name indicate the Python version it was built against (e.g., the subfolder 3.7 contains the PyLink library for Python 3.7). When you launch OpenSesame, you can see the version of Python that it is using in the Console. Please copy the “pylink” folder from the corresponding version folder and paste it to the following folder:

```
/Applications/OpenSesame/Contents/Resources/lib/[python version]/site-packages
```

(where [python version] will reflect the Python version that OpenSesame is using, e.g., python3.7)

Ubuntu:

Open up a terminal and enter the following:

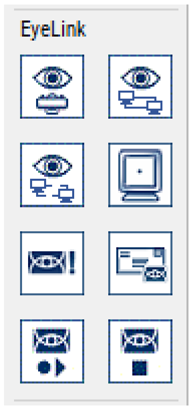
```
python3 /usr/share/EyeLink/SampleExperiments/Python/install_pylink.py
```

5) After the above steps, please be sure to set the IP address of the experimental PC to “100.1.1.2” (without quotes) and Subnet mask to “255.255.255.0” (without quotes) so the Display PC is on the same network as the EyeLink Host PC (https://www.sr-support.com/thread-281.html). Open the example experiment that comes with the EyeLink plugin to test the link between the two machines.

OpenSesame supports Pygame (legacy), Psychopy, and Expyriment as its backend. We encourage users to use the Psychopy backend, as it is robust and supports frequently used visual stimuli for visual psychophysics (e.g., gratings).

# 2 Usage of the Plugin

After the Plugin has been installed, one should see eight items come up in the item toolbar of the OpenSesame interface.



To use the plugin, simply drag one of these items to the required location in the experiment sequence. For general cognitive tasks, we recommend that users follow these integration steps:

1) **Experiment level**
   a) Connect to the tracker when the script initializes. Please use the *el_Connect* item for this task. The configuration options available for this item will be elaborated in the next section.
   b) This will help the user to maintain optimal tracking accuracy. For fMRI or tasks in which interruption of the task should be avoided, users can calibrate the tracker once at the beginning of each run / session. The item for this function is *el_CamSetup*. This item will help users to transfer the camera image to the experimental PC, to adjust the pupil / CR threshold by using hot keys on the

   experimental PC keyboard, to calibrate the tracker and validate the calibration results.

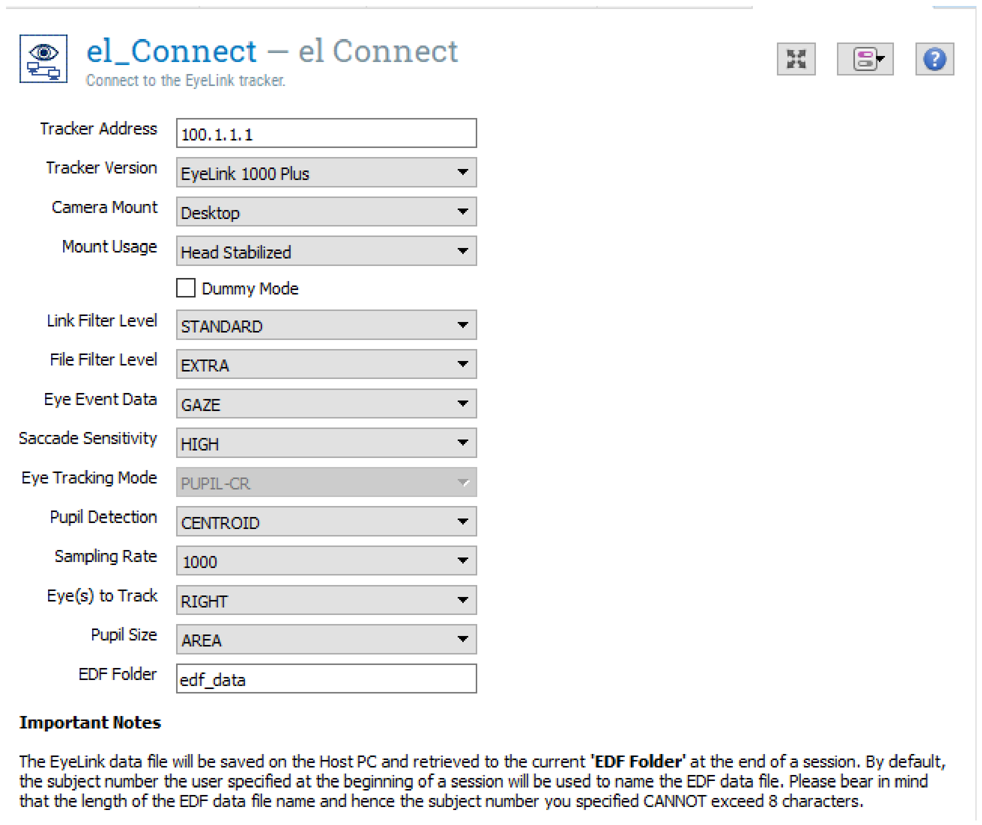
   c) Run the experimental trials one-by-one and record eye movement data.
   d) Disconnect from the Eyelink Host PC and transfer the data file to the experimental PC.

2) **Trial level**
   a) Show backdrop on the Host PC. Some people would like to see the real time gaze over a background image or landmarks on the Host PC during recording. This is optional, but can be implemented at the beginning of each trial, so it won't affect trial event timing. We recommend using an inline script here for flexibility.
   b) Drift-correction or drift-check. This procedure will check the tracking accuracy and give the user a chance to re-calibrate the tracker, if needed.
   c) Start recording. We start and stop recording at the beginning and end of each trial, so the inter-trial intervals won't be recorded; this will reduce the size of the EDF data file. For EEG and tasks where continuous recording is preferred, please start recording at the beginning of each run / session. When start recording, the user also has the option to send a "recording status" message to the tracker; this message will be shown in the bottom-right corner of the Host PC screen.
   d) Draw experimental graphics and send messages to the tracker to mark the onset of these stimuli, and maybe also the interest areas that will be used in data analysis. This is IMPORTANT, otherwise, there is no way to tell when and what stimuli was presented from the eye movement data file.
   e) Collect subject responses and send messages to the tracker.
   f) Stop recording and send all experimental variables to the tracker. These variables will be accessible from the Data Viewer software, a powerful data analysis and visualization tool developed by SR Research. The *el_stopRecording* node will automatically send all variables to the tracker; for user-defined variables, an inline script is needed.

We have provided example scripts with all the recommended usage of the tracker. The functions of each of the items in the plugin are briefly explained below.

## 2.1 el_Connect

Establish a link to the EyeLink Host PC, configure the tracker, and automatically open a data file on the Host to record the eye movement data.



| | |
|---|---|
| Tracker Address | 100.1.1.1 |
| Tracker Version | EyeLink 1000 Plus |
| Camera Mount | Desktop |
| Mount Usage | Head Stabilized |
| | ☐ Dummy Mode |
| Link Filter Level | STANDARD |
| File Filter Level | EXTRA |
| Eye Event Data | GAZE |
| Saccade Sensitivity | HIGH |
| Eye Tracking Mode | PUPIL-CR |
| Pupil Detection | CENTROID |
| Sampling Rate | 1000 |
| Eye(s) to Track | RIGHT |
| Pupil Size | AREA |
| EDF Folder | edf_data |

**Important Notes**

The EyeLink data file will be saved on the Host PC and retrieved to the current '**EDF Folder**' at the end of a session. By default, the subject number the user specified at the beginning of a session will be used to name the EDF data file. Please bear in mind that the length of the EDF data file name and hence the subject number you specified CANNOT exceed 8 characters.

