

Getting Started with Python, PyLink, and Pygame

Version: 2.2 • **Last Updated:** 24/04/2026

This guide covers the core principles for integrating SR Research EyeLink eye trackers with Python, both with Pygame and without a dedicated media module. You can build experimental tasks using a multimedia library like Pygame, or run pure Python scripts if your task does not require visual or auditory stimuli.

The **PyLink** (`sr-research-pylink`) library and pygame `pygame-eyelink-coregraphics` dependency are available as standalone PyPI packages. Once they are installed, you do not need the EyeLink Developers Kit to run the examples.

To read the comprehensive documentation for PyLink, see the [PyLink API User Guide](#). It is included with the EyeLink Developers Kit installation in the following locations:

- **Windows:** `C:\Program Files (x86)\SR Research\EyeLink\SampleExperiments\Python\`
(Available as `pylink api userguide.pdf` and `pylink.chm`)
- **macOS:** `/Applications/Eyelink/SampleExperiments/Python/pylink api userguide.pdf`
- **Linux:** `/usr/share/EyeLink/SampleExperiments/Python/pylink api userguide.pdf`

If you are using Python-based experiment builders like PsychoPy or OpenSesame, please note that while they also rely on PyLink (`sr-research-pylink`), they utilize their own specific coregraphics implementations. Because of this, alongside their unique interfaces and code structures, they have their own specific setup and integration steps. You can find the dedicated documentation for these platforms here:

- [Getting Started with PsychoPy / EyeLink Integration](#)
- [Getting Started with OpenSesame](#)

1. Setup & Installation

Before running the EyeLink examples, set up your Display PC's network connection to communicate with the EyeLink Host PC. To set up your connection refer to: [How do I configure my network settings to connect to the EyeLink Host PC?](#).

To install the necessary dependencies for the Pygame examples, open a terminal or command prompt and run:

```
pip install pygame sr-research-pylink pygame-eyelink-coregraphics
```

Alternatively, if you are working from a provided `pygame_examples` directory, run:

```
pip install -r requirements.txt
```

Important: For offline installation steps see section **4.1 Offline Installation**.

2. Example Scripts Overview

A. Scripts Without a Multimedia Library

Two example Python scripts do not require additional Python multimedia libraries (like Pygame). These are useful examples for backend data retrieval or non-visual tasks.

These tasks only require PyLink.

```
pip install sr-research-pylink
```

Template	Description
linkEvent	This script demonstrates the frequently used commands for connecting to the tracker, configuring tracker parameters, starting/ending recording, and messaging for event logging. Most importantly, this script shows how to retrieve eye events (Fixation Start / End, Saccade Start / End, etc.) during data recording from the stimulus presentation PC.
linkSample	This script demonstrates the frequently used commands for connecting to the tracker, configuring tracker parameters, starting/ending recording, and messaging for event logging. Most importantly, this script shows how to retrieve samples (time stamped gaze position, pupil size, etc.) in real-time during data recording.

B. Example Scripts Descriptions Using Pygame

The following templates demonstrate standard integration methods for scripts using Pygame:

Template	Description
picture	This example shows how to connect to and disconnect from the tracker, how to open and close data files, how to start / stop recording, and the standard messages for integration with the Data Viewer software. We show four pictures one-by-one on each trial, and a trial terminates upon a keypress response or until 3 secs have elapsed.
pursuit	This example shows how to record the target position in a smooth pursuit task. This script also shows how to record dynamic interest area and target position information to the EDF data file so Data Viewer can recreate the interest area and play back the target movement.
saccade	This example shows how to retrieve eye events (saccades) during testing. A visual target appears on the left side, or right side of the screen and the participant is required to quickly shift gaze to look at the target (pro-saccade) or a mirror location on the opposite side of the central fixation (anti-saccade).
fixationWindow_fastSamples	This example shows how to implement a gaze-based trigger. First a fixation cross is shown at the center of the screen. The trial moves on only when gaze has been directed to the fixation cross.
GC_window	This script shows how to manipulate visual stimuli based on real-time gaze data. A mask is shown at the current gaze position in the "mask" condition; in the "window" condition, the image is masked, and a window at the current gaze position will reveal the image hidden behind the mask.
MRI_demo	This example shows how to implement continuous eye tracker recording through a block of trials (e.g., in an MRI setup), and how to synchronize the presentation of trials with a sync signal from the MRI. With a long recording, we start and stop recording at the beginning and end of a testing session (run), rather than at the beginning and end of each experimental trial. We still send the TRIALID and TRIAL_RESULT messages to the tracker, and Data Viewer will still be able to segment the long recording into small segments (trials).

