Gaze Recording with Tobii Eye Trackers: Usage of **screen based Tobii Eye Trackers** from within a 3rd party program

Tutorial

Simon Maurer - simon.maurer@humdek.unibe.ch
23rd June 2023

Contents

Chapter 1

Introduction

This documents describes how to use the Tobii Eye Tracker Spark and the Tobii Eye Tracker 4C to record gaze data of a subject and (optionally) feed the gaze data to the mouse device. This allows to use the eye tracker to control the mouse pointer position such that the mouse pointer is placed at the screen coordinates where the subject is gazing at. This documentation comes with a set of tools (executable .exe files) that provide this functionality as well as some auxiliary tools that help to calibrate and test the eye tracker.

The project aims at providing a set of executables which allow to use the Eye Trackers in conjunction with third party applications that allow to execute external programs. Specifically, ztree or openseame. The following set of executables are provided:

CustomCalibrate.exe This program uses the Tobii Pro SDK and launches a custom calibration process which allows to calibrate the eye tracker without having to rely on the unstable calibration software provided by Tobii.

TobiiCalibrate.exe This program is a simple wrapper for the Tobii calibration tool. It launches the calibration GUI where the subject is led through the calibration process. The calibration data is stored in the current profile of the eye tracker engine. **Attention:** At the time of this writing (05.06.2023) the calibration program reachable through the Tobii Pro Eye Tracker Manager cannot be used due to an issue that is currently investigated by Tobii.

Gaze.exe This program uses the Tobii Pro SDK to extract the gaze position on the screen where the subject is looking at. The extracted data is recorded and stored to a file. Optionally, the mouse cursor position is updated to this position such that the mouse cursor is controlled by the gaze of the subject. Instead of using an eye tracker device it is also possible to simply log the mouse coordinates. `Gaze.exe` runs infinitely until it is terminated by an external command. This should **not** be done with a forced kill (e.g. by executing the command `taskkill /F /IM Gaze.exe` or by killing the task with the task manager) because it prevents the program from terminating gracefully. This as several consequences:

- open files are not closed properly and the data stream is cut off. This can lead to corrupt files.
- if the feature of hiding the mouse pointer is used, the mouse will remain hidden.
- memory is not freed properly.

Instead the program `GazeControls.exe /command TERMINATE` should be used.

GazeControl.ext This program allows to interact with Gaze.exe by passing the argument `/command <COMMAND>` to the application. Passing an argument to an application can be done in command line or by crating a shortcut to the program. Corresponding shortcuts for all available `<COMMAND>`s are provided in the release package. The following `<COMMAND>`s are available:

`CUSTOM_CALIBRATE` uses the Tobii Pro SDK and launches a custom calibration process which allows to calibrate the eye tracker without having to rely on the calibration software provided by Tobii.

`DRIFT_COMPENSATION` launches a custom drift compensation process to compensate gaze drifts that may occur during experimentation.

`GAZE_RECORDING_DISABLE` requests Gaze.exe to stop recording gaze data. Gaze.exe will continue to run (and update the mouse pointer if configured accordingly) but no longer store gaze data to the disk.

`GAZE_RECORDING_ENABLE` requests Gaze.exe to start recording gaze data.

`MOUSE_TRACKING_DISABLE`requests Gaze.exe to stop updating the mouse pointer by the gaze position.

`MOUSE_TRACKING_ENABLE` requests Gaze.exe to start updating the mouse pointer by the gaze position.

`RESET_DRIFT_COMPENSATION` resets the drift compensation computed with the command `DRIFT_COMPENSATION`.

`TERMINATE` requests Gaze.exe to close gracefully and logs these events to the log file.

ShowMouse.exe This program allows to restore the standard mouse pointer. It might be useful if the program `Gaze.exe` crashes or is closed forcefully such that the mouse pointer is not restored after terminating. The subject might end up with a hidden mouse pointer. A good solution for such a case is to install a shortcut to `ShowMouse.exe` on the desktop in order to execute it with the keyboard.

1.1 Requesting a TPF GitLab Account

The necessary code for what is described in this document is contained in a `GitLab` instance running on the TPF server[1].

Please send a mail to TPF[2] with a request, specifying your name and the name of the TPF project you are interested in (e.g `tobii_eye_tracker_gaze`).

[1] http://phhum-g111-nns.unibe.ch:10012/
[2] tpf@humdek.unibe.ch

Chapter 2

Quick Start

In order to get started with Tobii eye tracking the following installation steps need to be performed:

- Install the Tobii Pro Eye Tracker Manager application.
- Setup the *Tobii Pro Spark* device with the following steps:
  - Connect the *Tobii Pro Spark* device to the computer. Make sure that the device is connected to a powered USB-type A port or, alternatively, use the provided USB-type A to USB-type C adapter and connect the device to a USB-type C port.
  - Launch the Tobii Pro Eye Tracker Manager application
  - Click on the button *Install New Eye Tracker*
  - Select the device *Tobii Pro Spark*
  - Click Install
  - Now the eye tracker device should appear in the list. Click on the device and setup the screen calibration.
  - Perform the device calibration and verify the gaze accuracy through the built-in gaze visualization
- Setup the *Tobii Eye Tracker 4C* device with the following steps:
  - Install the Tobii Experience software. This software comes with the device driver for the *Tobii Eye Tracker 4C*, however, it is only available as a beta version and is no longer maintained.
  - Connect the *Tobii Eye Tracker 4C* device to the computer.
  - Now the eye tracker device should appear in the list of available devices in the *Tobii Pro Eye TRacker Manager* application. Click on the device and setup the screen calibration.
  - Perform the device calibration and verify the gaze accuracy through the built-in gaze visualization
- Install the Gaze toolset provided by the TPF. To install the toolset download[1] the latest version (requires a subscription, see Section 1.1) and extract all files to an installation path of your choosing. The Gaze toolset installation path will be referred to as `<Gaze toolset path>` throughout this document.

2.1 Quick Start with ztree

In order to get started with a quick experiment you need the following things (*server* refers to the machine running `ztree` and *client* refers to the machines running `zleaf`):

- a ztree installation with a server and one or more clients. To install `ztree` download the latest `ztree` version from the download secetion[2] (requires a license and a login, see Section 2.1.1) and extract the server file `ztree.exe` to the chosen installation path on the server and the client file `zleaf.exe` to the chosen installation path on each client. Throughout this documentation the installation paths of the `ztree` client and server will be referred to as `<zleaf path>` and `<ztree path>`, respectively.