The options that can be set with this item are explained in the table below.

| | |
|---|---|
| Tracker Address | The IP address of the Eyelink Host PC. The default IP address of the Host PC is 100.1.1.1; the IP address of the experimental PC should be in the same network of the Host PC, i.e., in the range of 100.1.1.2-255. |
| Tracker Version | The model of EyeLink tracker being used for data collection: Eyelink I, EyeLink II, EyeLink 1000, EyeLink 1000 Plus, or EyeLink Portable Duo. Some configuration options may not be available for certain models, e.g., sampling rate cannot be set from the EyeLink Plugin for EyeLink I and II, and the Pupil-only tracking method is not available in EyeLink 1000, 1000 Plus and Portable Duo. |
| Camera Mount | The mounting solution of the EyeLink camera, i.e., Desktop, Tower, Arm, Long-range. |

| | |
|---|---|
| Mount Usage | Tracking in either Head Stabilized or Remote (head free to move) mode. The remote mode is unavailable for Tower and Long-range mounts. Tracking in remote mode requires the use of a target sticker on the subject’s forehead. |
| Dummy Mode | Run the tracker in “simulation” mode, i.e., no physical connection to the tracker is established. In Dummy Mode, the user should press F1 to skip Camera setup/calibration, and the drift-correction/check target will briefly flash and then disappear (as no tracker is physically connected to the experimental PC). |
| Link Filter Level | The EyeLink trackers utilizes a heuristic filtering algorithm for denoise purposes (see Stampe[2], 1993). Each increase in filter level reduces noise by a factor of 2 to 3 but introduces a 1-sample delay to the data available over the link. The default option is set to STANDARD, but users can turn off Link Filter if real time online access of gaze data is critical. |
| File Filter Level | Same as above, but applies to the data recorded in file. |
| Eye Event Data | Set how velocity information for saccade detection is to be computed. This option is almost always set to GAZE. Please see the EyeLink user manual (section 4) for details of various eye events (e.g., fixation, saccade). |
| Saccade Sensitivity | Sensitivity of the Eyelink online parse, see Section 4.3.9 of the user manual. For Eyelink II and newer models, HIGH-velocity=22 deg/sec, acceleration=3200 deg/sec$^2$; NORMAL-velocity=30 deg/sec, acceleration=8000 deg/sec$^2$. |
| Eye Tracking Mode | Select the tracking algorithm. EyeLink I operates in Pupil-only mode, while EyeLink II operates in either Pupil-only or Pupil-CR mode. EyeLink 1000 and newer models all operate in Pupil-CR mode. The Pupil-CR tracking algorithm is resilient to small head movements (i.e., drift-free to certain extent). This is why force drift-correcting the tracker is not recommended for EyeLInk 1000 and newer models. |
| Pupil Detection | Set the algorithm used to detect the pupil center. This option only applies to EyeLink 1000 and newer models. |
| Sampling rate | The sampling rate of the tracker. |
| Eye(s) to Track | Set the eyes to track; can be changed on the Host PC manually. |

[2] Stampe, D.M. (1993) Heuristic filtering and reliable calibration methods for video-based pupil-tracking systems. *Behavior Research Methods, Instruments, & Computers*, 25(2), 137-142.

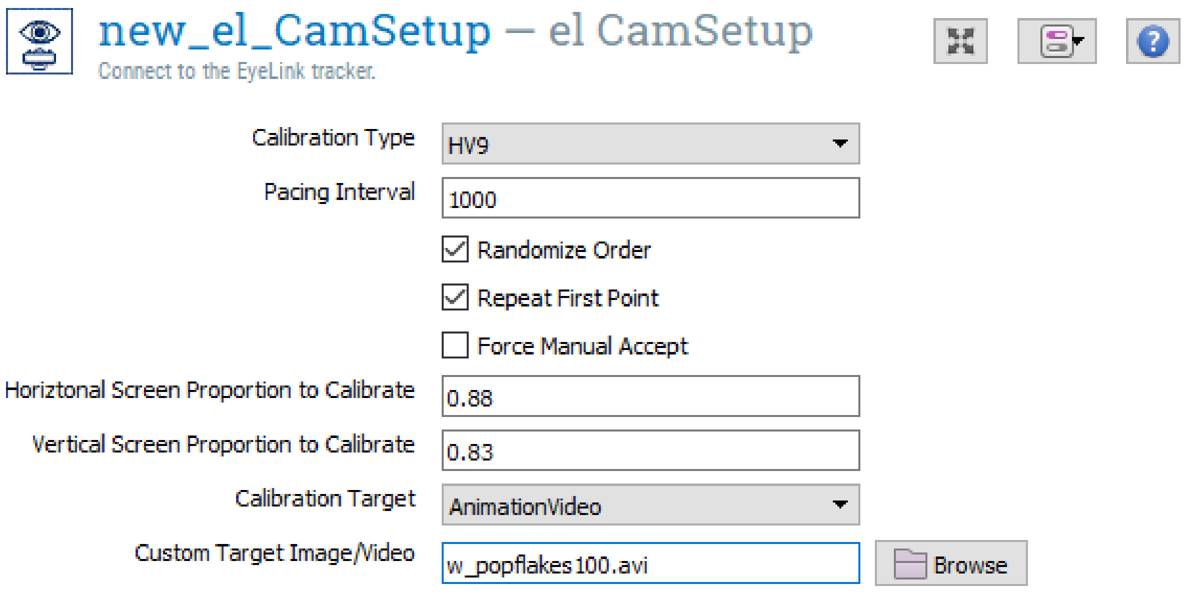
|  |  |
|---|---|
| Pupil Size | Record the pupil size in arbitrary units, AREA and DIAMETER measures are equivalent: DIAMETER = 256*SQRT(AREA/PI). The pupil size recorded in the data files is in arbitrary units; calibration during testing is required if one needs to report pupil size in real world units (i.e., mm). Please see this SR Support forum post (https://www.sr-support.com/thread-154.html). |
| EDF Folder | The EyeLink data file will be saved on the Host PC and retrieved to the current 'EDF Folder' at the end of a session. By default, the subject number the user specified at the beginning of a session will be used to name the EDF data file. Please bear in mind that the length of the EDF data file name and hence the subject number you specified CANNOT exceed 8 characters. |

## 2.2 el_Disconnect

This item will disconnect from the EyeLink Host PC and retrieve the EDF data file over the link. No option to configure for this item.

## 2.3 el_CamSetup

This item wraps all the functions a user may need to calibrate the tracker. Using animation calibration target (video) has not yet been implemented. One should put this item at the beginning of each block of trials. The configuration options are explained in the table below.

