User Manual for FSenSync

Version 101

(The manual is a work in progress)

30st of January 2019

Klaus Förger, klaus@forger.fi

Table of Contents

1 Introduction

The purpose of this manual is to give a concise description of all the FSenSync apps, desktop programs, and the produced data files. The FSenSync (Förger Analytics Sensor Synchronization) is a collection of apps running on Android devices and desktop programs which are connected and temporally synchronized by the FSenSync Server. Three common use cases for FSenSync include creation of video documentation, running of scientific experiments which use on sensor data and timed stimuli, and media art/games with data streaming over WLAN.

The main idea of the FSenSync is to provide a way to synchronize clocks robustly over WLAN. The FSenSync also aims to provide access to data from various sensors in a simple CSV (Comma Separated Values) format which can be easily downloaded to a computer and analyzed with pretty much any tool you might want to use. The FSenSync is intentionally build up from separate apps, because that allows people to use and develop only those parts of FSenSync that they really need. This approach has enabled very fast prototyping and development of new apps. The approach also makes it easy to throw away apps that do not work well, thus reducing the accumulation of garbage in the code base.

The FSenSync is useful as an additional layer on top of sensors as if you are able to measure things accurate enough you will find that:

1. No two clocks run at the exactly same pace
2. No sensors or output channels are free of lag

The FSenSync was originally developed by Klaus Förger (klaus@forger.fi) for analysis of improvised dance in the ICI project (funded and run by Paris 8 University and CNRS). Later on it has been developed further to fit the needs of the Aalto Behavioral Laboratory run by the Aalto University Department of Neuroscience and Biomedical Engineering.

Most parts of the FSenSync are licensed under GNU GPL v3. Exceptions include the BioHarness and the Move apps as they contain parts that are not compatible with the GPL license. The FSenSync Common Libraries (used in Android apps) and FSenSync Desktop Libraries (used in desktop apps) are licensed under the 2-Clause BSD License to allow development of closed source apps that can work with the FSenSync.

2 Components and requirements

The main components you need for running the FSenSync include a computer for running the server, Android devices for running the apps, a router for WLAN access, and possible Bluetooth enabled external sensors.

The server has been tested to run well on Linux and Mac. On Windows computers, the server does run, but there can be some issues related to performance and external Python scripts. The server is recommended to be run on Java Runtime Environment (JRE) version 8. The server has low requirements for computing power and memory. Requirements for disk space vary depending on the type of recordings you make. For example, recorded accelerations need only a few megabytes per hour, but recorded videos can fill many gigabytes.

The most important requirement for the computer is the stability of the clock. FSenSync is intended for recordings with millisecond synchronization between recordings, and often the default settings of a computer involve adjustments to the clock that can mess up timings. Possible reasons for the jumps in time flow include the computer going into sleep mode and adjusting the time after waking up, the user setting a new time, switching to daylight saving time, and automatic syncing with a network time server. Jumps in the server time are seldom an issue after the computer has been awake for more than a minute. However, if they do happen, the server displays a warning message, and stops syncing the time for any apps that are currently recording data.

The WLAN router has a great affect on the performance of the FSenSync. The clock synchronization requires that at least some network packages can be sent with a minimal delay (under 5ms). It works best if the WLAN is exclusively reserved for FSenSync. Another way to improve the performance is to not have Internet access on the WLAN as Android devices may start downloading large updates at random times. Currently, the initial device discovery is built on broadcast packages, which are usually relayed only to devices connected directly to the same router.

The router does not have to be an external device. Instead, it is possible to use the computer or one of the Android devices as a router. This may limit the number devices that can connect simultaneously, but may give lower latencies than even expensive external routers. Some routers can cause large delays even when there is very low amount of traffic in the network. Another cause for large latencies can be power saving features on the Android devices. For example, some devices start delaying part of the network packages when their screen is closed.

The Android devices are the last major component. It is best to use Android version 6 or later. There is a lot of variation between devices from different manufacturers, so all apps cannot be guaranteed to run perfectly on all devices. See the FSenSync Device Test Table for more information on hardware support.

3 Server

The FSenSync Server is used for controlling the FSenSync apps and desktop programs.

Linux, Mac and Window can be used to run the server, but on Windows the performance can vary. Java Runtime Environment (JRE) version 8 is recommended. Start the server by clicking on the exec_Server/FSenSyncServer.jar, or from command-line use: java -jar FSenSyncServer.jar

The server settings are stored in text file ‘settings.txt’ next to the server executable. The first line of the file is the default experiments folder, which is the folder where the server creates the individual experiment directories. The folder can also be selected from the server start up window. The second line is the font scaling. The default value is 1.2. Selecting a larger scaling value may be needed when using a screen with a very small pixels. The server needs to be restarted before new settings take effect. The third line is the path to the Anaconda bin folder which should contain Python executable and FFmpeg.