3. Code Walkthrough

Each example script is heavily commented and serves as a great starting point. Here we will walk through the `picture.py` script from the `pygame_examples` to illustrate the PyLink library and steps for Data Viewer integration.

3.1 Set up an EDF data file name

At the beginning of the script a prompt requires an EDF data filename. The filename should not exceed 8 characters and should only contain letters, numbers, and underscores (_). The script also sets up local folders to store the data file and the resources (e.g., images) we used in the experiment.

```
# Set up EDF data file name and local data folder
#
# The EDF data filename should not exceed eight alphanumeric characters
# use ONLY number 0-9, letters, and _ (underscore) in the filename
edf_fname = 'TEST'

# Prompt user to specify an EDF data filename
# before we open a fullscreen window
while True:
    # use "raw_input" to get user input if running with Python 2.x
    try:
        input = raw_input
    except NameError:
        pass
    prompt = '\nSpecify an EDF filename\n' + \
        'Filename must not exceed eight alphanumeric characters.\n' + \
        'ONLY letters, numbers and underscore are allowed.\n\n--> '
    edf_fname = input(prompt)
    # strip trailing characters, ignore the '.edf' extension
    edf_fname = edf_fname.rstrip().split('.')[0]

    # check if the filename is valid (length <= 8 & no special char)
    allowed_char = ascii_letters + digits + '_'
    if not all([c in allowed_char for c in edf_fname]):
        print('ERROR: Invalid EDF filename')
    elif len(edf_fname) > 8:
        print('ERROR: EDF filename should not exceed 8 characters')
    else:
        break

# Set up a folder to store the EDF data files and the associated resources
# e.g., files defining the interest areas used in each trial
results_folder = 'results'
if not os.path.exists(results_folder):
    os.makedirs(results_folder)

# We download EDF data file from the EyeLink Host PC to the local hard
# drive at the end of each testing session, here we rename the EDF to
# include session start date/time
```

```

time_str = time.strftime("_%Y_%m_%d_%H_%M", time.localtime())
session_identifier = edf_fname + time_str

# create a folder for the current testing session in the "results" folder
session_folder = os.path.join(results_folder, session_identifier)
if not os.path.exists(session_folder):
    os.makedirs(session_folder)

```

3.2 Connect to the tracker

The Display PC connects to the EyeLink Host PC over a local network connection at the beginning of an eye-tracking experiment. The command for initializing a connection is `pylink.EyeLink()`. This command takes just one parameter, the IP address of the Host PC. If you omit the IP address, this method will use the default address of the EyeLink Host PC, which is 100.1.1.1.

The command `pylink.EyeLink()` returns an EyeLink object (i.e., `el_tracker` in the example code below), which has a set of methods that we can use to control the EyeLink tracker. Once there is an active connection, the EyeLink Host PC status will switch to "TCP / IP Link Open" (shown in the top-right corner of the Host PC screen).

```

# Step 1: Connect to the EyeLink Host PC
#
# The Host IP address, by default, is "100.1.1.1".
# the "el_tracker" objected created here can be accessed through the Pylink
# Set the Host PC address to "None" (without quotes) to run the script
# in "Dummy Mode"
if dummy_mode:
    el_tracker = pylink.EyeLink(None)
else:
    try:
        el_tracker = pylink.EyeLink("100.1.1.1")
    except RuntimeError as error:
        print('ERROR:', error)
        pygame.quit()
        sys.exit()

```

If access to an EyeLink eye tracker is not available or you want to debug your experimental script on a computer that is not connected, you can toggle the `dummy_mode` variable to `True` to open a simulated connection to the tracker.

3.3 Open an EDF data file

At the beginning of a new eye-tracking session, first open a data file on the Host PC to store the eye movement data. This file is retrieved from the Host PC at the end of the session after it is closed. For backward compatibility, the EDF file name should not exceed eight characters (not including the .edf extension). To open an EDF data file on the Host PC, use the `openDataFile()` command.

```

# Step 2: Open an EDF data file on the Host PC
edf_file = edf_fname + ".EDF"
try:
    el_tracker.openDataFile(edf_file)
except RuntimeError as err:
    print('ERROR:', err)
    # close the link if we have one open
    if el_tracker.isConnected():
        el_tracker.close()
    pygame.quit()
    sys.exit()