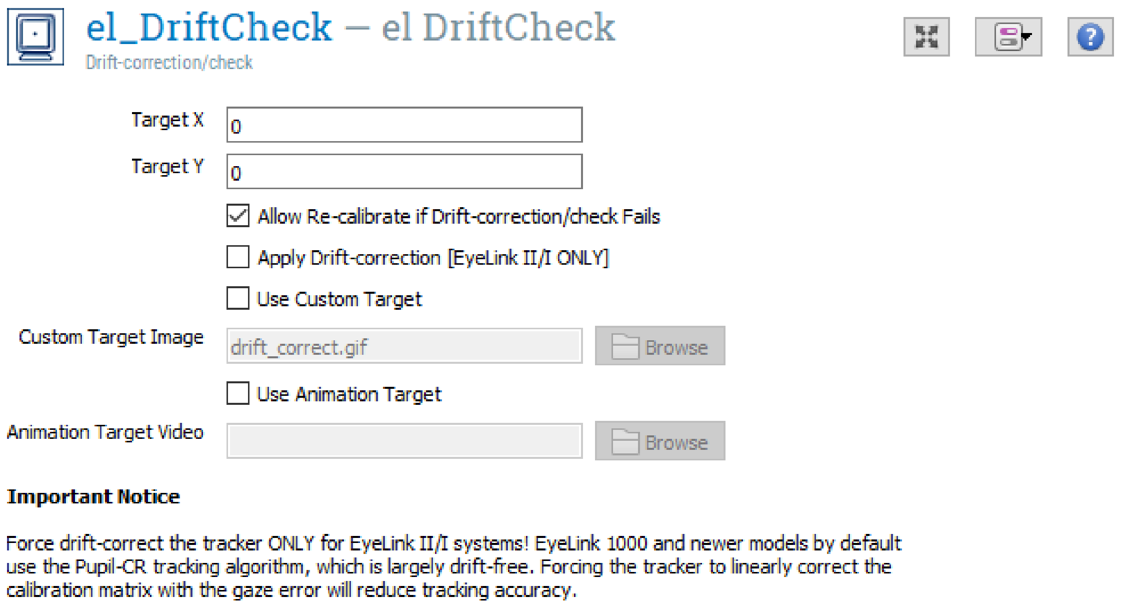

new_el_CamSetup — el CamSetup
Connect to the EyeLink tracker.

Calibration Type: HV9
Pacing Interval: 1000
[✓] Randomize Order
[✓] Repeat First Point
[ ] Force Manual Accept
Horiztonal Screen Proportion to Calibrate: 0.88
Vertical Screen Proportion to Calibrate: 0.83
Calibration Target: AnimationVideo
Custom Target Image/Video: w_popflakes100.avi

| | |
|---|---|
| Calibration Type | Select the calibration type, i.e, HV9 for a 9-point calibration. When tracking in remote mode, it is recommended to use HV13, whereas in head-stabilized mode, HV9 gives the best calibration. |
| Pacing Interval | Set the pacing interval for the calibration / validation targets, i.e., after how much time will the next calibration target be presented after the current calibration target has been accepted. |
| Randomize Order | Randomize the order of the calibration / validation targets |
| Repeat First Point | Repeat the first point. This option is enabled by default, and helps to improve calibration results. |
| Force Manual Accept | Manually accept fixation duration calibration / validation by pressing SPACEBAR or ENTER. One can switch to automatic mode at any time during calibration / validation by pressing “A” on the Host or experimental PC keyboard. |
| Horizontal Screen Proportion to Calibrate | The horizontal proportion of the screen to calibrate. This option is useful when the subject display is large and the top corners may be outside the tracking range of the tracker. One can manually specify the calibration / validation target positions, but this is the recommended way of controlling the size of the calibrated screen region. |
| Vertical Screen Proportion to Calibrate | The horizontal proportion of the screen to calibrate. |
| Calibration Target | Select which type of calibration target to use. The default is a bull’s eye shaped dot, but one can also use an image or a video as the calibration target. |
| Custom Target Image/Video | Select an image or a video file from the File Pool to use as the calibration target. |

## 2.4 el_DriftCheck

This item helps to drift-correct the tracker. The EyeLink II and EyeLink I trackers mount the eye camera on the headband and are susceptible to gaze-drifts as the headband may slip during recording. Drift-correction helps to maintain the tracking accuracy. For EyeLink 1000 and newer models, the tracker uses the Pupil-CR tracking method by default. This method is relatively drift-free and there is no need to drift-correct the tracker; by default the drift-correction routine only checks the gaze error without linearly correcting the gaze data. The various configuration options of this item are listed in the table below.



Target X: 0

Target Y: 0

[✓] Allow Re-calibrate if Drift-correction/check Fails

[ ] Apply Drift-correction [EyeLink II/I ONLY]

[ ] Use Custom Target

Custom Target Image: drift_correct.gif

[ ] Use Animation Target

Animation Target Video:

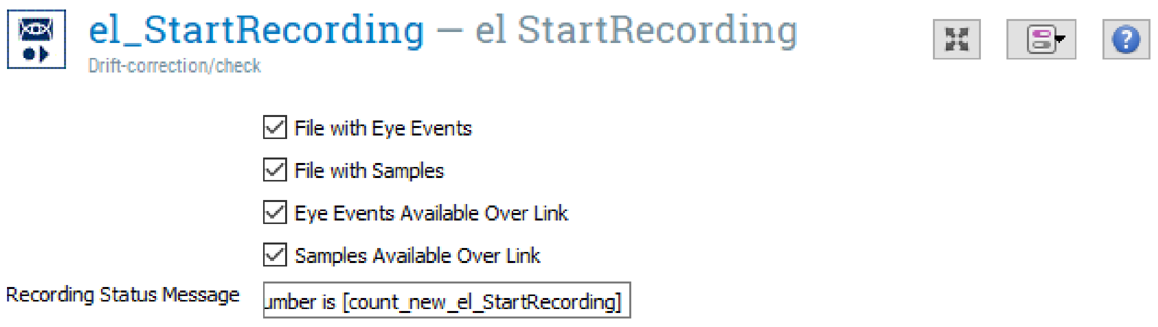
**Important Notice**

Force drift-correct the tracker ONLY for EyeLink II/I systems! EyeLink 1000 and newer models by default use the Pupil-CR tracking algorithm, which is largely drift-free. Forcing the tracker to linearly correct the calibration matrix with the gaze error will reduce tracking accuracy.

| | |
|---|---|
| Target X / Y | The X, Y coordinates of the drift-correction / check target in OpenSesame's default screen coordinates (i.e., 0,0 correspond to the center of the screen). The drift-correction / check target does not necessarily need to be presented at the center of the screen. |
| Allow Re-calibrate if Drift-correction / check Fails | Allow the user to press ESCAPE  key to setup the tracker and to re-calibrate. |
| Apply Drift-correction | Force the tracker to drift-correct only when using EyeLInk I / II. |
| Use Custom Target | Use an image from the File Pool as the drift-correction / check target. |
| Custom Target Image | Select an image file to use as the drift-correction target. |
| Use Animation Target | Use a video as the drift-correction target. |
| Animation Target Video | Select a video to use as the drift-correction target. |