[1] http://phhum-g111-nns.unibe.ch:10012/TBI/TBI-tobii_eye_tracker_gaze/tree/master/release
[2] https://www.uzh.ch/ztree/ssl-dir/index.php

- the ztree sample file. The file is located in <Gaze toolset path>\sample\template.ztt. Make sure to change the Path variable in Background of the sample file such that it points to <Gaze toolset path>.

The provided sample ztree program performs the following stages:

Calibrate Eye Tracker A guest calibration is performed where the Tobii calibration tool is started and the calibration data is stored in a guest profile within the Tobii eye tracker software. The calibration is very intuitive and has a game-like feel to it.

Test Eye Tracker The Tobii eye tracker test tool is started which allows the subject or experimenter to verify the accuracy of the eye tracker. If the accuracy is not sufficient a recalibration can be started from within the test tool.

Gaze To Mouse The gaze of the subject is tracked and transformed to mouse coordinates which allows to control the mouse pointer with the gaze of the subject.

Terminate Terminates the gaze-to-mouse functionality and concludes this simple ztree program.

The behaviour of the Gaze toolset can be configured with a configuration file (see Chapter 4 for more details). A sample configuration file is provided in <Gaze toolset path>\sample\config.json that can be modified to suit different requirements. Copy the configuration file either to <Gaze toolset path> or <zleaf path> and modify it there.

2.1.1 Requesting Access to the Latest ztree Release

To download the latest release of the ztree software a login must be requested here[3]. Make a careful note of the terms and conditions.

2.2 Quick Start with opensesame

In order to get started with a quick experiment the following steps are required:

- Install the latest version of opensesame with your preferred installation method.
- As a starting point use the provided opensesame sample file. The file is located in <Gaze toolset path>\sample\template.o
  Make sure to change the gazeAppPath variable in gaze_initialisation of the sample file such that it points to <Gaze toolset path>.

The provided sample opensesame program includes the following python scripts:

gaze_initialisation Sets the gazeAppPath variable which will be used in all gaze python script to locate the gaze toolset.

gaze_calibration The custom calibration application is started to perform a gaze calibration. Note that the subject_nr variable is passed as argument to the application. This is used to name the output file accordingly.

gaze_start The gaze application is started in the background and will run during the whole of the experiment. Note that the subject_nr variable is passed as argument to the application. This is used to name the output file accordingly.

gaze_recording_enable Communicates to the gaze application to start storing gaze data to the disk.

gaze_recording_disable Communicates to the gaze application to stop storing gaze data to the disk. The Tobii eye tracker test tool is started which allows the subject or experimenter to verify the accuracy of the eye tracker.

gaze_stop Instructs the gaze application to gracefully shutdown.

The behaviour of the Gaze toolset can be configured with a configuration file (see Chapter 4 for more details). A sample configuration file is provided in <Gaze toolset path>\sample\config.json that can be modified to suit different requirements. Copy the configuration file either to <Gaze toolset path> or <zleaf path> and modify it there.

[3] https://www.uzh.ch/ztree/ssl-dir/index.php?action=obtain

Chapter 3

Use Gaze Toolset in 3rd Applications

This chapter provides instructions of how to use the Gaze toolest in a 3rd party application. The gaze toolest contains several executable applications which can be started from within 3rd party applications to add eye tracker support. The two main pieces is the `CustomCalibration` application which allows to calibrate the eye tracker and the `Gaze` application which allows to record gaze data. The applications `GazeClose`, `GazeRecordingDisable`, `GazeRecordingEnable`, `MouseTrackingDisable`, and `MouseTrackingEnable` only work in conjunction with the `Gaze` application and are used to control the `Gaze` application (refer to Section 1 for an overview).

3.1 Basic Principles of a Screen Based Tobii Eye Tracker

The eye tracker is mounted on the bottom of the screen such that its cameras are able to capture the eyes of the subject. Several infrared LEDs are visible once the device is connected and properly working (see Figure 3.1). This requires the Tobii software to be installed and running.

Figure 3.1: Picture of the Tobii Eye Tracker 4C in operation.

In order for the eye tracker to being able to track the gaze of the subject it must be set up correctly and calibrated. The setup process must only be done once an can be done through the Tobii Pro Eye Tracker Manager application. However, the eye tracker must be calibrated for each individual the gaze data is measured. Hence, the experiment protocol needs to include a calibration step for each subject. A user-friendly calibration tool is included in the Tobii software package and a custom calibration tool `CustomCalibration` is provided in the gaze toolest. Once the eye tracker is calibrated for a subject, everything is ready.

3.2 Execute 3rd Party Applications in `ztree`

This section describes how to execute application from within `ztree`. In a first step, it is useful to define a global variable `Path` in a `ztree` program that points to the location of the toolset.

In order to define a path

1. click on the last table in `Background`
2. choose the menu `Treatment` → `New Program...`
3. define the path of the installation folder of the `Gaze` toolset, i.e. `<Gaze toolset path>` as `Path` variable (e.g. `Path="C:\\Users\\Max Muster\\Documents\\My Experiment\\Gaze\\";`).
   Note that two backslashes are required for each path delimiter because in ztree \ is used as escape character.

When writing the different stages of a ztree program, calls to external applications can be made. This can be achieved as follows:

1. click on the position in your ztree experiment file where you want to include the call to the external program
2. choose the menu Treatment → New External Program...
3. choose Run on z-Leaf
4. add the call to the external program to the field Command Line
   (e.g a call to the guest calibration tool: <><Path|-1>TobiiGuestCalibrate.exe
   Note that the Path variable is used to indicate the location of the program. Figure 3.2 shows an example of calling the Tobii guest calibration tool.

Figure 3.2: External program call definition in ztree. The variable Path must be globally defined.