```

Including header information in the EDF data file is good practice. Without it, it can be difficult to tell which EDF data file belongs to which research project.

```

# Add a header text to the EDF file to identify the current experiment name
# This is OPTIONAL. If your text starts with "RECORDED BY " it will be
# available in DataViewer's Inspector window by clicking
# the EDF session node in the top panel and looking for the "Recorded By:"
# field in the bottom panel of the Inspector.
preamble_text = 'RECORDED BY %s' % os.path.basename(__file__)
el_tracker.sendCommand("add_file_preamble_text '%s'" % preamble_text)

```

3.4 Configure the tracker

A consistent approach to configure tracker parameters (e.g. the calibration type) is to send commands to the Host PC with the `sendCommand()` method at the onset of the script.

```

# Step 3: Configure the tracker
#
# Put the tracker in offline mode before we change tracking parameters
el_tracker.setOfflineMode()

# Get the software version: 1-EyeLink I, 2-EyeLink II, 3/4-EyeLink 1000,
# 5-EyeLink 1000 Plus, 6-Portable DUO
eyelink_ver = 0 # set version to 0, in case running in Dummy mode
if not dummy_mode:
    vstr = el_tracker.getTrackerVersionString()
    eyelink_ver = int(vstr.split()[-1].split('.')[0])
    # print out some version info in the shell
    print('Running experiment on %s, version %d' % (vstr, eyelink_ver))

# File and Link data control
# what eye events to save in the EDF file, include everything by default
file_event_flags = 'LEFT,RIGHT,FIXATION,SACCADE,BLINK,MESSAGE,BUTTON,INPUT'
# what eye events to make available over the link, include everything by default
link_event_flags = 'LEFT,RIGHT,FIXATION,SACCADE,BLINK,BUTTON,FIXUPDATE,INPUT'
# what sample data to save in the EDF data file and to make available

```

```

# over the link, include the 'HTARGET' flag to save head target sticker
# data for supported eye trackers
if eyelink_ver > 3:
    file_sample_flags =
'LEFT,RIGHT,GAZE,HREF,RAW,AREA,HTARGET,GAZERES,BUTTON,STATUS,INPUT'
    link_sample_flags = 'LEFT,RIGHT,GAZE,GAZERES,AREA,HTARGET,STATUS,INPUT'
else:
    file_sample_flags =
'LEFT,RIGHT,GAZE,HREF,RAW,AREA,GAZERES,BUTTON,STATUS,INPUT'
    link_sample_flags = 'LEFT,RIGHT,GAZE,GAZERES,AREA,STATUS,INPUT'
el_tracker.sendCommand("file_event_filter = %s" % file_event_flags)
el_tracker.sendCommand("file_sample_data = %s" % file_sample_flags)
el_tracker.sendCommand("link_event_filter = %s" % link_event_flags)
el_tracker.sendCommand("link_sample_data = %s" % link_sample_flags)

# Optional tracking parameters
# Sample rate, 250, 500, 1000, or 2000, check your tracker specification
# if eyelink_ver > 2:
#     el_tracker.sendCommand("sample_rate 1000")
# Choose a calibration type, H3, HV3, HV5, HV13 (HV = horizontal/vertical),
el_tracker.sendCommand("calibration_type = HV9")
# Set a gamepad button to accept calibration/drift check target
# You need a supported gamepad/button box that is connected to the Host PC
el_tracker.sendCommand("button_function 5 'accept_target_fixation'")

```

3.5 Open a window

To present the visual stimuli and to calibrate the tracker you need to open a graphics window. Send the correct screen resolution to the Host PC with the `screen_pixel_coords` command and log this info in the EDF data file with a `DISPLAY_COORDS` message. The `DISPLAY_COORDS` message will ensure the Trial View window is sized appropriately in Data Viewer.

```

# Step 4: set up a graphics environment for calibration
#
# open a Pygame window
win=None
if full_screen:
    win = pygame.display.set_mode((0, 0), FULLSCREEN | DOUBLEBUF, vsync = 1)
else:
    win = pygame.display.set_mode((0, 0), 0, vsync = 1)

# Windows focus fix for pygame 2
if os.name == 'nt' and pygame.__version__.split('.')[0] == '2':
    hwnd = pygame.display.get_wm_info()["window"]
    ctypes.windll.user32.SetForegroundWindow(hwnd)

scn_width, scn_height = win.get_size()
pygame.mouse.set_visible(False) # hide mouse cursor

# Pass the display pixel coordinates (left, top, right, bottom) to the tracker
# see the EyeLink Installation Guide, "Customizing Screen Settings"