## 2.5 el_StartRecording

Start the track in recording mode. One may also send a status message to the tracker to show the current condition/trial, or the progress of the task on the Host PC screen. One can also specify what types of data is available over the link during recording, and what types of data are saved in EDF data files.


el_StartRecording — el StartRecording

Drift-correction/check

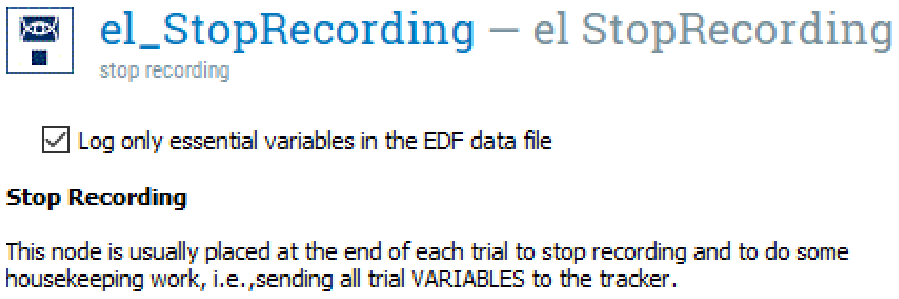
- [✓] File with Eye Events
- [✓] File with Samples
- [✓] Eye Events Available Over Link
- [✓] Samples Available Over Link

Recording Status Message: umber is [count_new_el_StartRecording]

| | |
|---|---|
| File with Eye Events | Store event data in the EDF data file. |
| File with Samples | Store event data in the EDF data file. |
| Eye Events Available Over Link | Allows accessing event data over the link during recording. |
| Samples Available Over Link | Allows accessing sample data over the link during recording. |
| Recording Status Message | Send a message to the Host PC screen to show the current trial number, condition, etc. |

## 2.6 el_StopRecording

This item is usually placed at the end of a trial to stop the recording of eye movement data, and to do some housekeeping work. This node helps also to log the variables generated by OpenSesame into the EDF data files. One may choose to log only essential variables instead of all OpenSesame variables. A friendly note here is that user-defined variables are not logged by OpenSesame by default, so an inline script is recommended for logging these variables in the EDF data file. Please see the example OpenSesame projects for details.


el_StopRecording — el StopRecording

stop recording

☑ Log only essential variables in the EDF data file

**Stop Recording**

This node is usually placed at the end of each trial to stop recording and to do some housekeeping work, i.e.,sending all trial VARIABLES to the tracker.

| | |
|---|---|
| Log only essential variables in the EDF data file | An OpenSesame testing session would generate a large number of variables, some are essential to data analysis whereas others are not. Tick this option to record only the important variables in the EDF data file. |

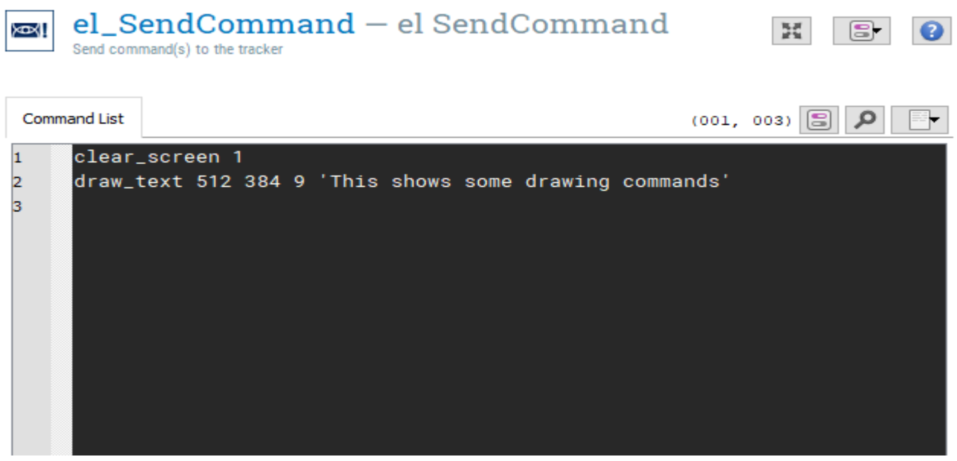
## 2.7 el_SendCommand

Send commands to the tracker. If you need to send multiple commands, put each command in a line, for instance,

```
sampling_rate 500
draw_cross 512 384
```

The various 'draw' commands can be very useful and one can use them to draw simple landmarks on the Host display during recording. These commands (e.g., clear_screen, draw_line, draw_box, draw_text) can be found in the COMMANDS.INI file on the Host PC, under /elcl/exe. One can also send various commands to configure the tracker options, for instance, setting the sampling rate to 500 Hz (see the example above).

**This node is not as flexible as inline scripts; please see the OpenSesame projects accompanying the plugin for example code.**


el_SendCommand — el SendCommand
Send command(s) to the tracker
Command List
(001, 003)

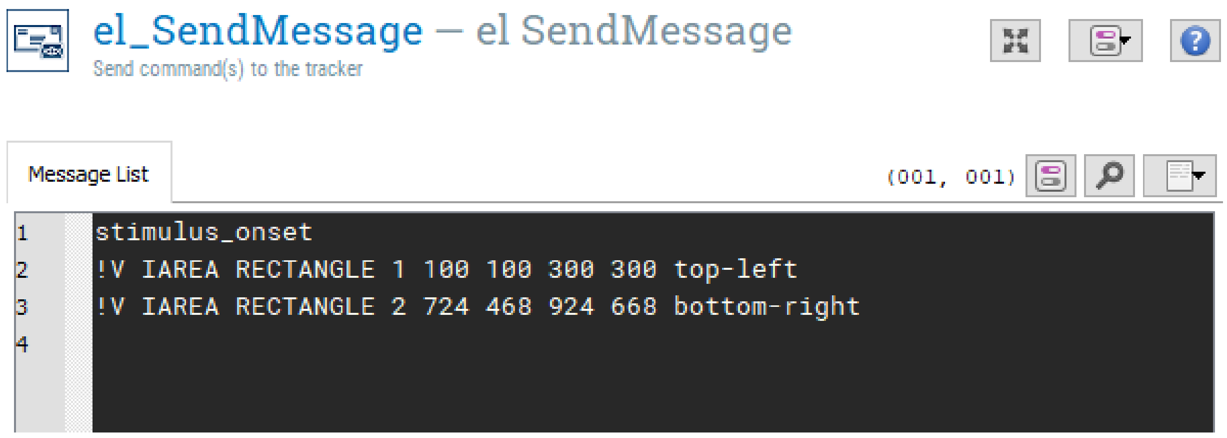
```
clear_screen 1
draw_text 512 384 9 'This shows some drawing commands'
```

## 2.8 el_SendMessage

Messages are very IMPORTANT and we need messages in the DATA FILE to tell what events happened during a trial and at what time. Messages should be sent to the tracker everytime a stimulus screen is on or a response has been issued. A message should not exceed 113 characters. One can send multiple messages, please put one message in a line.