3.3 Execute 3rd Party Applications in opensesame

Opensesame allows to run python scripts within the experiment protocols and python can be used to start external applications. All python scripts within an opensesame experiment share the same workspace which means that a variable defined in one script will be accessible by all other scripts as well. This can be used to define the installation path of the gaze toolset in order to avoid typing the same path multiple times. To to this append a new item *inline_script* to the experiment and add the following lines (make sure to set the path to where the gaze toolset was installed):

```
gazeToosetPath="C:\\GazeToolset"
print(f"use gaze toolset path {gazeToolsetPath}")
```

For each application to execute from within the experiment use a separate *inline_script*. To run the CustomCalibration application use the following script (note that the subject number is passed as an argument to the application):

```
import subprocess

print("start custom calibration")
subprocess.run([f"{gazeToolsetPath}\\CustomCalibrate.exe], "/subject", f"{var.get(u'subject_nr')}")
print("custom calibration done")
```

To run the Gaze application use a command to run the application in the background instead of waiting for the application to terminate:

```
import subprocess

print("start gaze process")
subprocess.Popen([f"{gazeToolsetPath}\\Gaze.exe], "/subject", f"{var.get(u'subject_nr')}")
```

To run any other application of the gaze toolset use subprocess.run without passing any arguments. For example, to gracefully terminate the Gaze application with GazeClose, use the following script:

```
import subprocess

print("stop gaze process")
subprocess.run([f"{gazeToolsetPath}\\GazeClose.exe]")
```