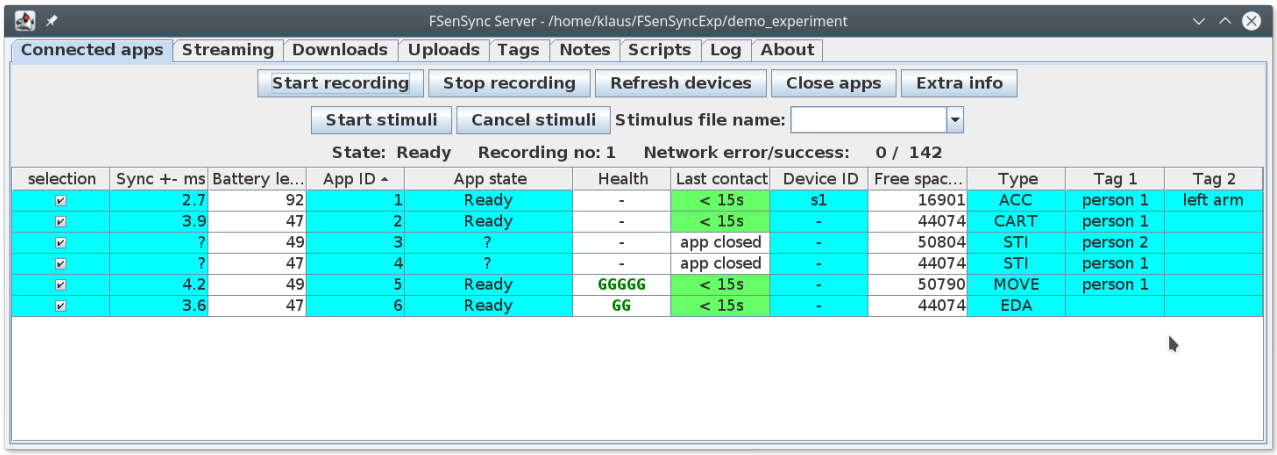

Figure 1: The connected app stab

The ‘Connected apps’ tab allows starting/stopping recordings, refreshing devices (needed if the network is changed), closing apps, and starting/canceling stimuli. The commands are repeated up to five times to the selected apps, and then the server gives up. You can press the buttons several times as the apps ignore repeated commands.

The ‘Connected apps’ tab shows information such as sync accuracy in milliseconds and the app state. Possible state include: ‘Initial sync’, ‘Waiting for sensors’, ‘Ready’, ‘Recording’, ‘Preparing play’, ‘Playing’, ‘File transfer’, and ‘Post sync’. State ‘Ready’ means that the app is synced and can start recording. The ‘Extra info’ button shows further details that can be useful if there are network problems or you are streaming data in real-time.

The ‘Stimulus file name’ combo box suggests files from the previously uploaded batch.

The ‘Health’ column shows the state of the sensors connected to the Move and the Eda apps. ‘B’ means bad streaming or slow start of streaming, ‘G’ means good streaming, and ‘I’ means initial state while connecting to the sensor.

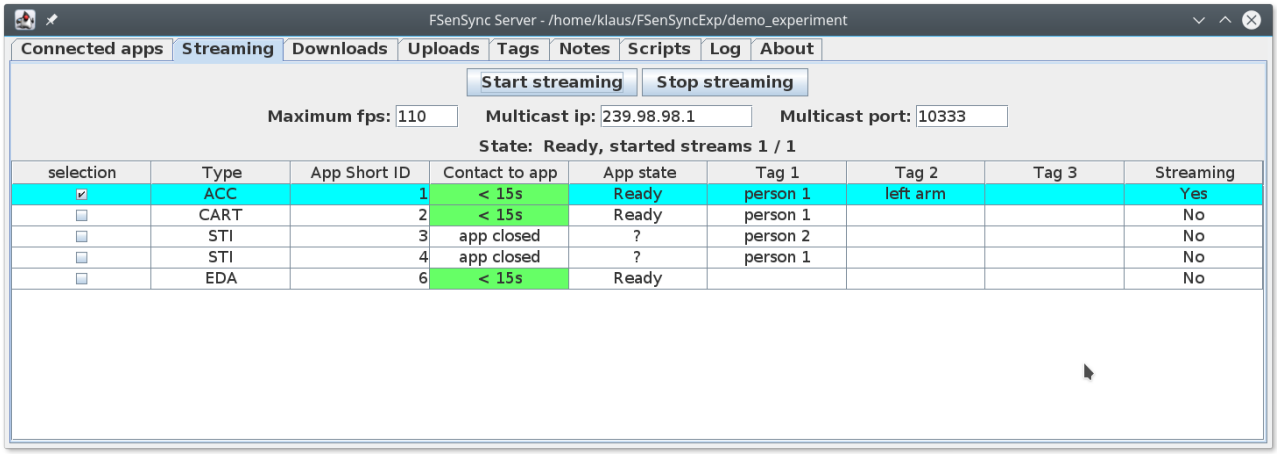

Figure 2:The Streaming tab

The ‘Streaming’ tab allows starting multicast streaming over LAN for apps that support it. Examples of receiving and visualizing the streams are included as Processing sketches. Streaming is currently supported by the Acceleration, Stimulus, Cartography, Eda, VPro, and BioHarness apps.

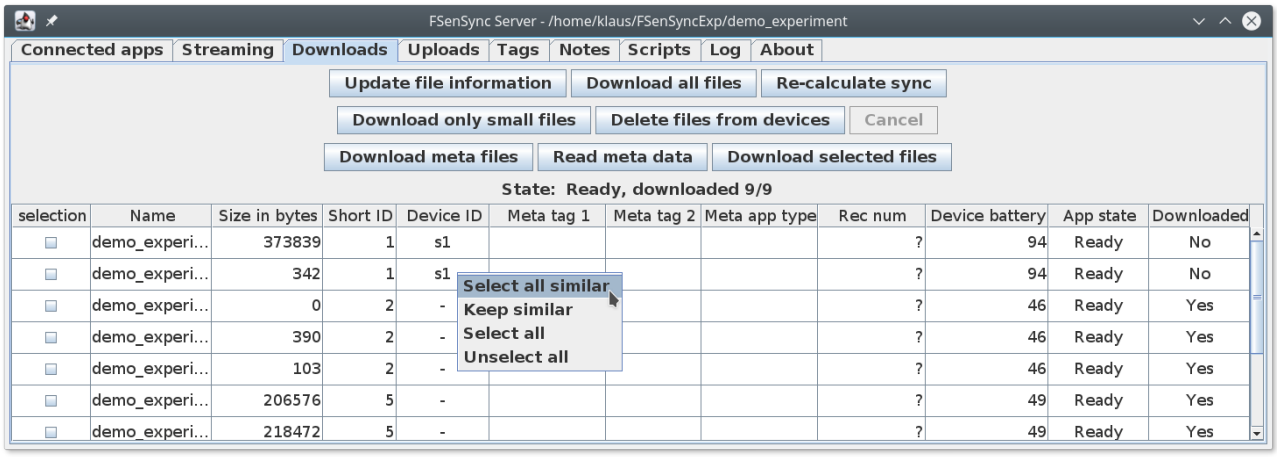

Figure 3:The Downloads tab

The ‘Downloads’ tab allows downloading and calculating the final sync of the recorded files. In most cases, just pressing ‘Download all files’ is enough. The downloaded files are saved into the ‘downloaded’ folder, and the versions with the final sync are put to the ‘synced’ folder. For any analysis use the files in the ‘synced’ folder.