```

```

el_coords = "screen_pixel_coords = 0 0 %d %d" % (scn_width - 1, scn_height - 1)
el_tracker.sendCommand(el_coords)

# Write a DISPLAY_COORDS message to the EDF file
# Data Viewer needs this piece of info for proper visualization, see Data
# Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
dv_coords = "DISPLAY_COORDS 0 0 %d %d" % (scn_width - 1, scn_height - 1)
el_tracker.sendMessage(dv_coords)

```

3.6 Calibration graphics library

With the EyeLink tracker, calibration is a two-step process: calibrating the tracker and evaluating the calibration results at validation. Both steps involve presenting a series of visual targets at different known screen positions. The participant shifts their gaze to each calibration target location.

The calibration graphics window must be configured before we call this command. To use this library, we need to first import the `CalibrationGraphics` class from the installed `pygame_eyelink_coregraphics` module. We then configure its various parameters, e.g., foreground/background color, calibration target, and audio_mode. Note that the calibration target could be a 'circle' (default) or a 'picture'. The calibration audio_mode can be either 'default' (standard beeps), 'none' (no sound), or 'custom' (specify the three .wav files).

```

# Configure a graphics environment (genv) for tracker calibration
# The new audio_mode parameter accepts 'default', 'none', or 'custom'.
genv = CalibrationGraphics(el_tracker, win, audio_mode='default')

# Set background and foreground colors
# parameters: foreground_color, background_color
foreground_color = (0, 0, 0)
background_color = (128, 128, 128)
genv.setCalibrationColors(foreground_color, background_color)

# Set up the calibration target
#
# The target could be a "circle" (default) or a "picture",
# To configure the type of calibration target, set
# genv.setTargetType to "circle", "picture", e.g.,
# genv.setTargetType('picture')
#
# Use genv.setPictureTarget() to set a "picture" target, e.g.,
# genv.setPictureTarget(os.path.join('images', 'fixTarget.bmp'))

# Use a picture as the calibration target
genv.setTargetType('picture')
genv.setPictureTarget(os.path.join('images', 'fixTarget.bmp'))

# Configure the size of the calibration target (in pixels)
# genv.setTargetSize(24)

# Beeps to play during calibration, validation and drift correction

```



```
#
# The audio_mode parameter in CalibrationGraphics() supports 3 modes:
# 'default' - Uses the bundled default tones (target, good, error)
# 'none'     - Disables all calibration and validation beeps
# 'custom'   - Allows you to use your own sound files
#
# If you initialized the class with audio_mode='custom', you MUST call
# setCalibrationSounds() to explicitly set the paths to your .wav files.
# Example:
# genv.setCalibrationSounds('my_target.wav', 'my_good.wav', 'my_error.wav')

# Request Pylink to use the Pygame window we opened above for calibration
pylink.openGraphicsEx(genv)
```

3.7 Helper functions

Helpful functions are defined to clear the screen, terminate the task, register keypresses, send messages, and abort a trial.

```
# Step 5: Run the experimental trials
# define a few helper functions for trial handling

def show_message(message, fg_color, bg_color):
    """ show messages on the screen

    message: The message you would like to show on the screen
    fg_color/bg_color: color for the texts and the background screen
    """

    # clear the screen and blit the texts
    win_surf = win
    win_surf.fill(bg_color)

    scn_w, scn_h = win_surf.get_size()
    message_fnt = pygame.font.SysFont('Arial', 32)
    msgs = message.split('\n')
    for i in range(len(msgs)):
        message_surf = message_fnt.render(msgs[i], True, fg_color)
        w, h = message_surf.get_size()
        msg_y = scn_h / 2 + h / 2 * 2.5 * (i - len(msgs) / 2.0)
        win_surf.blit(message_surf, (int(scn_w / 2 - w / 2), int(msg_y)))

    pygame.display.flip()

def wait_key(key_list, duration=sys.maxsize):
    """ detect and return a keypress, terminate the task if ESCAPE is pressed

    parameters:
    key_list: allowable keys (pygame key constants, e.g., [K_a, K_ESCAPE])
```

```

duration: the maximum time allowed to issue a response (in ms)
        wait for response 'indefinitely' (with sys.maxsize)
"""

got_key = False
# clear all cached events if there are any
pygame.event.clear()
t_start = pygame.time.get_ticks()
resp = [None, t_start, -1]

while not got_key:
    # check for time out
    if (pygame.time.get_ticks() - t_start) > duration:
        break

    # check key presses
    for ev in pygame.event.get():
        if ev.type == KEYDOWN:
            if ev.key in key_list:
                resp = [pygame.key.name(ev.key),
                        t_start,
                        pygame.time.get_ticks()]
                got_key = True

            if (ev.type == KEYDOWN) and (ev.key == K_c):
                if ev.mod in [KMOD_LCTRL, KMOD_RCTRL, 4160, 4224]:
                    terminate_task()

# clear the screen following each keyboard response
win_surf = win
win_surf.fill(genv.getBackgroundColor())
pygame.display.flip()

return resp

def abort_trial():
    """Ends recording

    We add 100 msec to catch final events
    """

    # Stop recording
    if el_tracker.isRecording():
        # add 100 ms to catch final trial events
        pylink.pumpDelay(100)
        el_tracker.stopRecording()

    # clear the screen
    surf = win
    surf.fill((128, 128, 128))
    pygame.display.flip()
    # Send a message to clear the Data Viewer screen
    el_tracker.sendMessage('!V CLEAR 128 128 128')