Chapter 4

Configuration of the Gaze Toolset

The Gaze toolset can be configured to work with different installations. It allows to specify installation paths and gives some control over implemented features (e.g. mouse hiding or data logging). The Listing 4.1 shows the default configuration values and provides an explanation for each value. The configuration file contains detailed descriptions for each parameter. It is recommended to go through all the configuration parameters and read the description in order to get an understanding of the available configuration options.

```
{
  // Allows to define the order and the delimiters between the different calibration result values.
  // The definition is of the form
  //  {0}<delim>{1}<delim> ... <delim>{24}
  // where <delim> can be customized (e.g. '\t' for tab, ',' for comma, etc.) and where the numbers are
  // replaced by the following values
  //  - 0: x-coordinate of the calibration point (normalized value)
  //  - 1: y-coordinate of the calibration point (normalized value)
  //  - 2: x-coordinate of the gaze point of the left eye (normalized value)
  //  - 3: y-coordinate of the gaze point of the left eye (normalized value)
  //  - 4: validity of the gaze data of the left eye
  //  - 5: x-coordinate of the gaze point of the right eye (normalized value)
  //  - 6: y-coordinate of the gaze point of the right eye (normalized value)
  //  - 7: validity of the gaze data of the right eye
  // To log all possible values with a tab (i.e. '\t') as delimiter use the empty string:
  // "CalibrationLogColumnOrder": "",
  // This configuration value has no effect if "CalibrationLogWriteOutput" is set to false.
  "CalibrationLogColumnOrder": "",

  // Defines the titles of the calibration log value columns. A title for all possible columns must
  // be defined. Titles for values that are removed from the "CalibrationLogColumnOrder" parameter
  // will not be logged but must still be defined here. The index of a title must correspond to the
  // value number of the configuration parameter "CalibrationLogColumnOrder".
  // This configuration value has no effect if "CalibrationLogWriteOutput" is set to false.
  "CalibrationLogColumnTitle": [
    "calibrationPoint_x",
    "calibrationPoint_y",
    "left_gazePoint_x",
    "left_gazePoint_y",
    "left_gazePoint_isValid",
    "right_gazePoint_x",
    "right_gazePoint_y",
    "right_gazePoint_isValid"
  ],

  // Define the calibration points to be shown during the calibration process. Each point is given as
  // a normalize coordinate where [0, 0] is the top left corner and [1, 1] the bottom right corner of
  // the screen. Any number of points is permitted.
  "CalibrationPoints": [
    [ 0.7, 0.5 ],
    [ 0.3, 0.5 ],
    [ 0.9, 0.9 ],
```

```
    [ 0.1, 0.9 ],
    [ 0.5, 0.1 ],
    [ 0.1, 0.1 ],
    [ 0.9, 0.1 ],
    [ 0.5, 0.9 ]
  ],

  // Defines whether gaze calibration data is written to a log file. If set to false, all the configuration
  // items matching the pattern "CalibrationLog*" are ignored.
  "CalibrationLogWriteOutput": true,

  // Use this parameter to associate the configuration with an experiment. When "Gaze.exe" is
  // executed, a copy of this configuration file is stored at the "DataLogPath" where the parameter
  // "ConfigName" is postfixed to the filename of the copied config file. E.g., by default the following
  // file will be produced at "DataLogPath": <timestamp>_<computer name>_config_experiment_x.json
  // Note that the following characters are not allowed in a file name: <>:"/\|?*
  "ConfigName": "experiment_x",

  // Allows to define the order and the delimiters between the different gaze data values.
  // The definition is of the form
  //  {0}<delim>{1}<delim> ... <delim>{24}
  // where <delim> can be customized (e.g. '\t' for tab, ',' for comma, etc.) and where the numbers are
  // replaced by the following values

  //  - 0: timestamp of the gaze data item (uses DataLogFormatTimeStamp)

  //  - 1: x-coordinate of the drift compensated combined 2d gaze point (normalized value)
  //  - 2: y-coordinate of the drift compensated combined 2d gaze point (normalized value)
  //  - 3: x-coordinate of the raw combined 2d gaze point (normalized value)
  //  - 4: y-coordinate of the raw combined 2d gaze point (normalized value)
  //  - 5: flag indicating whether the combined 2d gaze point values are valid
  //  - 6: x-coordinate of the drift compensated combined 3d gaze point (mm in UCS)
  //  - 7: y-coordinate of the drift compensated combined 3d gaze point (mm in UCS)
  //  - 8: y-coordinate of the drift compensated combined 3d gaze point (mm in UCS)
  //  - 9: x-coordinate of the raw combined 3d gaze point (mm in UCS)
  //  - 10: y-coordinate of the raw combined 3d gaze point (mm in UCS)
  //  - 11: z-coordinate of the raw combined 3d gaze point (mm in UCS)
  //  - 12: flag indicating whether the combined 3d gaze point values are valid
  //  - 13: x-coordinate of the combined 3d gaze origin (mm in UCS)
  //  - 14: y-coordinate of the combined 3d gaze origin (mm in UCS)
  //  - 15: z-coordinate of the combined 3d gaze origin (mm in UCS)
  //  - 16: flag indicating whether the combined 3d gaze origin values are valid
  //  - 17: the distance of the gaze origin to the gaze point (mm)

  //  - 18: x-coordinate of the raw left 2d gaze point (normalized value)
  //  - 19: y-coordinate of the raw left 2d gaze point (normalized value)
  //  - 20: flag indicating whether the left 2d gaze point values are valid
  //  - 21: x-coordinate of the raw left 3d gaze point (mm in UCS)
  //  - 22: y-coordinate of the raw left 3d gaze point (mm in UCS)
  //  - 23: z-coordinate of the raw left 3d gaze point (mm in UCS)
  //  - 24: flag indicating whether the left 3d gaze point values are valid
  //  - 25: x-coordinate of the left 3d gaze origin (mm in UCS)
  //  - 26: y-coordinate of the left 3d gaze origin (mm in UCS)
  //  - 27: z-coordinate of the left 3d gaze origin (mm in UCS)
  //  - 28: flag indicating whether the left 3d gaze origin values are valid
  //  - 29: the distance of the gaze origin to the gaze point (mm)
  //  - 30: the pupil diameter of the left eyes (mm)
  //  - 31: flag indicating whether the pupil diameter is valid

  //  - 32: x-coordinate of the raw right 2d gaze point (normalized value)
  //  - 33: y-coordinate of the raw right 2d gaze point (normalized value)
  //  - 34: flag indicating whether the right 2d gaze point values are valid
  //  - 35: x-coordinate of the raw right 3d gaze point (mm in UCS)
  //  - 36: y-coordinate of the raw right 3d gaze point (mm in UCS)
  //  - 37: z-coordinate of the raw right 3d gaze point (mm in UCS)
  //  - 38: flag indicating whether the right 3d gaze point values are valid
```

```
//  - 39: x-coordinate of the right 3d gaze origin (mm in UCS)
//  - 40: y-coordinate of the right 3d gaze origin (mm in UCS)
//  - 41: z-coordinate of the right 3d gaze origin (mm in UCS)
//  - 42: flag indicating whether the right 3d gaze origin values are valid
//  - 43: the distance of the gaze origin to the gaze point (mm)
//  - 44: the pupil diameter of the right eyes (mm)
//  - 45: flag indicating whether the pupil diameter is valid

// To log all possible values with a tab (i.e. '\t') as delimiter use the empty string:
// "DataLogColumnOrder": "",
// This configuration value has no effect if "DataLogWriteOutput" is set to false.
"DataLogColumnOrder": "",

// Defines the titles of the gaze data log value columns. A title for all possible columns must be
// defined. Titles for values that are removed from the "DataLogColumnOrder" parameter will not be
// logged but must still be defined here. The index of a title must correspond to the value number
// of the configuration parameter "DataLogColumnOrder".
// This configuration value has no effect if "DataLogWriteOutput" is set to false.
"DataLogColumnTitle": [
  "timestamp",

  "combined_gazePoint2dCompensated_x",
  "combined_gazePoint2dCompensated_y",
  "combined_gazePoint2d_x",
  "combined_gazePoint2d_y",
  "combined_gazePoint2d_isValid",
  "combined_gazePoint3dCompensated_x",