One can also send “Data Viewer Integration Messages” to the tracker. These messages will be used by the Data Viewer software, a data analysis and visualization tool provided by SR Research to load the interest areas, background images, etc. Please see the Data Viewer user manual for a full list of Data Viewer integration messages (see the “Protocol for EyeLink Data to Viewer Integration” section).

**This node is not as flexible as inline scripts; please see the OpenSesame projects accompanying the plugin for example code.**


el_SendMessage — el SendMessage
Send command(s) to the tracker
Message List
(001, 001)

```
stimulus_onset
!V IAREA RECTANGLE 1 100 100 300 300 top-left
!V IAREA RECTANGLE 2 724 468 924 668 bottom-right
```

# 3 A few useful tips

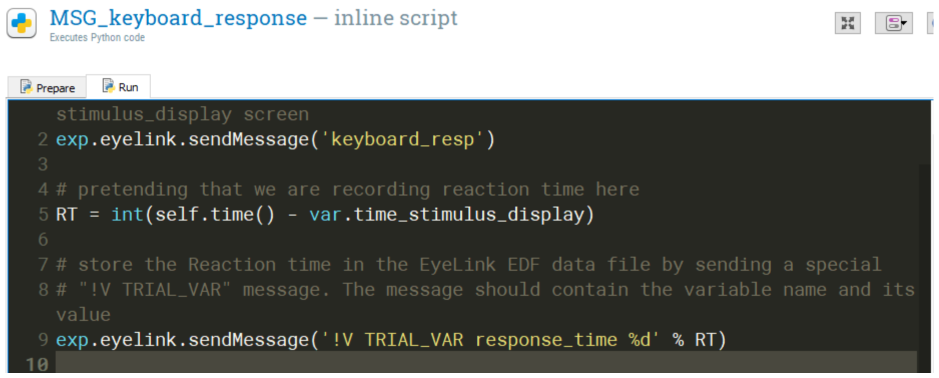
## 3.1 Referring to the tracker from inline scripts

The tracker instance initialized by the plugin can be accessed from inline scripts by referencing “self.experiment.eyelink”. For instance, putting the following commands in an inline script will clear the screen of the Host PC and draw a cross at the center.

```
exp.eyelink.sendCommand('clear_screen 1')
exp.eyelink.sendCommand('draw_cross 512 384 15')
```

For all the attributes and methods of the tracker instance, please refer to the PyLink API User Guide for details. The PyLink API User Guide is packaged within the EyeLink Developers Kit and can be accessed from:

| | |
|---|---|
| Windows: | Start Menu -> All Programs -> SR Research-> Manuals |
| macOS: | /Applications/Eyelink/SampleExperiments/Python/ |
| Linux: | /usr/share/EyeLink/SampleExperiments/Python/ |



```
stimulus_display screen
exp.eyelink.sendMessage('keyboard_resp')

# pretending that we are recording reaction time here
RT = int(self.time() - var.time_stimulus_display)

# store the Reaction time in the EyeLink EDF data file by sending a special
# "!V TRIAL_VAR" message. The message should contain the variable name and its
value
exp.eyelink.sendMessage('!V TRIAL_VAR response_time %d' % RT)
```

The ability to access the tracker from inline scripts allows users to access all functions wrapped in the PyLink library. Be sure to put your custom code in the “Run” rather than the “Prepare” tab.

## 3.2 Send a backdrop image to the Host

One needs to use an inline script to send an image to the host to use as the recording backdrop. In the “picture” example, we have the following lines of code at the beginning of each trial. We read in the image we would like to send to the tracker with PIL (Python Image Library), then convert the image pixels into a format that is recognizable by the EyeLink Host. We then call the bitmapBackdrop() command to send over the image.

```
import PIL.Image as PIL_Image
import pylink
#backdrop image on the Host PC
img = pool[var.image]
img = PIL_Image.open(img)
w,h = img.size
pixels = img.load()
# use the list comprehension trick to convert all image pixels into a <pixel>
format
# supported by the Host PC, pixels = [line1, ...lineH], line = [pix1,...pixW],
pix=(R,G,B)
pixels_2transfer = [[pixels[i, j] for i in range(w)] for j in range(h)]
exp.eyelink.sendCommand('clear_screen 0') # clear the host screen
# call the bitmapBackdrop() command to show backdrop image on the Host
# arguments: width, height, pixel, crop_x, crop_y, crop_width, crop_height, x, y on
Host, option
pos_x = int(var.width/2.0 - w/2.0)
pos_y = int(var.height/2 - h/2.0)
exp.eyelink.bitmapBackdrop(w, h, pixels_2transfer, 0, 0, w, h, pylink.SV_NOREPLACE,
pos_x, pos_y, pylink.BX_MAXCONTRAST)
```

The bitmapBackdrop() command works for all EyeLink trackers. If you have the EyeLink Developers Kit version 2.0 and a compatible PyLink library installed alongside OpenSesame, you may also use the imageBackdrop() command instead. The advantage of this new command is that it no longer requires converting images to the pixel format noted above; the downside of this new command is that you cannot scale the images on the Host PC screen. For compatibility consideration, we have commented out this command in the “picture” example.

```
# show a backdrop image on the Host screen, imageBackdrop() the recommended
# function, if you do not need to scale the image on the Host
# parameters: image_file, crop_x, crop_y, crop_width, crop_height,
#             x, y on the Host, drawing options
#
# img_path = pool[var.image]
# el_tracker.imageBackdrop(img_path, 0, 0, scn_width, scn_height,
#                          pylink.BX_MAXCONTRAST)
```

## 3.3 Drawing simple landmarks on the Host PC

Instead of sending pictures over to the Host, it is usually sufficient to have landmarks drawn on the Host PC. The “Visual World Task” illustrates how to do so with an inline script. The commands we need to send to the tracker are simple, but you may need to do some math to figure out “where” to draw the landmarks. In the example “Visual World

Task", we draw four boxes on the Host to mark the position of all images shown to the participants, then we put labels in the boxes, so the experimenter knows which image is the target.

### 3.4 Interest Area definitions

It is IMPORTANT to keep in mind that the Interest Area definitions recognizable by the Data Viewer software requires users to specify the Interest Area in a screen coordinate where 0,0 is the top-left corner of the screen. In OpenSesame, the origin of the default screen coordinates is the center of the screen instead. So, one needs to do a bit of math when creating interest area messages. This feature is illustrated in the "Visual World Task" example. We have four images, so we use a for-loop to figure out the position of each image, then we send the interest area definition messages to the tracker.

# 4 The picture example

This is a simple passive viewing task. We show an image on the screen; the subject presses a key to see the next image. The structure of the task is fairly simple, we connect to the tracker, calibrate the tracker, do the trials, then disconnect from the tracker. In each trial, we send the backdrop image to the Host, then perform drift-check, start recording, show the image, wait for a keyboard response, then record the variables and stop recording.