```

```
# send a message to mark trial end
el_tracker.sendMessage('TRIAL_RESULT %d' % pylink.TRIAL_ERROR)

return pylink.TRIAL_ERROR
```

3.8 Calibrate the tracker

To calibrate the eye tracker, call the `doTrackerSetup()` command. Calibration is performed at the beginning of a testing session and whenever the tracking accuracy declines.

```
# Step 5: Set up the camera and calibrate the tracker

# Show the task instructions
task_msg = 'In the task, you may press the SPACEBAR to end a trial\n' + \
    '\nPress Ctrl-C to if you need to quit the task early\n'
if dummy_mode:
    task_msg = task_msg + '\nNow, press ENTER to start the task'
else:
    task_msg = task_msg + '\nNow, press ENTER to calibrate tracker'

show_message(task_msg, (0, 0, 0), (128, 128, 128))
wait_key([K_RETURN])

# skip this step if running the script in Dummy Mode
if not dummy_mode:
    try:
        el_tracker.doTrackerSetup()
    except RuntimeError as err:
        print('ERROR:', err)
        el_tracker.exitCalibration()
```

3.9 Run through all trials

The parameters of each trial are put into a list at the beginning of the script. Following calibration, a loop iterates through the list.

```
# Step 6: Run the experimental trials, index all the trials

# construct a list of 4 trials
test_list = trials[:]*2

# randomize the trial list
random.shuffle(test_list)

trial_index = 1
for trial_pars in test_list:
```

```
run_trial(trial_pars, trial_index)
trial_index += 1
```

3.10 The `run_trial()` function

In the script, we defined a `run_trial()` function to group the commands we need to execute on each trial.

3.10.1 Backdrop on the Host

Commands are sent to clear the Host PC screen and then draw the new trial's backdrop or landmarks. This is optional, but can be a helpful feature if you would like to monitor gaze during testing overlaid on an image or landmark. Drawing a backdrop on the Host PC screen is illustrated with two different commands, `imageBackdrop()` and `bitmapBackdrop()`.

```
def run_trial(trial_pars, trial_index):
    """ Helper function specifying the events that will occur in a single trial

    trial_pars - a list containing trial parameters, e.g.,
                  ['cond_1', 'img_1.jpg']
    trial_index - record the order of trial presentation in the task
    """

    # unpacking the trial parameters
    cond, pic = trial_pars

    # load the image to display, here we stretch the image to fill full screen
    img = pygame.image.load('./images/' + pic)
    img = pygame.transform.scale(img, (scn_width, scn_height))

    # get the currently active window
    surf = win

    # put the tracker in the offline mode first, el_tracker is within scope
    el_tracker.setOfflineMode()

    # clear the host screen before we draw the backdrop
    el_tracker.sendCommand('clear_screen 0')

    # 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
    el_tracker.imageBackdrop(os.path.join('images', pic),
                             0, 0, scn_width, scn_height, 0, 0,
                             pylink.BX_MAXCONTRAST)

    # If you need to scale the backdrop image on the Host, use the old Pylink
    # bitmapBackdrop(), which requires an additional step of converting the
    # image pixels into a recognizable format by the Host PC.
    # pixels = [line1, ...lineH], line = [pix1,...pixW], pix=(R,G,B)
    #
```

```

# the bitmapBackdrop() command takes time to return, not recommended
# for tasks where the ITI matters, e.g., in an event-related fMRI task
# parameters: width, height, pixel, crop_x, crop_y,
#             crop_width, crop_height, x, y on the Host, drawing options
#
# Use the code commented below to convert the image and send the backdrop
#
# pixels = [[img.get_at((i, j))[0:3] for i in range(scn_width)]
#           for j in range(scn_height)]
# el_tracker.bitmapBackdrop(scn_width, scn_height, pixels,
#                           0, 0, scn_width, scn_height,
#                           0, 0, pylink.BX_MAXCONTRAST)

# OPTIONAL: draw landmarks on the Host screen
# In addition to backdrop image, You may draw simples on the Host PC to use
# as landmarks. For illustration purpose, here we draw some texts and a box
# For a list of supported draw commands, see the "COMMANDS.INI" file on the
# Host PC (under /elcl/exe)
left = int(scn_width/2.0) - 60
top = int(scn_height/2.0) - 60
right = int(scn_width/2.0) + 60
bottom = int(scn_height/2.0) + 60
draw_cmd = 'draw_filled_box %d %d %d %d 1' % (left, top, right, bottom)
el_tracker.sendCommand(draw_cmd)