  "combined_gazePoint3dCompensated_y",
  "combined_gazePoint3dCompensated_z",
  "combined_gazePoint3d_x",
  "combined_gazePoint3d_y",
  "combined_gazePoint3d_z",
  "combined_gazePoint3d_isValid",
  "combined_originPoint3d_x",
  "combined_originPoint3d_y",
  "combined_originPoint3d_z",
  "combined_originPoint3d_isValid",
  "combined_gazeDistance",
  "combined_pupilDiameter",
  "combined_pupilDiameter_isValid",

  "left_gazePoint2d_x",
  "left_gazePoint2d_y",
  "left_gazePoint2d_isValid",
  "left_gazePoint3d_x",
  "left_gazePoint3d_y",
  "left_gazePoint3d_z",
  "left_gazePoint3d_isValid",
  "left_gazeOrigin3d_x",
  "left_gazeOrigin3d_y",
  "left_gazeOrigin3d_z",
  "left_gazeOrigin3d_isValid",
  "left_gazeDistance",
  "left_pupilDiameter",
  "left_pupilDiameter_isValid",

  "right_gazePoint2d_x",
  "right_gazePoint2d_y",
  "right_gazePoint2d_isValid",
  "right_gazePoint3d_x",
  "right_gazePoint3d_y",
  "right_gazePoint3d_z",
  "right_gazePoint3d_isValid",
  "right_gazeOrigin3d_x",
  "right_gazeOrigin3d_y",
  "right_gazeOrigin3d_z",
```

```
    "right_gazeOrigin3d_isValid",
    "right_gazeDistance",
    "right_pupilDiameter",
    "right_pupilDiameter_isValid"
  ],

  // Number of maximal allowed output data files in the output path. Oldest files are deleted first. To
  // keep all files set the value to 0. A value of 1 means that only the output of the current execution
  // is kept.
  // Note that if multiple clients write to the same folder, this value should be set to at least the
  // number of clients.
  // This configuration value has no effect if "DataLogWriteOutput" is set to false.
  "DataLogCount": 200,

  // Defines whether gaze data storing is disabled on Gaze application start. If set to false gaze
  // data will be stored to the outupt gaze file as soon as the device connection is established.
  // If set to true data storing must be enabled manually through the application GazeRecordingEnable.
  // This configuration value has no effect if "DataLogWriteOutput" is set to false.
  "DataLogDisabledOnStartup": false,

  // Allows to define the format of how the pupil diameter (in millimetres) will be logged. Use the .NET
  // syntax to specify the format:
  //  https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types
  // Note that the numbers will be represented differently depending in the localisation settings of the
  // windows installation (e.g. 123,4 for DE_CH or 123.4 for EN_US).
  // This configuration value has no effect if "DataLogWriteOutput" is set to false.
  "DataLogFormatDiameter": "0.000",

  // Allows to define the format of how normalized data points will be logged. Use the
  // .NET syntax to specify the format:
  //  https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types
  // Note that the numbers will be represented differently depending in the localisation settings of the
  // windows installation (e.g. 123,4 for DE_CH or 123.4 for EN_US).
  // This configuration value has no effect if "DataLogWriteOutput" is set to false.
  "DataLogFormatNormalizedPoint": "0.000",

  // Allows to define the format of how the gaze origin values (in millimetres) will be logged. Use the
  // .NET syntax to specify the format:
  //  https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types
  // Note that the numbers will be represented differently depending in the localisation settings of the
  // windows installation (e.g. 123,4 for DE_CH or 123.4 for EN_US).
  // This configuration value has no effect if "DataLogWriteOutput" is set to false.
  "DataLogFormatOrigin": "0.00",

  // Allows to define the format of the timestamp. Use the .NET syntax to specify the format:
  //  https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types
  // Note that special characters (e.g. ':', '.') need to be escaped with '\\'.
  // This configuration value has no effect if "DataLogWriteOutput" is set to false.
  "DataLogFormatTimeStamp": "hh\\:mm\\:ss\\.fff",

  // Defines the location of the output file. It must be the path to a folder (not a file). If empty,
  // the output file is produced in the directory of the caller (e.g the directory of zleaf.exe).
  // This configuration value has no effect if "DataLogWriteOutput" is set to false.
  // To avoid confusion with path locations it is recommended to use absolute paths, e.g.:
  //  C:\\Users\\Subject\\Documents
  "DataLogPath": "",

  // Defines whether gaze data is written to a log file. If set to false, all the configuration items
  // matching the pattern "DataLog*" are ignored.
  "DataLogWriteOutput": true,

  // In order to detect a fixation with the I-DT algorithm a dispersion threshold is required.
  // Provide an angle in degrees. The fixation detection is only used for drift compensation (for
  // calibration and validation the functions provided by the manufactor are used).
  "DispersionThreshold": 0.5,
```

```
  // Specifies the amount of time (in milliseconds) to wait for a fixation point during drift compensation.
  // If the timer elapses drift compensation will be aborted. Use a value of zero for infinite timout.
  "DriftCpompensationTimer": 5000,

  // Defines the location of the license files. It must be the path to a folder (not a file).
  // This is only required if the eye tracker device requires an external license file.
  // If an eye tracker does not require a license file either omit this configuration item or
  // set the empty string. Use %S as a placeholder for the device serialnumber and %A as a
  // placeholder for the device address. The placeholders will be replaced by the actual values
  // of the first device found in the connection list.
  // To avoid confusion with path locations it is recommended to use absolute paths, e.g.:
  //  C:\\Users\\Subject\\Documents\\tobii_licenses
  "LicensePath": "",

  // Defines whether the mouse cursor shall be hidden on the calibration window.
  "MouseCalibrationHide": false,

  // Defines wheter the mouse cursor shall be controlled by the gaze of the subject during the
  // experiment. If set to true the mouse cursor will be controlled by the gaze of the subject when
  // Gaze.exe is executed and control will be released when GazeClose.exe is executed.
  "MouseControl": false,

  // Defines whether the mouse cursor shall be hidden during the experiemnt. If set to true the
  // mouse cursor will be hidden when Gaze.exe is executed and restored when GazeClose.exe is executed.
  // This parameter is ignored if "MouseControl" is set to false.
  "MouseControlHide": false,

  // Defines the Path to the standard mouse pointer icon. This is used to restore the mouse pointer.
  // This parameter is ignored if "MouseControl" or "MouseHide" is set to false.
  "MouseStandardIconPath": "C:\\Windows\\Cursors\\aero_arrow.cur",

  // Specifies the amount of time (in milliseconds) to wait for the eyetracker to become ready while it
  // is in any other state. If the eyetracker is not ready within the specified time the subject will
  // be notified with a popup window. This is only relevant for Gaze.exe as the CustomCalibration.exe
  // reacts immediately to lost connections and uses its own GUI to display an error message.
  // Use 0 for immediate reaction.
  "ReadyTimer": 5000,

  // Defines the Tobii installation path. It must be the path to a folder (not a file).
  "TobiiApplicationPath": "C:\\<LoaclApplicationData>\\Programs\\TobiiProEyeTrackerManager",

  // The Tobii application to run a calibration.
  "TobiiCalibrate": "TobiiProEyeTrackerManager.exe",

  // The arguments to pass to the calibration application. Use %S as a placeholder for the device
  // serialnumber and %A as a placeholder for the device address. The placeholders will be replaced
  // by the actual values of the first device found in the connection list.
  "TobiiCalibrateArguments": "--device-sn=%S --mode=usercalibration",

  // Choose the tracker device (1: Tobii Pro SDK, 2: Mouse Tracker).
  // Note that for some eye trackers the Tobii SDK Pro requires a license file to work
  // (see parameter "LicesePath").
  "TrackerDevice": 1
}
```