### 4.1 Inline script for backdrop image

The only bit worth noting is the inline script we used to send the backdrop image to the tracker. The script is presented in Section 3.2. The code is highly reusable; users only need to change the line that specifies the image file they would like to use as backdrop. In the picture example, the image file names are stored in a data column named "image", the following line will grab the image from the "pool" and use it as the backdrop.

```
import PIL.Image as PIL_Image
import pylink
#backdrop image on the Host PC
img = pool[var.image]
img = PIL_Image.open(img)
w,h = img.size
pixels = img.load()
```

```
# use the list comprehension trick to convert all image pixels into a <pixel>
format
# supported by the Host PC, pixels = [line1, ...lineH], line = [pix1,...pixW],
pix=(R,G,B)
pixels_2transfer = [[pixels[i, j] for i in range(w)] for j in range(h)]
exp.eyelink.sendCommand('clear_screen 0') # clear the host screen
# call the bitmapBackdrop() command to show backdrop image on the Host
# arguments: width, height, pixel, crop_x, crop_y, crop_width, crop_height, x, y on
Host, option
pos_x = int(var.width/2.0 - w/2.0)
pos_y = int(var.height/2 - h/2.0)
exp.eyelink.bitmapBackdrop(w, h, pixels_2transfer, 0, 0, w, h, pylink.SV_NOREPLACE,
pos_x, pos_y, pylink.BX_MAXCONTRAST)
```

## 4.2 Showing background image in Data Viewer

Data Viewer is a powerful data analysis and visualization tool. Sometimes it is helpful to have the background image when examining the gaze data and to create visualizations, e.g., a heatmap.

To record the background image of each trial, you need to send an “IMGLOAD” message to the tracker. The format of the IMGLOAD message can be found in the Data Viewer user manual. Here in the example we provided a “CENTER” command so the message will require Data Viewer to draw the image in reference to the image center during visualization.

```
# We need a time offset to proper use a message to mark the onset of the
stimulus_display screen

time_offset = int(self.time() - var.time_stimulus_display)
exp.eyelink.sendMessage('%d stimulus_display' % time_offset)

# Send another message to let Data Viewer know where to load the background image
during
# visualization; this require a special "!V IMGLOAD CENTER" message
img_path = '../images/' + var.image
img_x = var.width/2
img_y = var.height/2
imgload_msg = '%d !V IMGLOAD CENTER %s %d %d'%(time_offset, img_path, img_x, img_y)
exp.eyelink.sendMessage(imgload_msg)
```

There are two additional things worth mentioning here.

- It is possible to add a time offset to a message so the timestamp of the messages mark the actual trial events. For instance, if you have a picture that will be presented for 1-sec, sending the message before the image sketchpad may give you a screen refresh error. Instead, you can send the message following the image presentation, by including a time offset (like shown below) in the message.

- ghfx The EDF data files won't contain any background image for you. Instead, the EDF files store a "path" to the images, so Data Viewer can find the images when loading the EDF data files. Note that the "path" to the image file is relative to where you store your EDF data file. In the "picture" example, the EDF data files are stored in the "edf_file" folder, so we need ".." to go up one level to get to the "images" folder.

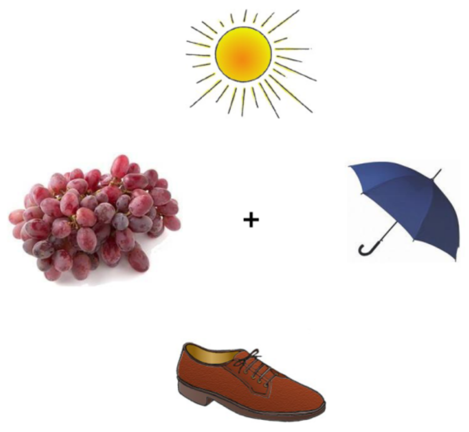
# 5 Visual World example

This example script shows how to program a Visual World type task in OpenSesame. One may believe that a tool with a nice graphical user interface will require minimum scripting, this is almost always NOT the case. OpenSesame requires quite a bit of "inline" scripting for tasks of medium to high complexity. The scripting part is discussed in more detail in a later section.

## 5.1 Overview of the task

The task is straightforward. A drift-correction/check target (cross) first appears on the screen, then 4 objects appear. After a preview period of 1000 msec, an audio file starts to playback. The subject's task is to quickly move the mouse cursor to click the target object (grapes in the present task) when they hear the target word ("grapes"). By performing a time-binning analysis over the sample gaze data, one can plot a nice figure to show the "decision-making" process after the target word has been played. For an overview of the Visual World paradigm, please see the review paper by Huettig, Rommers, & Meyer (2011).

*Huettig F, Rommers, J, & Meyer, A. S. (2011). Using the visual world paradigm to study language processing: A review and critical evaluation. psychologica 137(2):151-171. DOI: 10.1016/j.actpsy.2010.11.003*



This task obviously requires precise timing for audio playback. Please bear in mind that SR Research has not done any sort of timing verification for OpenSesame and we encourage users to perform their own tests if stimulus timing is critical for their tasks.

The overall organization of the script is not complicated, for introductory tutorials on programming in OpenSesame, please check the Tutorial section on the OpenSesame website, [http://osdoc.cogsci.nl](http://osdoc.cogsci.nl).

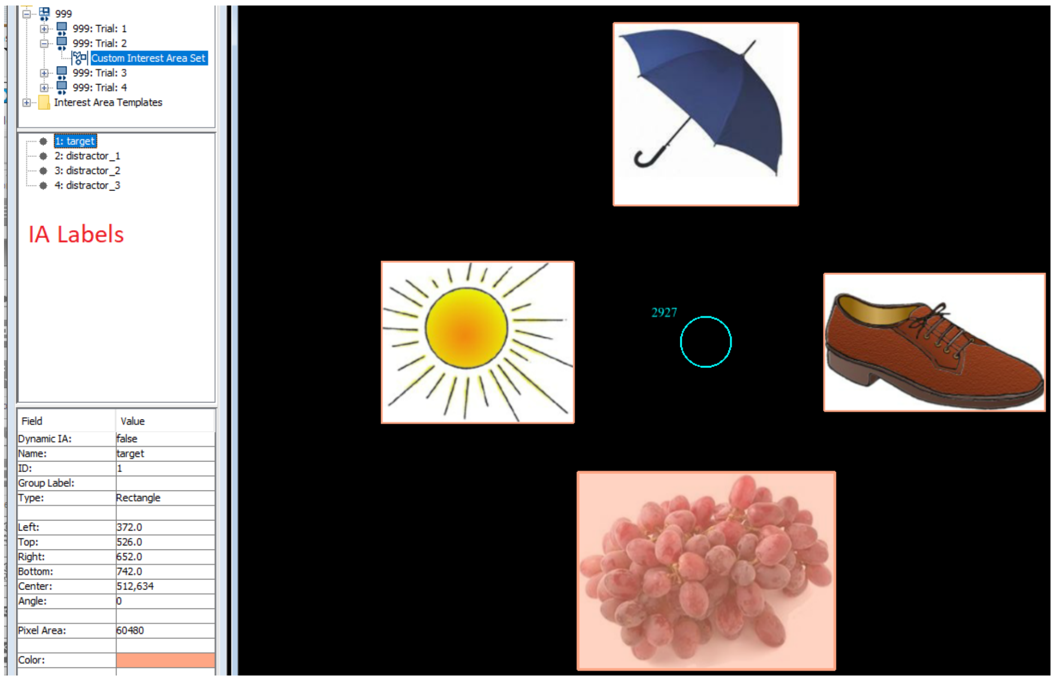
## 5.2 Interest Areas and Host PC landmarks

The present task uses a few inline scripts and the most complicated one is the “sti_preparation” inline. This script is responsible for quite some tasks we would recommend if one plans to use (or may use) the EyeLink Data Viewer software to analyze and visualize the eye movement data later on.