The ‘Downloads’ tab includes possibility to select files to be downloaded for example based on tags by using the right click menu. To show the tags, first download only the meta files, and then click ‘Read meta data’.

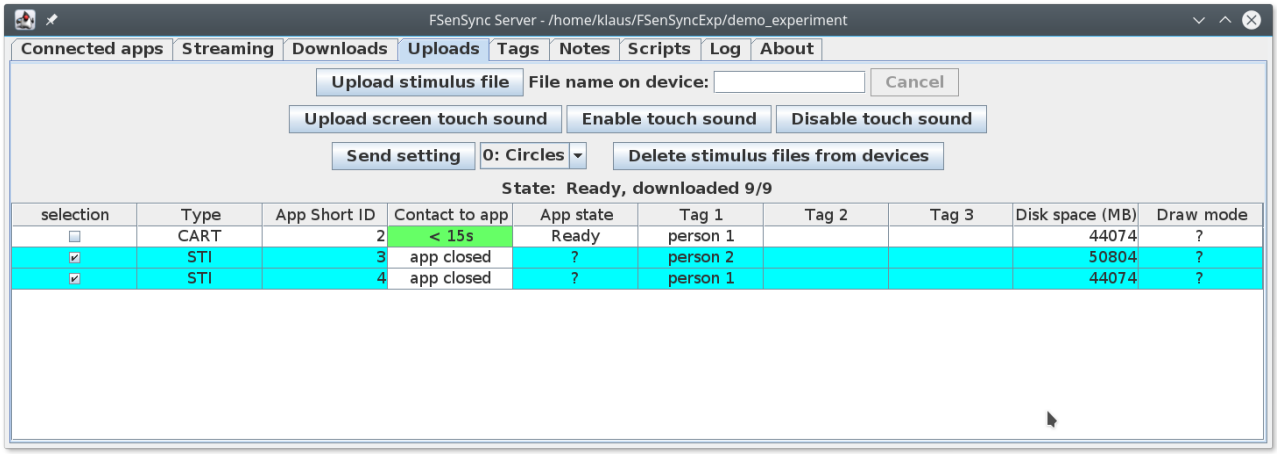

Figure 4: The Uploads tab

The ‘Uploads’ tab allows uploading stimulus files to the Stimulus, the Video Annotation, and the Cartography apps. Example files are provided in folder ‘setting_and_app_file_examples’. You can upload multiple files from a folder by selecting all desired files in the file selection window.

The ‘File name on device’ allows renaming a file that is uploaded to an app. This can be used for starting a different file on different Stimulus apps at the same time. This easily leads to terrible confusions, and the functionality is hopefully replaced in the future with a proper stimulus scheduler.

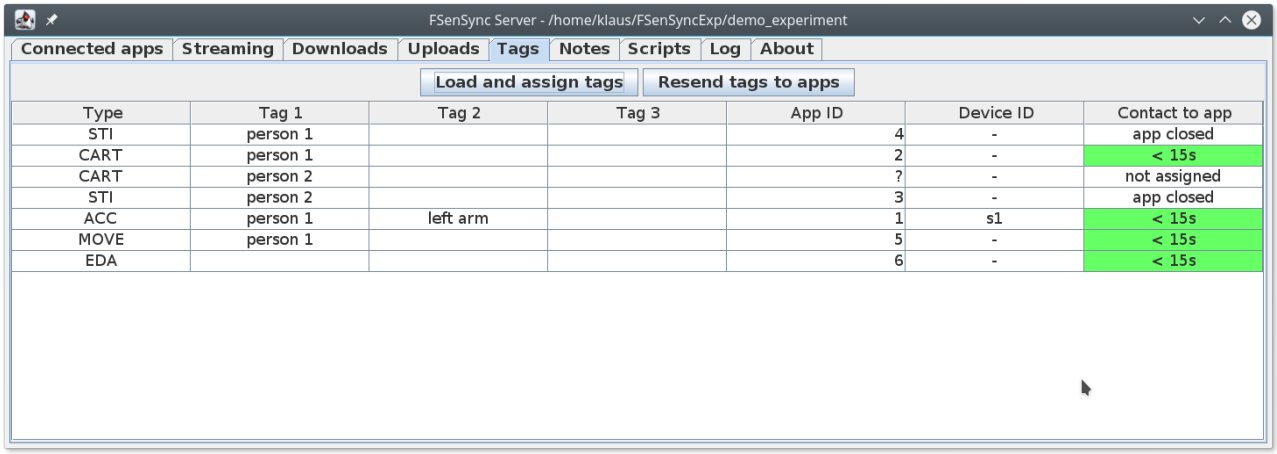

Figure 5: The Tags tab

The ‘Tags’ tab allows loading tag files which assign tag to the apps. The tags are put into the meta files that are created for every recording.

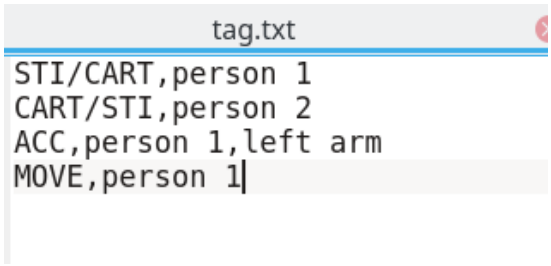

Figure 6: An example tag file

The tags can be shared by apps running on the same device. If shared tags are used they should be put before any other tags into the tag file.