Listing 4.1: Default configuartion values

Note that the configuration file follows the `json` syntax which must not be violated. If the following points are respected, no problem should arise:

- the configuration parameters are enclosed in '{' and '}'.
- all configuration parameters are of the form `"key":value` where `"key"` must not be changed.
- each configuration line ends with a ',' except for the last line where it is omitted.
- the Windows path delimiter '\' must be escaped (i.e. write '\\' when describing a path)

- `json` supports standard data types (e.g. integer, boolean, string). Use the same type as the default value.
- everything following a '//' is considered a comment.

Each executable of the toolset uses the same common configuration file. The configuration file must be named `config.json` and is read from the following places with the indicated priority:

1. in the directory of the caller, i.e. in the execution folder of the ztree client or the opensesame application
2. in the directory of the executables, i.e. in `<Gaze toolset path>`

If no configuration file can be found, the application fails.

> **Warning:** A word of warning when using the mouse hiding feature. This feature hides the mouse when running `Gaze.exe`. If `Gaze.exe` is forcefully closed or crashes, the mouse pointer stays hidden. For such cases the `ShowMouse.exe` utility can be used to restore the mouse pointer.

4.1 Gaze Output File

When running the program `Gaze.exe` an output file can be generated which holds the gaze data provided by the Tobii engine. The output file is saved in the directory specified by `OutputPath` in `config.json`. The name of the output file follows the form

```
<yyyyMMddTHHmmss>_<hostName>_<ConfigName>[_<SubjectNumber>]_gaze[_err-<code>].txt
```

where

- `<yyyyMMddTHHmmss>` is replaced by the timestamp indicating when the file was created (e.g. 20180129T085521 stands for 29.01.2018 08:55:21).
- `<hostName>` is replaced by the name of the machine
- `<ConfigName>` is replaced by the configuration value `"ConfigName"` as specified in the configuration file (see Listing 4.1 for more details)
- `<SubjectNumber>` is replaced by the subject number passed by argument `/subject` to the application. If no argument `/subject` is passed the subject number is omitted in the file name (as well as the prefixed underline character).
- `[_err-<code>]` is either omitted if no error occurred during the experiment or indicates data errors where
  - `<code>` is replaced by an error code which is a binary string where each character can either be 1, indicating an error or 0, indicating no error (see Section 4.1.2)

A gaze data output file is only generated if the parameter `"DataLogWriteOutput"` in the configuration file is set to `true` (which is the default). The presentation of the data is configurable with the help of the parameters `"DataLogColumnOrder"`, `"DataLogColumnTitle"`, `"DataLogFormatDiameter"`, `"DataLogFormatOrigin"`, and `"DataLogFormatTimeStamp"` (see Listing 4.1 for more details).

A gaze point describes a point on the screen in x and y coordinates (pixel values) the user is gazing at. This coordinate system is called *Active Display Coordinate System (ADCS)* and is illustrated in Figure 4.1.

By default, the system is configured to log all available data points. However, this can easily be changed through the parameter `"DataLogColumnOrder"`. For example, to only store a minimalistic set of data points use the value `"{0}\t{3}\t{4}"`. This will only store the timestamp of when the gaze point was captured by the eye tracker (data field number 0) and the x and y coordinates of the gaze point (data field numbers 3 and 4, respectively). This produces an output file that is similar to the following:

```
Timestamp   combined_gazePoint2d_x  combined_gazePoint2d_y
12:51:55.409    426 342
12:51:55.420    430 341
12:51:55.431    431 341
...
```

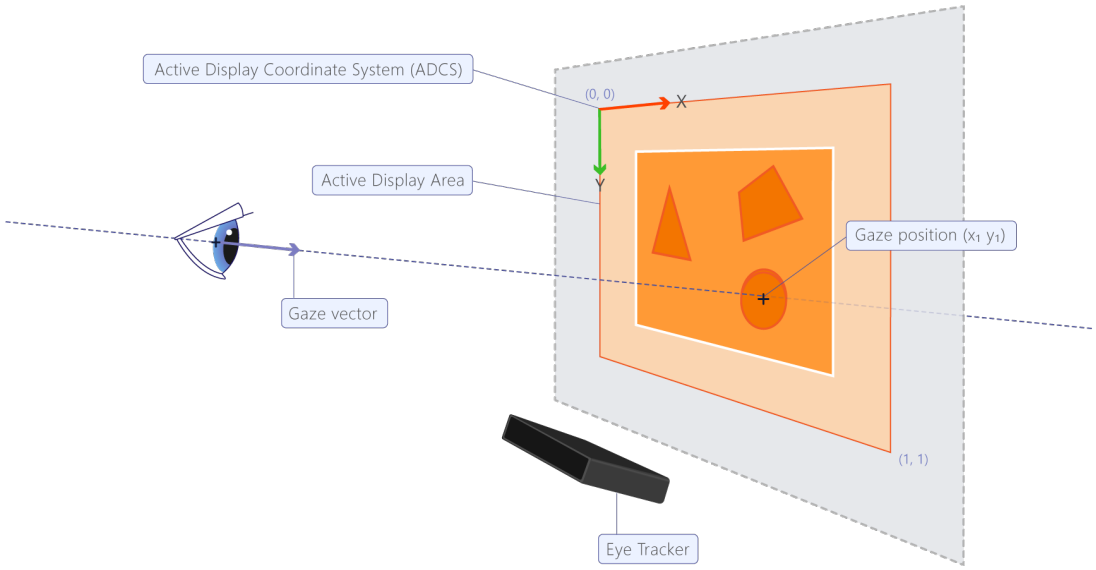

Figure 4.1: The Active Display Coordinate System (ADCS). Figure source: Tobii

When opening the file with a spreadsheet software such as LibreOffice Calc or Microsoft Office Excel two things need to be considered:

1. The column delimiter needs to be set to the delimiter set in the configuration file (`'\t'` by default).
2. The language needs to be set to the language of the windows system where the configuration file was produced. This is important because numbers are represented differently in different languages and depending on the settings, commas might be interpreted as delimiters when they are not.

When using Tobii Pro SDK, much more values are provided and can be logged to the output file. This includes the pupil diameters of each eye as well as the eye positions in space. The latter uses a coordinate system which is called *User Coordinate System (UCS)*. The UCS is illustrated in Figure 4.2.

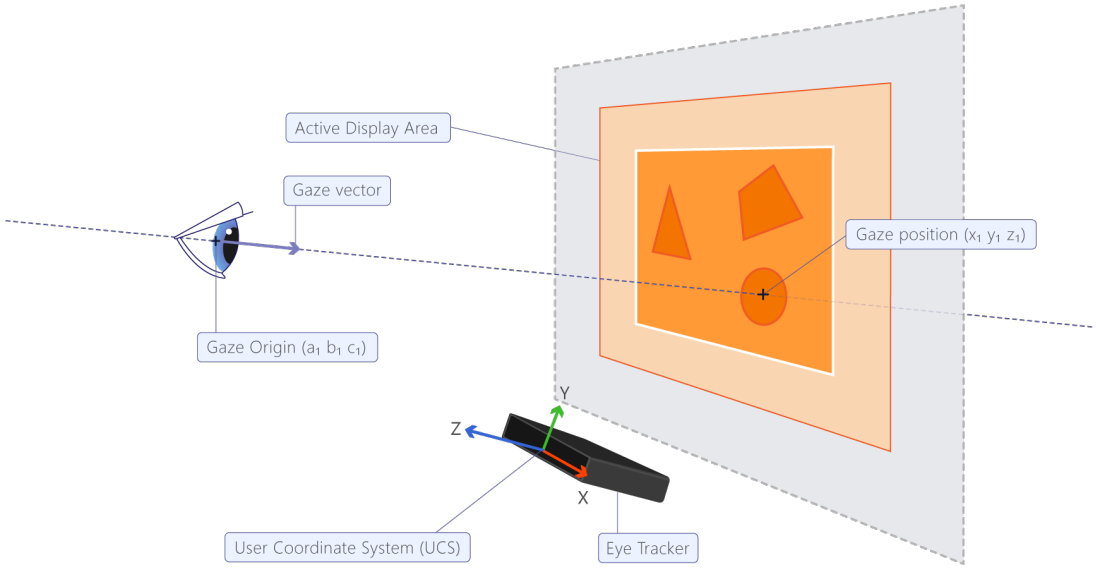

Figure 4.2: The User Coordinate System (UCS). Figure source: Tobii

In order to log additional data fields, the parameter `"DataLogColumnOrder"` must be modified by adding the required numbers, representing different data fields.