Here we first create a list of four screen locations with the xy_circle() function. Then, we change the position of the objects based on the location specified in the “block” loop. The “location” of each object is specified in the block loop as integers, i.e., 1, 2, 3, 4. We use this variable as an index to set the position of each object by using the list of screen coordinates (sti_pos) we created with xy_circle().

```
# the positions of the objects
pos = xy_circle(n=4, rho = 250)

# get the canvas on which the objects are shown
sti = items['stimulus_display'].canvas

# clear the Host screen
exp.eyelink.sendCommand('clear_screen 0')

# Clear the screen in Data Viewer
exp.eyelink.sendMessage('!V CLEAR_SCREEN 128 128 128')

# width and height of the images we use
img_w, img_h =[240, 240]

# position the images (tar, dis_1, dis_2, dis_3)
pos_index = [var.target_loc-1, var.distractor_1_loc-1, var.distractor_2_loc-1,
var.distractor_3_loc-1]
# image labels on the sketchpad
img_labels = ['tar', 'dis_1', 'dis_2', 'dis_3']
# name of the files
img_files = [var.target_img, var.distractor_1_img, var.distractor_2_img,
var.distractor_3_img]

for i in range(4):
    s = img_labels[i]
# set image position on the canvas
    sti[s].x, sti[s].y = pos[pos_index[i]]

# image position in the typical coordinates in computer graphics
    ia_left = int(sti[s].x - img_w/2 + var.width/2)
    ia_top = int(sti[s].y - img_h/2 + var.height/2)
    ia_right = int(sti[s].x + img_w/2 + var.width/2)
    ia_bottom = int(sti[s].y + img_h/2 + var.height/2)

# send Interest Area messages, IA labels should start with 1
    ia_msg = '!V IAREA RECTANGLE %d %d %d %d %d %s' %(i+1, ia_left, ia_top,
ia_right, ia_bottom, s)
    exp.eyelink.sendMessage(ia_msg)

# draw landmarks on the the PC screen
    landmark_msg = 'draw_box %d %d %d %d 15' % (ia_left, ia_top, ia_right,
ia_bottom)
    exp.eyelink.sendCommand(landmark_msg)
# label the landmarks with tar, dis_1, dis_2, dis_3
    exp.eyelink.sendCommand('draw_text %d %d 15 %s'%(ia_left+120, ia_top+120, s))

# IMGload commands for drawing pictures in Data viewer
    img_path = '../images/' + img_files[i]
    img_x = sti[s].x+var.width/2
    img_y = sti[s].y+var.height/2
    imgload_msg = '!V IMGLOAD CENTER %s %d %d'%(img_path, img_x, img_y)
    exp.eyelink.sendMessage(imgload_msg)
```

Then, in a for-loop, we send interest area messages to the tracker, draw the backdrop image on the Host, and record messages that will be used by Data Viewer to load the background images. It is important to bear in mind that the screen coordinates used by the tracker have the origin (0,0) at the top-left corner of the screen, whereas that in OpenSesame by default is the center of the screen. Transformation of the coordinates is needed here. In the for-loop, we first get the left, top, right, and bottom of each of the images. Then, use this information to construct interest area messages, and also the landmark drawing commands. The interest areas are all rectangular ones, for other types ofInterest Areas, please see the Data Viewer User Manual. With these messages, the Interest area definitions will be automatically loaded into Data Viewer when an EDF data file is opened.



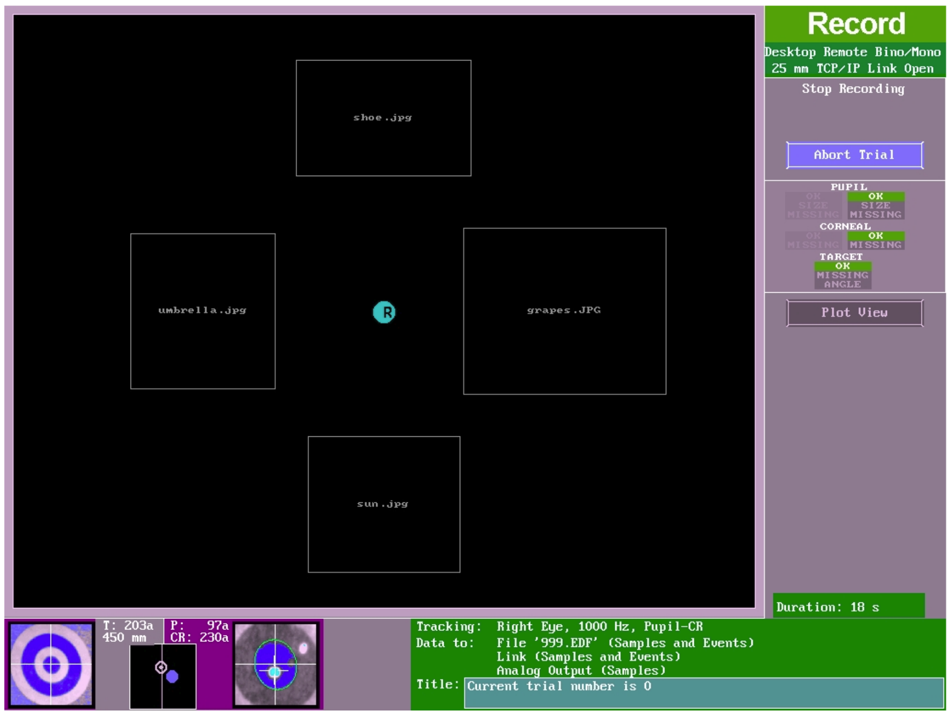
999
999: Trial: 1
999: Trial: 2
Custom Interest Area Set
999: Trial: 3
999: Trial: 4
Interest Area Templates

1: target
2: distractor_1
3: distractor_2
4: distractor_3

| Field | Value |
|---|---|
| Dynamic IA: | false |
| Name: | target |
| ID: | 1 |
| Group Label: | |
| Type: | Rectangle |
| | |
| Left: | 372.0 |
| Top: | 526.0 |
| Right: | 652.0 |
| Bottom: | 742.0 |
| Center: | 512,634 |
| Angle: | 0 |
| | |
| Pixel Area: | 60480 |
| | |
| Color: | 🟧 |