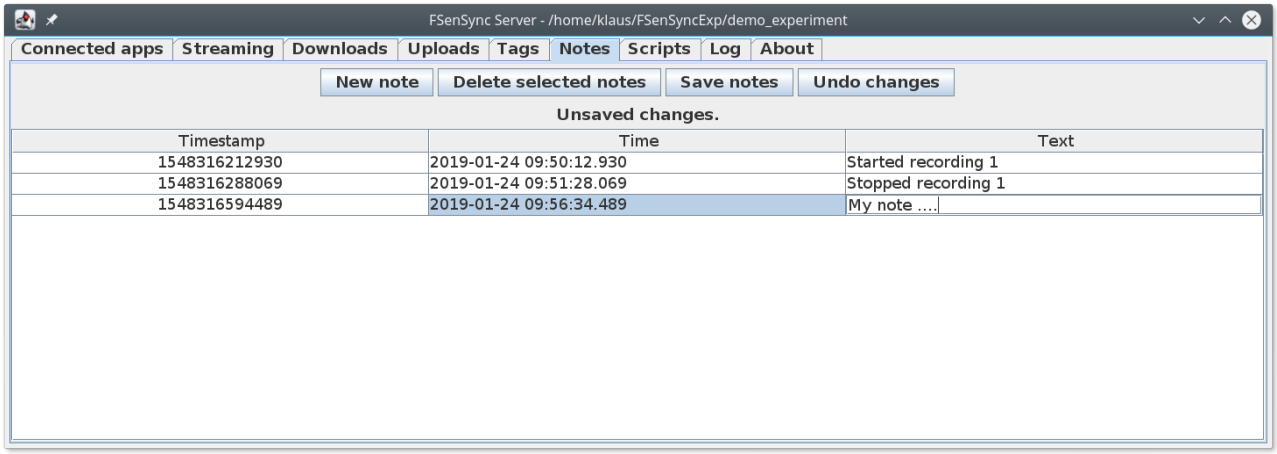

Figure 7:Notes tab

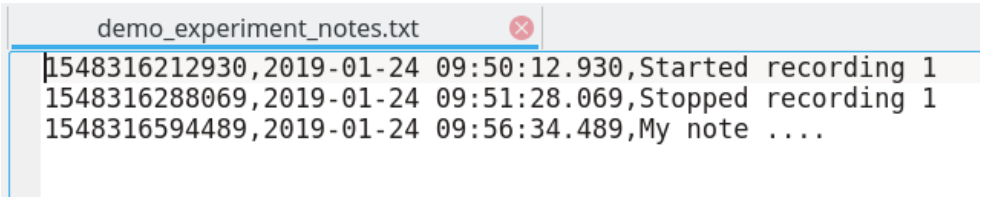

Figure 8: Note file corresponding to the notes shown in Fig. 7

The ‘Notes’ tab shows the notes created by starting and stopping recordings. New notes can be created, and old ones edited. The notes are timestamped in the same timeline as all the sensor data. The notes can be found from ‘[experiment_name]_notes.txt’ file. The notes are backed up automatically at the start of each recording.

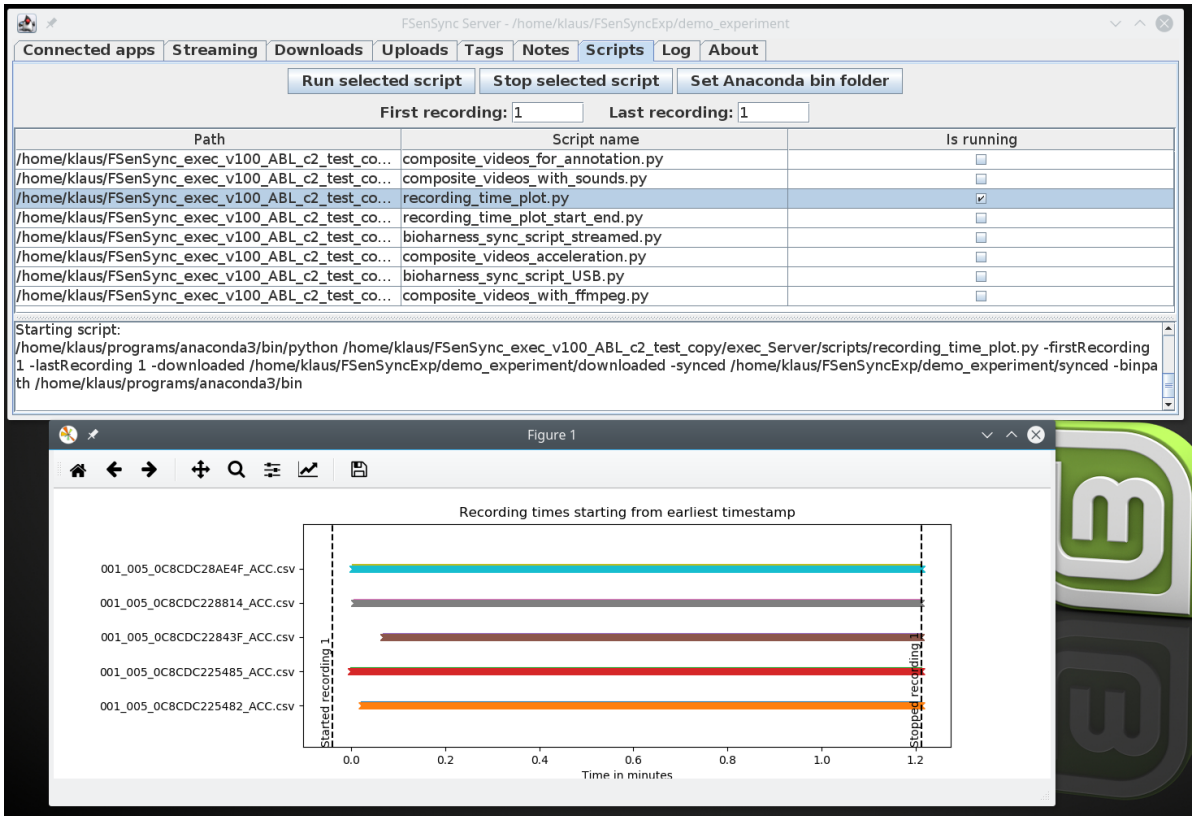

Figure 9: The Scripts tab and a launched visualization

The ‘Scripts’ tab allows running Python script if the Anaconda (https://www.anaconda.com/download) has been installed and the bin path is set. The final sync and compositing of videos are done with Python scripts. The Python scripts are not fully supported on Windows.