When using the empty string, all possible data fields are logged:

```
"DataLogColumnOrder": ""
```

4.1.1 Gaze Data Fields

The comments above the parameter `"DataLogColumnOrder"` in the configuration file (see Listing 4.1) provide a short description for each available data field. In the following some further information is provided.

The following five data field types with individual data fields are provided:

1. timestamps indicating when the data was sampled by the eye tracker (field number: 0)
2. 2D gaze point ADCS coordinates (see Figure 4.1) as normalized values

   Mouse Tracker

   - the x and y coordinates of the mouse pointer
     (field numbers: 3 and 4)

   Tobii Pro SDK

   - raw x and y coordinates of the left and the right eye
     (field numbers: 20, 21, 34, and 35)
   - the drift compensated x and y coordinates which are computed by the drift compensation process using the I-DT algorithm for fixation detection and then comparing the averaged gaze samples to an oracle gaze to computing the drift of the gaze
     (field numbers: 1 and 2)
   - the average x and y coordinates which are computed from raw data as defined by Equation 4.1
     (field numbers: 3 and 4)

$$\text{val} = \begin{cases} \text{NaN} & \text{if val\_right and val\_left are not valid} \\ \text{val\_left} & \text{if val\_right is not valid} \\ \text{val\_right} & \text{if val\_left is not valid} \\ \frac{\text{val\_left}+\text{val\_right}}{2} & \text{otherwise} \end{cases} \tag{4.1}$$

3. 3D gaze point UCS coordinates (see Figure 4.2) in millimeters

   Mouse Tracker not applicable

   Tobii Pro SDK

   - raw x, y, and z coordinates of the left and the right eye
     (field numbers: 23, 24, 25, 37, 38, and 39)
   - the drift compensated x, y, and z coordinates which are computed by the drift compensation process using the I-DT algorithm for fixation detection and then comparing the averaged gaze samples to an oracle gaze to computing the drift of the gaze
     (field numbers: 6, 7, and 8)
   - the average x and y coordinates which are computed from raw data as defined by Equation 4.1
     (field numbers: 9, 10, and 11)

4. gaze origin UCS coordinates (see Figure 4.2) in millimetres

   Mouse Tracker not applicable

   Tobii Pro SDK

   - raw x, y, and z coordinates of the left and the right eye
     (field numbers: 27, 28, 29, 41, 42, and 43)
   - the average x, y, and z coordinates which are computed from raw data as defined by Equation 4.1
     (field numbers: 13, 14, and 15)
   - the distance from the gaze origin to the left and the right gaze point, respectively which are computed from raw data as defined by Equation 4.2
     (field numbers: 31 and 45)

$$\text{dist} = \sqrt{x^2 + y^2 + z^2} \tag{4.2}$$

   - the average distance which is computed from the distance of the left and the right eye as defined by Equation 4.1
     (field number: 17)

5. pupil diameters in millimetres

Mouse Tracker not applicable

Tobii Pro SDK

- raw diameters of the left and the right eye
  (field numbers: 32 and 46)
- the average diameter which is computed from raw data as defined by Equation 4.1
  (field number: 18)

6. data validity indicators as true or false

Mouse Tracker not supported

Tobii Pro SDK

- separate values that indicate whether gaze points, gaze origins, and pupil diameters are valid for combined values, the left and the right eye, respectively
  (field numbers: 5, 12, 16, 19, 22, 26, 30, 33, 36, 40, 44, and 47)

Notice: The time resolution of the mouse tracker device is rather poor (15 milliseconds). Therefore it can happen that multiple mouse events are logged with the same timestamp.

4.1.2 Gaze Data Error Code

The data error code that is postfixed to the output file name is a binary string where each character indicates whether a specific error has occurred (indicated with the number `1`) or not (indicated with the number `0`). Multiple errors can occur at the same time, hence, the position of a `1` in the error code indicates the specific error. If all positions of the error code are 0, the errors are not postfixed to the output file name.

The following list describes the individual errors that can occur during an experiment:

code `10`
: This code indicates that during the experiment the eye tracker device stopped tracking. This means that in the output file gaze data is potentially missing. This can be caused by a malfunctioning of the eye tracker device or simply because the device was disconnected. The exact time instances of such an occurrence is logged to the log file.

code `01`
: This code indicates that the system had to fall back to the Mouse Tracker. This error occurs if the system was configured to use the Tobii Pro SDK but was not able to do so. This usually happens when the license file is not accessible. Check whether the license files are accessible by the system and whether the license path is correctly defined in parameter `"LicensePath"`.

4.2 Calibration Output File

When running the program `CustomCalibration.exe` an output file can be generated which holds the calibration results provided by the Tobii engine. The output file is saved in the directory specified by `OutputPath` in `config.json`. The name of the output file follows the form

```
<yyyyMMddTHHmmss>_<hostName>_<ConfigName>[_<SubjectNumber>]_calibration[_err-<code>].txt
```

where

- `<yyyyMMddTHHmmss>` is replaced by the timestamp indicating when the file was created (e.g. 20180129T085521 stands for 29.01.2018 08:55:21).
- `<hostName>` is replaced by the name of the machine
- `<ConfigName>` is replaced by the configuration value `"ConfigName"` as specified in the configuration file (see Listing 4.1 for more details)

- `<SubjectNumber>` is replaced by the subject number passed by argument `/subject` to the application. If no argument `/subject` is passed the subject number is omitted in the file name (as well as the prefixed underline character).
- `[_err-<code>]` is either omitted if no error occurred during the experiment or indicates data errors where
  - `<code>` is replaced by an error code which is a binary string where each character can either be `1`, indicating an error or 0, indicating no error (see Section 4.2.2)

A calibration output file is only generated if the parameter `"CalibrationLogWriteOutput"` in the configuration file is set to `true` (which is the default).

The configuration of the calibration data fields to be stored to the output file follows the same logic as described with the gaze output in Section 4.1, except that the presentation of the data is defined through parameters `"CalibrationLogColumnOrder"`, `"CalibrationColumnTitle"`, `"DataLogFormatNormalizedPoint"`, and `"DataLogFormatTimeStamp"` (see Listing 4.1 for more details).