As noted above, one can transfer the images being presented to the subject to the Host PC screen, so the experimenter can examine which object is being fixated during recording. However, transferring images to the Host is not advisable in cases where a brief intertrial interval is required. Instead, one can always use the drawing commands to draw some landmarks on the Host PC. This can be done with the following line of code in the Run tab. Here we also used the “draw_text” command to show which image is presented in each landmark box on the Host PC.

```
# draw landmarks on the the PC screen
    landmark_msg = 'draw_box %d %d %d %d 15' % (ia_left, ia_top, ia_right,
ia_bottom)
    exp.eyelink.sendCommand(landmark_msg)
# label the landmarks with tar, dis_1, dis_2, dis_3
    exp.eyelink.sendCommand('draw_text %d %d 15 %s'%(ia_left+120, ia_top+120, s))
```

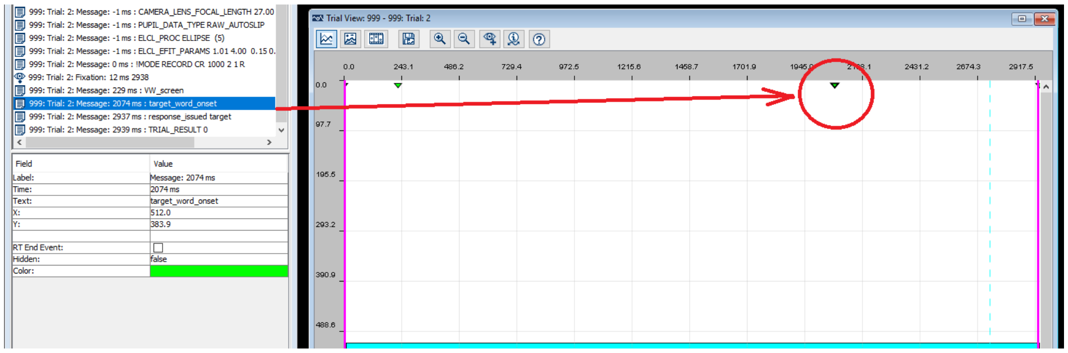


```
Tracking:  Right Eye, 1000 Hz, Pupil-CR
Data to:   File '999.EDF' (Samples and Events)
           Link (Samples and Events)
           Analog Output (Samples)
Title:     Current trial number is 0
```

## 5.2 Message to mark trial events

In the “MSG_target_word” inline script, we simply wait for a while until the target word is played in the audio file. Then we send a message to the tracker to mark the onset of the target word (“grapes”). This message is critical to our data analysis, without this message, it would be challenging to align the eye movement data to the onset of the target word.

```
# wait for the onset of the target word, then should the mouse cursor
clock.sleep(var.target_word_onset_time)

# send a message to let the tracker know the onset of the target word
exp.eyelink.sendMessage('target_word_onset')
```

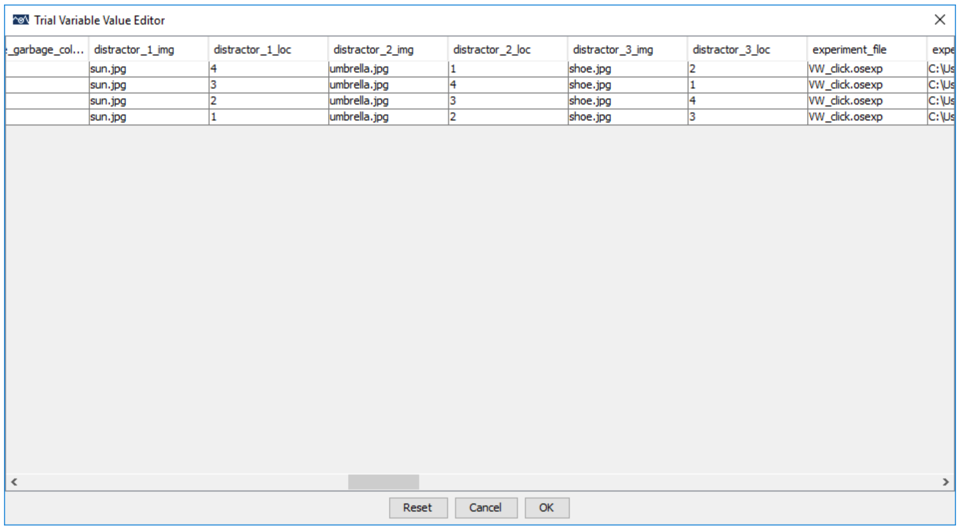


```
999: Trial: 2: Message: -1 ms : CAMERA_LENS_FOCAL_LENGTH 27.00
999: Trial: 2: Message: -1 ms : PUPIL_DATA_TYPE RAW_AUTOSLIP
999: Trial: 2: Message: -1 ms : ELCL_PROC ELLIPSE  (5)
999: Trial: 2: Message: -1 ms : ELCL_EFIT_PARAMS 1.01 4.00  0.15 0.
999: Trial: 2: Message: 0 ms : !MODE RECORD CR 1000 2 1 R
999: Trial: 2: Fixation: 12 ms 2938
999: Trial: 2: Message: 229 ms : VW_screen
999: Trial: 2: Message: 2074 ms : target_word_onset
999: Trial: 2: Message: 2937 ms : response_issued target
999: Trial: 2: Message: 2939 ms : TRIAL_RESULT 0
```

| Field | Value |
|---|---|
| Label: | Message: 2074 ms |
| Time: | 2074 ms |
| Text: | target_word_onset |
| X: | 512.0 |
| Y: | 383.9 |
| | |
| RT End Event: | ☐ |
| Hidden: | false |
| Color: | 🟩 |

## 5.3 Trial variables

At the end of each trial, when the *el_StopRecording* node is called, all variables in the OpenSesame namespace will be recorded in the EDF data file. These variables are critical for group-level analysis in Data Viewer and can be accessed from the Trial Variable Value Editor within Data Viewer.



| _garbage_col... | distractor_1_img | distractor_1_loc | distractor_2_img | distractor_2_loc | distractor_3_img | distractor_3_loc | experiment_file | expe |
|---|---|---|---|---|---|---|---|---|
| | sun.jpg | 4 | umbrella.jpg | 1 | shoe.jpg | 2 | VW_click.osexp | C:\Us |
| | sun.jpg | 3 | umbrella.jpg | 4 | shoe.jpg | 1 | VW_click.osexp | C:\Us |
| | sun.jpg | 2 | umbrella.jpg | 3 | shoe.jpg | 4 | VW_click.osexp | C:\Us |
| | sun.jpg | 1 | umbrella.jpg | 2 | shoe.jpg | 3 | VW_click.osexp | C:\Us |

# 6 Known issues and limitations

Please bear in mind that the EyeLink plugin is still beta software and we have noticed the following issues during our testing.

- The .set_pos() function only works when the backend is Psychopy. So, in the Visual World task, the mouse cursor won't necessarily be placed at the center of the screen after the target word has been played.
- Animated calibration targets can be useful for infant studies. The current implementation of animated calibration targets (videos) presents the frames of the video as images one-by-one, thus no sound is played during the calibration.

If you run into any issues when using the EyeLink plugin, please feel free to contact support@sr-research.com.