```

3.10.2 TRIALID message

A **TRIALID** message marks the onset of a new trial. This message is required for Data Viewer to correctly segment the data file into trials.

```

# send a "TRIALID" message to mark the start of a trial, see Data
# Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
el_tracker.sendMessage('TRIALID %d' % trial_index)

```

3.10.3 Record status message

Informative text about the current trial can be shown on the Host PC as a Recording Status Message. For example, the current trial number.

```

# record_status_message : show some info on the Host PC
# here we show how many trial has been tested
status_msg = 'TRIAL number %d' % trial_index
el_tracker.sendCommand("record_status_message '%s'" % status_msg)

```

3.10.4 Drift-check / drift-correction

An essential command in this example is `doDriftCorrect()`. This command is executed before recording gaze and presents a single target on the screen. The experimenter can confirm the gaze position by pressing the spacebar or Enter keys. This feature checks the tracking accuracy and allows users to recalibrate if necessary, by pressing Esc.

The `doDriftCorrect()` command takes four parameters. The first two are x, y pixel coordinates for the drift-check target (integers). The third parameter specifies whether PyLink should draw the target. The fourth parameter controls whether PyLink should exit to Camera Setup if the ESCAPE key is pressed. This is the best option to allow a new calibration to be completed mid-task.

```
# drift check
# we recommend drift-check at the beginning of each trial
# the doDriftCorrect() function requires target position in integers
# the last two arguments:
# draw_target (1-default, 0-draw the target then call doDriftCorrect)
# allow_setup (1-press ESCAPE to recalibrate, 0-not allowed)
#
# Skip drift-check if running the script in Dummy Mode
while not dummy_mode:
    # terminate the task if no longer connected to the tracker or
    # user pressed Ctrl-C to terminate the task
    if (not el_tracker.isConnected()) or el_tracker.breakPressed():
        terminate_task()
        return pylink.ABORT_EXPT

    # drift-check and re-do camera setup if ESCAPE is pressed
    try:
        error = el_tracker.doDriftCorrect(int(scen_width/2.0),
                                          int(scen_height/2.0), 1, 1)

        # break following a success drift-check
        if error is not pylink.ESC_KEY:
            break
    except:
        pass
```

3.10.5 Data recording

To start data recording, call `startRecording()`. This command takes four parameters specifying what types of data (event or sample) are recorded in the EDF data file and what types of data are available over the link during testing.

```
# put tracker in idle/offline mode before recording
el_tracker.setOfflineMode()

# Start recording
# arguments: sample_to_file, events_to_file, sample_over_link,
# event_over_link (1=yes, 0=no)
```

```

try:
    el_tracker.startRecording(1, 1, 1, 1)
except RuntimeError as error:
    print("ERROR:", error)
    abort_trial()
    return pylink.TRIAL_ERROR

# Allocate some time for the tracker to cache some samples
pylink.pumpDelay(100)