4.2.1 Calibration Data Fields

The comments above the parameter `"CalibrationLogColumnOrder"` in the configuration file (see Listing 4.1) provide a short description for each available data field. In the following some further information is provided.

The following three data field types with individual data fields are provided:

1. calibration point ADCS coordinates (see Figure 4.1) as normalized values

   Mouse Tracker not applicable

   Tobii Pro SDK
   - x and y coordinates of the calibration point
     (field numbers: 0 and 1)

2. gaze point ADCS coordinates (see Figure 4.1) as normalized values

   Mouse Tracker not applicable

   Tobii Pro SDK
   - x and y coordinates of the gaze point of the left and right eye
     (field numbers: 2, 3, 5, and 6)

3. data validity indicators as either `ValidAndUsed`, `ValidAndUnused`, and `Invalid`

   Mouse Tracker not supported

   Tobii Pro SDK
   - separate values that indicate whether gaze points are valid for the left and the right eye, respectively
     (field numbers: 4 and 7)

4.2.2 Calibration Data Error Code

The error code that is postfixed to the calibration output file name is a binary string where each character indicates whether a specific error has occurred (indicated with the number `1`) or not (indicated with the number `0`). Multiple errors can occur at the same time, hence, the position of a `1` in the error code indicates the specific error. If all positions of the error code are 0, the errors are not postfixed to the output file name.

The following list describes the individual errors that can occur during an experiment:

code `10`
: This code indicates that during the calibration the eye tracker device stopped tracking. This means that the calibration had to be redone in order to be valid. This can be caused by a malfunctioning of the eye tracker device or simply because the device was disconnected. The exact time instances of such an occurrence is logged to the log file.

code `01`
: This code indicates that the system was not able to start the calibration because the device does not support calibration. This error occurs if the configuration option `TrackerDevice` was set to a device with no calibration support.

4.3 Configuration File Dump

For each experiment where the utility `Gaze.exe` is executed, a dump of the configuration file is produced. This allows to associate a set of configuration values to an experiment, reuse the same configuration file should the experiment be repeated, and provides transparency of how the output data was produced. Note that in this configuration file all comments are omitted and the formatting (indentations, white spaces, carriage return) is removed. To reformat the file or display the file as a tree structure, online tools, such as the Online JSON Viewer[1], can be used.

The name of the dumped configuration file is of the form

```
<yyyyMMddTHHmmss>_<hostName>_<ConfigName>_config[_err-<code>-<code>].txt
```

where

- `<yyyyMMddTHHmmss>` is replaced by the timestamp indicating when the file was created (e.g. 20180129T085521 stands for 29.01.2018 08:55:21)
- `<hostName>` is replaced by the name of the machine
- `<ConfigName>` is replaced by the configuration value `"ConfigName"` as specified in the configuration file (see Listing 4.1 for more details)
- `[_err-<code>-<code>]` is either omitted if no error occurred during the experiment or indicates data errors where
  - `<code>` is replaced by an error code which is a binary string where each character can either be `1`, indicating an error or 0, indicating no error (see Section 4.3.1)

4.3.1 Configuration Error Code

The configuration error codes that are postfixed to the dumped configuration file name are binary strings where each character indicates whether a specific error has occurred (indicated with the number `1`) or not (indicated with the number `0`). Multiple errors can occur at the same time, hence, the position of a `1` in the error code indicates the specific error. If all positions of all error codes are 0, the errors are not postfixed to the dumped configuration file.

The following list describes the individual errors of the first error code that addresses general problems with the configuration file:

code `100`
: The system ignores the configuration file and falls back to the default configuration values. This happens if an invalid configuration file is provided that cannot be parsed. Verify the syntax of the configuration file and make sure that the key names are not modified and no additional keys are added to the file. Note that it is perfectly valid to not provide a configuration file which causes the system to use the default values without creating an error.

code `010`
: The system uses the current location (e.g. `<zleaf path>`) to store the output files. This happens if the provided path in the configuration file (parameter `"DataLogPath"`) is invalid or non-existent. Verify that the provided path exists and that no invalid characters are used (do not use `<>:"/\|?`).

code `001`
: This happens if no name was provided or the provided name in the configuration file (parameter `"ConfigName"`) is invalid. The configuration name is mandatory and the application failed due to this error. Verify that in the provided name no invalid characters are used (do not use `<>:"/\|?`).

The following list describes the individual errors of the second error code that specifically addresses problems concerning the formatting of the output file:

code `010000`
: The system falls back to the default column order. This happens if the parameter `"DataLogColumnOrder"` is invalid. Make sure that only existing data filed numbers are used and that the format string is valid. More information on format strings is provided on the MSDN page about Composite Formatting[2].

[1] http://jsonviewer.stack.hu/
[2] https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite-formatting

code 001000
The column titles are not printed to the output file. This happens if the value of parameter `"DataLogColumnTitle"` is not valid. Make sure that a title is provided for all possible data fields (not only the ones that are displayed).

code 000100
The system falls back to the default format for the timestamp. This happens if the value provided for the parameter `"DataLogFormatTimeStamp"` is invalid. Refer to the MSDN page about Formatting Types[3] for more information.

code 000010
The system falls back to the default format for the gaze origin coordinates. This happens if the value provided for the parameter `"DataLogFormatOrigin"` is invalid. Refer to the MSDN page about Formatting Types[3] for more information.

code 000001
The system falls back to the default format for the diameter of the pupil. This happens if the value provided for the parameter `"DataLogFormatDiameter"` is invalid. Refer to the MSDN page about Formatting Types[4] for more information.

code 100000
The system falls back to the default format for the normalized coordination point. This happens if the value provided for the parameter `"DataLogNormalizedPoint"` is invalid. Refer to the MSDN page about Formatting Types[5] for more information.

4.4 Log File

All executables write continuously to the same log file. This allows to track the eye tracker events that happened throughout a session within one log file. The log file is produced at the root directory of the application which is making the calls to the executables (e.g. at the location of zleaf.exe: `<zleaf path>`). The name of the log file is of the form `<hostName>_gaze.log` where `<hostName>` is replaced by the name of the machine.

[3] https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types
[4] https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types
[5] https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types