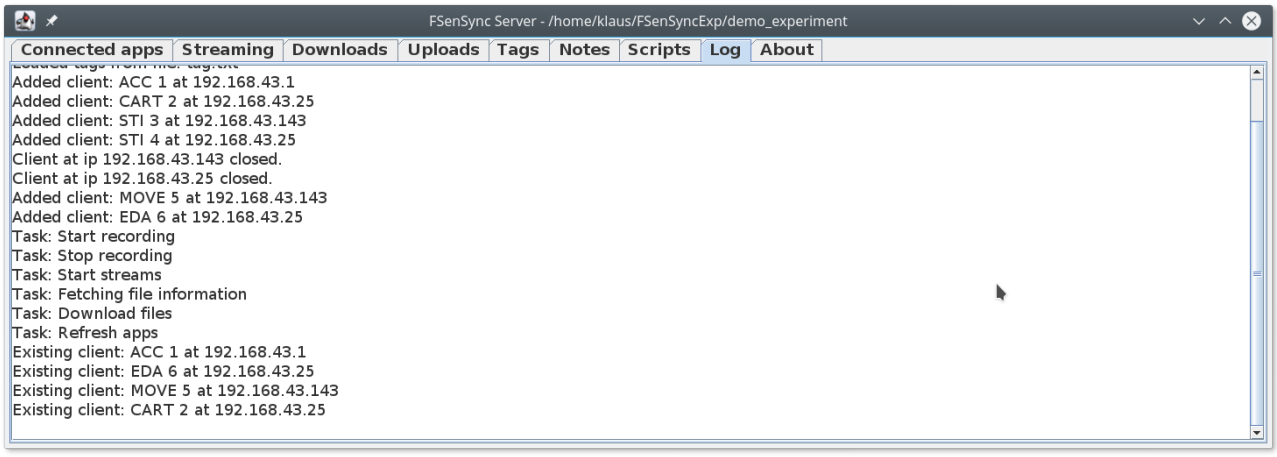

Figure 10: The Log tab

The ‘Log’ tab shows all major event and errors which can come from the server itself or be sent from an app. The same information is also saved to the file ‘debugLog.txt’. Note that the log may contain network related error messages even when everything runs smoothly.

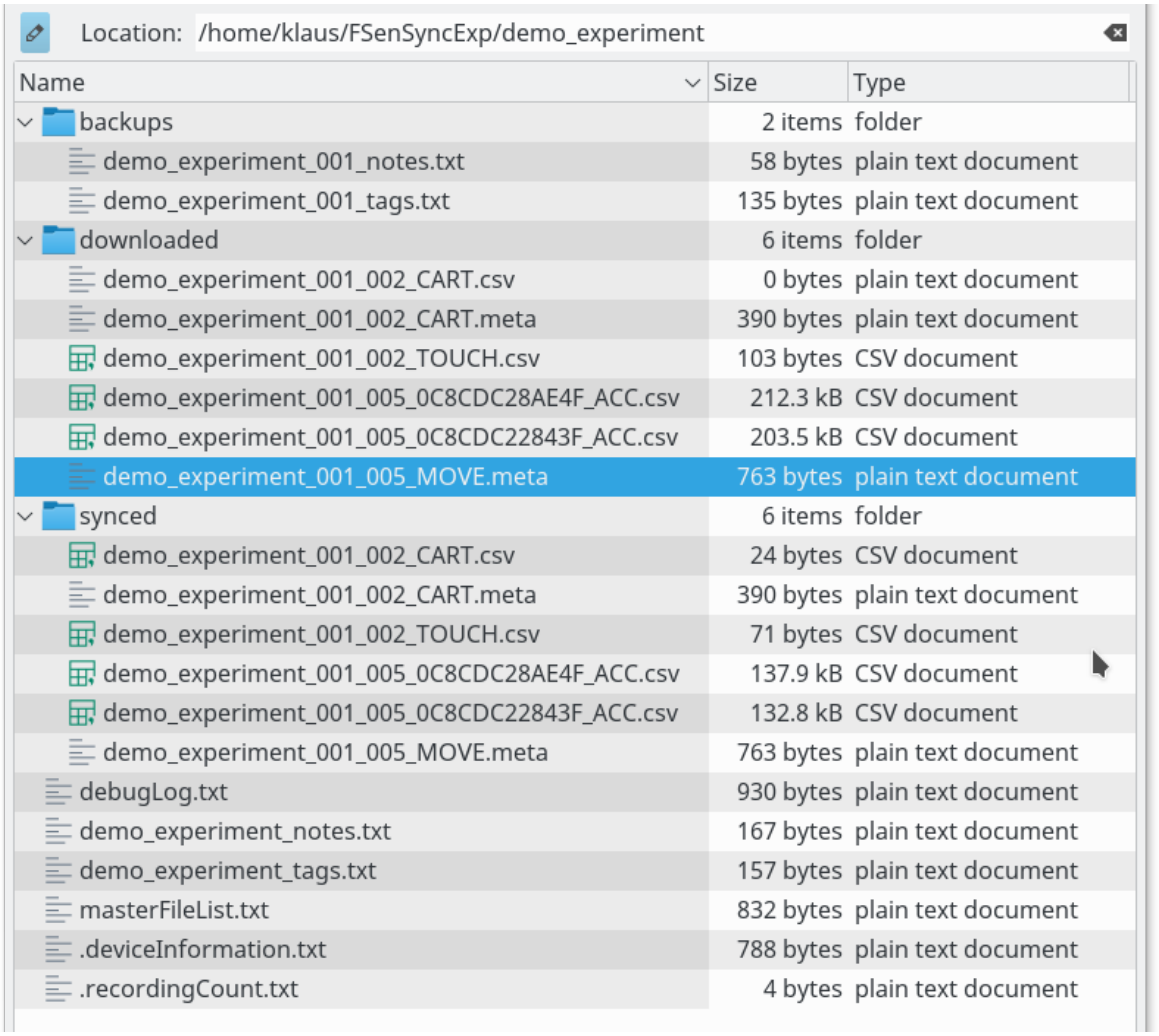

Figure 11: Folder structure of the experiments

The files and folders inside the experiment folder include:

- backups/ – The backups of note and tag files
- downloaded/ – The data files downloaded from the apps. These versions contain the clock drift information that is needed in calculating the final sync. It is best not to edit these as then you might lose data.
- synced/ – The data files with final sync done. Use these files in analysis of the data. The files in this folder can be safely edited as they can be recreated from the files in the 'downloaded folder' by pressing 'Re-calculate sync' button.
- uploadBackups/ – Copies of files uploaded to the apps. Files with identical names are overwritten by the latest uploaded version. These may be needed when analyzing experiments done with the Stimulus/Video Annotation/Cartography apps.
- debugLog.txt – Error messages and major events during the recordings
- [experiment_name]_notes.txt – Timestamped notes added by the user and the server

- [experiment_name]_tags.txt – The most recent version of tags for the apps
- masterFileList.txt – List of data files known to have existed on the apps. Do NOT edit this.
- .deviceInformation.txt – Ids of the connected app. Do NOT edit this.
- .recordingCount.txt – File that stored the latest recording number. Do NOT edit this.

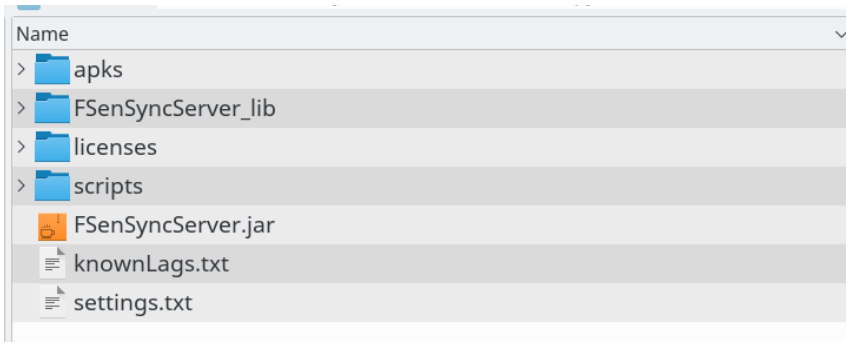

Figure 12: Folder structure of the server folder

The server folder (exec_Server) contains:

- apks/ – The installation files of the apps. You can add your own apks here.
- FSenSyncServerLibs/ – Libraries needed by the server
- licenses/ – License and notice files for all the Android apps and the server
- scripts/ – Python scripts that can be launched from the server. You can add you own scripts here.
- FSenSyncServer.jar – The server executable
- knownLags.txt – Sensor lag information that is used in calculating the final sync. You can add your own devices to the list. Currently used by the Acceleration app.
- settings.txt – Contains path to the main experiment folder, server user interface multiplier, and path to the Anaconda bin folder

The server allows automatic clock synchronization between the apps. The sync error between the clocks is usually just less than 5 milliseconds, and in principle even sub-millisecond accuracy could be achieved after correcting clock drifts. However, this does not mean that all the data would be synced as accurately. Each sensor and output channel has their own lags and timing variations which add to the total error. See information about those in sections of the apps.

4 Apps

4.1 Common to all apps

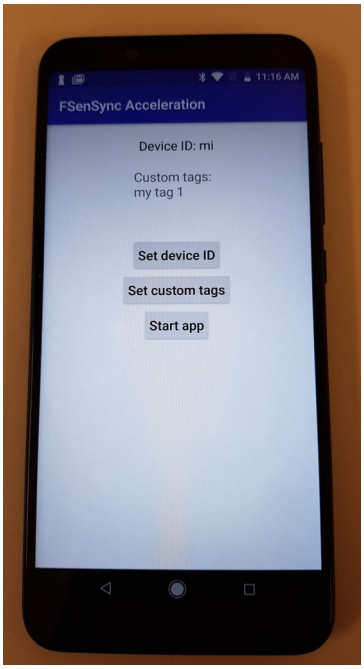

Figure 13: Start up screen

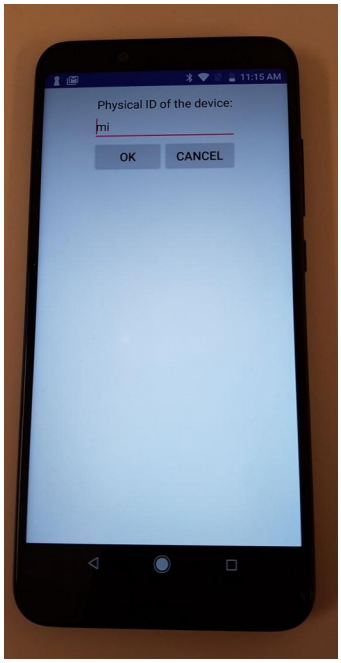

Figure 14:Physical id

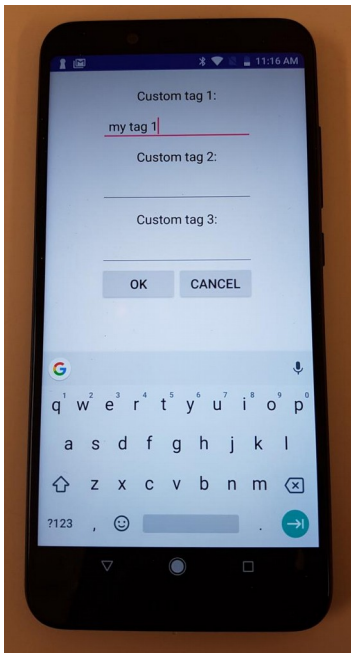

Figure 15: Custom tags

All the apps have the same start up screen. Here you can set the physical id string of the device that is shared between all the apps run on the same device. The string is visible on the server and on the screens of the apps. Custom tags can also be assigned to the apps. These tags work in the same way as the tags you can give on the server using a text file.