```

3.10.6 Logging messages

Messages should be sent to the tracker whenever a critical event occurs; for instance, when a picture appears on the screen. The expected formatting and the options available for the functions performed by these special messages are described in the [EyeLink Data Viewer Integration Messaging Protocol](#).

```

# show the image
surf.fill((128, 128, 128)) # clear the screen
surf.blit(img, (0, 0))
pygame.display.flip()
onset_time = pygame.time.get_ticks() # image onset time

# send over a message to mark the onset of the image
el_tracker.sendMessage('image_onset')

# Send a message to clear the Data Viewer screen, get it ready for
# drawing the pictures during visualization
el_tracker.sendMessage('!V CLEAR 128 128 128')

# send over a message to specify where the image is stored relative
# to the EDF data file, see Data Viewer User Manual, "Protocol for
# EyeLink Data to Viewer Integration"
bg_image = '../..//images/' + pic
imgload_msg = '!V IMGLOAD CENTER %s %d %d %d %d' % (bg_image,
                                                    int(scn_width/2.0),
                                                    int(scn_height/2.0),
                                                    int(scn_width),
                                                    int(scn_height))

el_tracker.sendMessage(imgload_msg)

# send interest area messages to record in the EDF data file
# here we draw a rectangular IA, for illustration purposes
# format: !V IAREA RECTANGLE <id> <left> <top> <right> <bottom> [label]
# for all supported interest area commands, see the Data Viewer Manual,
# "Protocol for EyeLink Data to Viewer Integration"
ia_pars = (1, left, top, right, bottom, 'screen_center')
el_tracker.sendMessage('!V IAREA RECTANGLE %d %d %d %d %d %s' % ia_pars)

# show the image for 5 secs; break if the SPACEBAR is pressed
pygame.event.clear() # clear all cached events if there were any
get_keypress = False

```

```

RT = -1
while not get_keypress:
    # present the picture for a maximum of 5 seconds
    if pygame.time.get_ticks() - onset_time >= 5000:
        el_tracker.sendMessage('time_out')
        break

    # abort the current trial if the tracker is no longer recording
    error = el_tracker.isRecording()
    if error is not pylink.TRIAL_OK:
        el_tracker.sendMessage('tracker_disconnected')
        abort_trial()
        return error

    # check for keyboard events
    for ev in pygame.event.get():
        # Stop stimulus presentation when the spacebar is pressed
        if (ev.type == KEYDOWN) and (ev.key == K_SPACE):
            # send over a message to log the key press
            el_tracker.sendMessage('key_pressed')

            # get response time in ms, PsychoPy report time in sec
            RT = pygame.time.get_ticks() - onset_time
            get_keypress = True

        # Abort a trial if "ESCAPE" is pressed
        if (ev.type == KEYDOWN) and (ev.key == K_ESCAPE):
            el_tracker.sendMessage('trial_skipped_by_user')
            # clear the screen
            surf.fill((128, 128, 128))
            pygame.display.flip()
            # abort trial
            abort_trial()
            return pylink.SKIP_TRIAL

        # Terminate the task if Ctrl-c
        if (ev.type == KEYDOWN) and (ev.key == K_c):
            if ev.mod in [KMOD_LCTRL, KMOD_RCTRL, 4160, 4224]:
                el_tracker.sendMessage('terminated_by_user')
                terminate_task()
                return pylink.ABORT_EXPT

    # clear the screen
    surf.fill((128, 128, 128))
    pygame.display.flip()
    el_tracker.sendMessage('blank_screen')
    # send a message to clear the Data Viewer screen as well
    el_tracker.sendMessage('!V CLEAR 128 128 128')

    # record trial variables to the EDF data file, for details, see Data
    # Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
    el_tracker.sendMessage('!V TRIAL_VAR condition %s' % cond)

```



```
el_tracker.sendMessage('!V TRIAL_VAR image %s' % pic)
el_tracker.sendMessage('!V TRIAL_VAR RT %d' % RT)
```

3.10.7 The `TRIAL_RESULT` message

At the end of a trial send a `TRIAL_RESULT` message. A flag of "0" with the `TRIAL_RESULT` message indicates the trial completed successfully. The `TRIAL_RESULT` message should be sent following the `stopRecording()` command, so the recording is enclosed in a pair of `TRIALID` and `TRIAL_RESULT` messages, which Data Viewer uses to segment the trials.

```
# stop recording; add 100 msec to catch final events before stopping
pylink.pumpDelay(100)
el_tracker.stopRecording()

# send a 'TRIAL_RESULT' message to mark the end of trial, see Data
# Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
el_tracker.sendMessage('TRIAL_RESULT %d' % pylink.TRIAL_OK)
```