All the apps create their own data files and also a meta file that contains extra information about the recording such as the tags, and for example size of the view in the Stimulus app.

4.2 Installer app

The preferred way to install FSenSync apps on the Android devices is the Installer app. To use it you need to copy the Installer apk to all the devices, start the server into an experiment, start the installer app on the devices and press "Start downloads". The devices and the computer running the server should be connected to the same WLAN for the downloads to work. The installer downloads all apk files from the "apks" folder that is next to the server executable. After downloading, the installer checks if the apks are newer then what is already installed, and starts the installations for the new ones. The user needs to accept all the new installations. The installer prints the status of each of the apps at the end of the process.

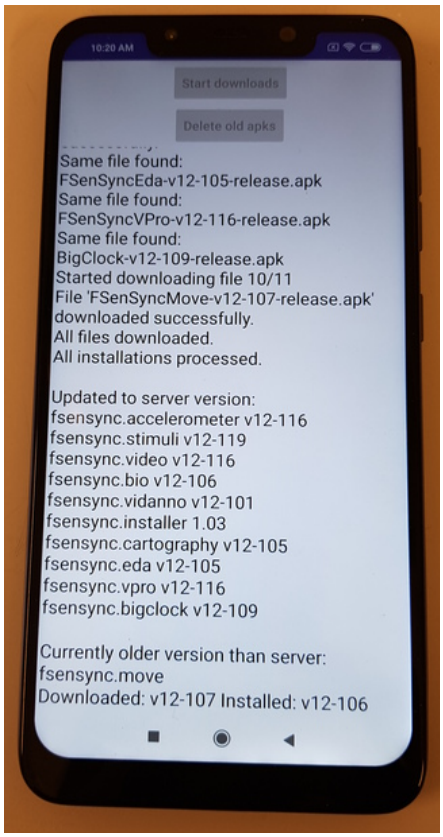

Figure 16: The installer app

If Installer app needs to update itself, you will need to run the Installer app twice for the whole process to finish. It is also possible to copy and install all the apks to the devices manually. The apks compiled at Förger Analytics are signed, thus if you develop your own versions of the apps, you may need to manually uninstall the old versions before a new version using your signing keys can be installed.

4.3 Acceleration (ACC)

The Acceleration app records acceleration from the sensor inside in the Android devices. The app tries to get acceleration at 100 samples per second, but some devices give a lower rate or fake the rate by repeating the same value many times. See the FSenSync Device Test Table for more information on hardware support.

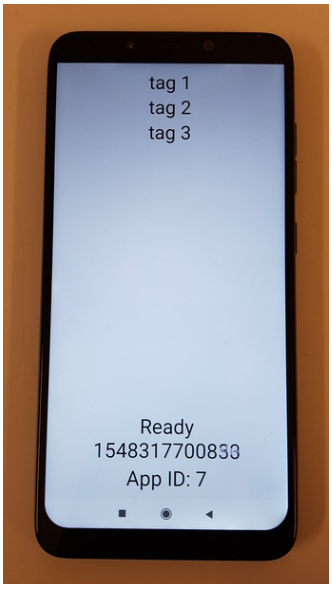

Figure 17: The Acceleration app

The data from the app comes in ACC.csv file that contains:

- Timestamp(milliseconds) – Synced server time
- Xacceleration – Acceleration on the X axis in meters/second^2
- Yacceleration – Acceleration on the Y axis in meters/second^2
- Zacceleration – Acceleration on the Z axis in meters/second^2

The final versions of the data files include a correction for the latency that is read from the ‘knownLags.txt’ from the server folder. The file contains lines with comma separated values such as ‘samsung_GT-N8000,ACC,10’. There the first value is the device type which is the same as in the meta files on line starting with ‘Device:’. The second value is type of the app which is currently always ‘ACC’. The third value is the lag of the accelerations in milliseconds that has been measured in calibration experiments.

The known lag is also taken into account in the timestamps of data streamed over OSC protocol. You can find an example of receiving the streamed data in the ‘FSS_visualization_acceleration’ Processing sketch.

4.4 BigClock (CLOCK)

The BigClock app displays the synced server time on the screen. Below the timestamp, there is a second counter, and a decisecond counter. The app can be used for rough syncing of video cameras that do not have FSenSync integration. Note that the visualization has additional lag of a few tens of milliseconds caused by the lag and the refresh rate screen of the screen. The app does not produce data files.

Figure 18: The BigClock app

4.5 Eda (EDA)

The Eda app allows synchronizing data from the Moodmetric rings (http://www.moodmetric.com/). The ‘Stop/Start scan’ button allows finding rings with a Bluetooth scan. The ‘Select rings’ allows selecting rings that should be connected to the device. The app allows giving individual tags to rings attached to it. The tags are saved in the meta files of the recordings.