3.10.8 The `terminate_task()` function

The `terminate_task()` function performs housekeeping at the end of a session or if the task terminated prematurely. This function handles three main tasks: 1. Put the tracker in Offline mode, 2. Close the data file, and 3. Download the file to the stimulus presentation PC and terminate the link to the tracker. It is recommended to put the tracker in Offline mode and add a delay before closing the data file. This ensures each function runs smoothly.

```
def terminate_task():
    """ Terminate the task gracefully and retrieve the EDF data file

    file_to_retrieve: The EDF on the Host that we would like to download
    win: the current window used by the experimental script
    """

    if el_tracker.isConnected():
        # Terminate the current trial first if the task terminated prematurely
        error = el_tracker.isRecording()
        if error == pylink.TRIAL_OK:
            abort_trial()

        # Put tracker in Offline mode
        el_tracker.setOfflineMode()

        # Clear the Host PC screen and wait for 500 ms
        el_tracker.sendCommand('clear_screen 0')
        pylink.msecDelay(500)

        # Close the edf data file on the Host
        el_tracker.closeDataFile()
```

```

# Show a file transfer message on the screen
msg = 'EDF data is transferring from EyeLink Host PC...'
show_message(msg, (0, 0, 0), (128, 128, 128))

# Download the EDF data file from the Host PC to a local data folder
# parameters: source_file_on_the_host, destination_file_on_local_drive
local_edf = os.path.join(session_folder, session_identifier + '.EDF')
try:
    el_tracker.receiveDataFile( edf_file, local_edf, 6 )
except RuntimeError as error:
    print('ERROR:', error)

# Close the link to the tracker.
el_tracker.close()

# quit pygame and python
pygame.quit()
sys.exit()

# Step 7: disconnect, download the EDF file, then terminate the task
terminate_task()

```

4. Offline Installation and Support

4.1 Offline Installation

If your Display PC is not connected to the internet (which is common for dedicated experimental setups), you can still install the required dependencies using an offline method.

Important: The `sr-research-pylink` package is Python-version specific. If the internet-connected computer you are using to download the packages does not have the exact same version of Python and operating system as your offline Display PC, you must explicitly tell `pip` which versions to target.

Step 1: Download the packages on an internet-connected computer

Standard Download (If both PCs have the exact same Python version and OS): Open a terminal or command prompt on a machine with internet access and run: `pip download pygame sr-research-pylink pygame-eyelink-coregraphics -d ./eyelink_packages`

Targeted Download (If the PCs have different Python versions or OS): You can force `pip` to download the specific pre-compiled `.whl` files (wheels) for your target Display PC. You must use the `--only-binary=:all:` flag (which tells `pip` to only fetch `.whl` files and ignore source distributions), along with the specific `--python-version`, `--platform`, and `--abi` of the offline machine.

To determine your `--abi` and `--platform` tags:

- **--abi:** For standard Python installations, this is simply `cp` followed by the Python version without the dot. For example, Python 3.9 is `cp39`, Python 3.10 is `cp310`, and Python 3.11 is `cp311`.

- **--platform:** Common platform tags include `win_amd64` (64-bit Windows), `macosx_10_9_x86_64` (Intel macOS), `macosx_11_0_arm64` (Apple Silicon macOS), or `manylinux2014_x86_64` (Linux).

For example, to download the packages for a Display PC running **64-bit Windows** and **Python 3.10**, you would run: `pip download --only-binary=:all: --python-version 3.10 --platform win_amd64 --abi cp310 pygame sr-research-pylink pygame-eyelink-coregraphics -d ./eyelink_packages`

Step 2: Transfer the files Move the `eyelink_packages` folder to your offline Display PC using a USB drive.

Step 3: Install the packages on the Display PC Open a terminal or command prompt on your offline Display PC, navigate to the location where you saved the `eyelink_packages` folder, and run the following command to install the packages directly from the local files: `pip install --no-index --find-links=./eyelink_packages pygame sr-research-pylink pygame-eyelink-coregraphics`

4.2 Technical Support

If you encounter any issues running the provided example scripts, or if you need assistance integrating EyeLink tracking into your own Python experiments, please do not hesitate to reach out to our support team.

You can contact us directly at support@sr-research.com.

To help us resolve your issue as quickly as possible, please include the following information in your email:

- Your EyeLink model and Host PC software version.
- The Operating System of your Display PC (e.g., Windows 11, macOS 26, Ubuntu 22.04).
- The version of Python you are using.
- A brief description of the issue, including any specific error messages or the name of the example script you are trying to run.