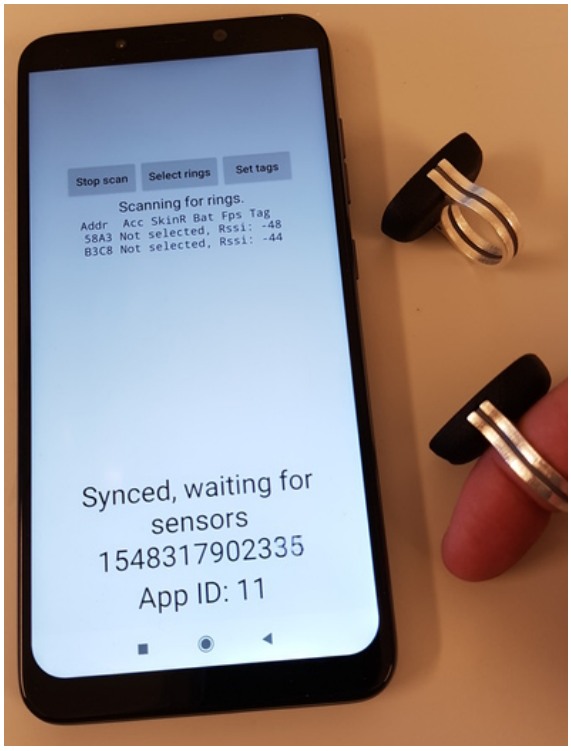

Figure 19: The Eda and and rings

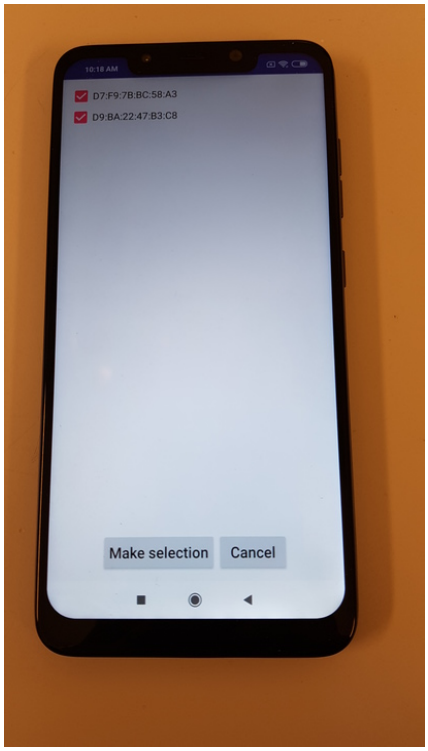

Figure 20: Ring selection

The data file name from the Eda app includes the Mac address as in ‘experiment_001_003_D9BA2247B3C8_EDA.csv’. The file contains:

- Timestamp(milliseconds)
- Status – Status value provided by the ring
- MoodNumber – The long term value used by the Moodmetric app
- SkinResistance – The current skin resistance
- Xacceleration – Acceleration on the X axis in meters/second^2
- Yacceleration – Acceleration on the Y axis in meters/second^2
- Zacceleration – Acceleration on the Z axis in meters/second^2

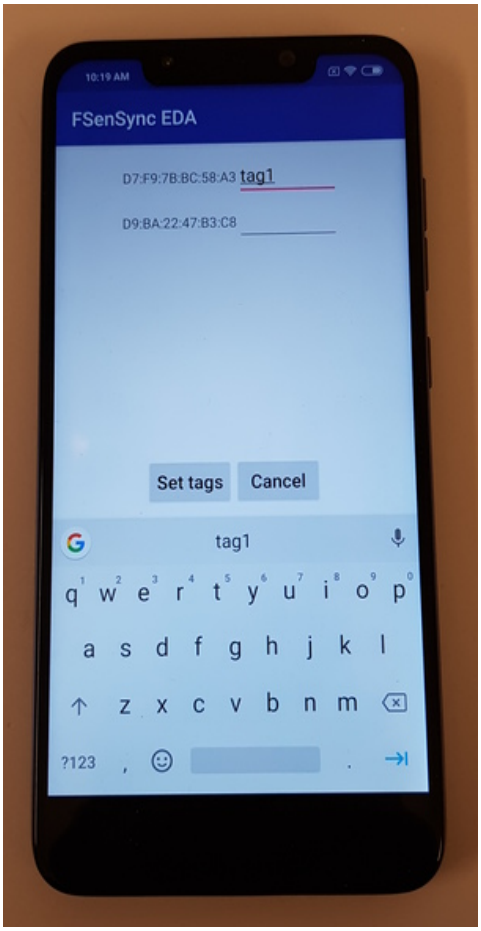

Figure 22: Tags to individual rings

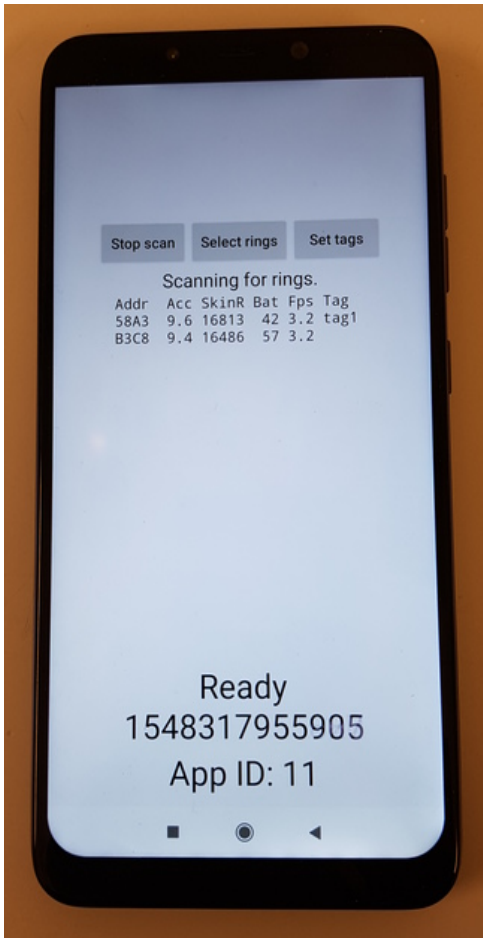

Figure 21: Tags visible on the screen

The Eda gives data at the rate of three samples per second. The synchronization accuracy of the app may vary as the sync is based on logging the arrival times of the data packets instead of a time query/response protocol. Thus, the time between samples can vary, and samples may also be lost due to a bad Bluetooth connection. To get data reliably, the rings should be no more than one or two meters from the Android device.

The Eda app supports OSC streaming of the data. See ‘FSS_visualization_eda’ Processing sketch for an example of receiving the data.

4.6 Stimuli (STI)

The Stimulus app allows playing sound, image, and video stimuli on Android devices. The app also records screen touches. The stimulus files must first be uploaded to the device on the ‘Uploads’ tab on the Server. Next, a file can be selected on the ‘Connected apps’ tab, and the the apps will play it three seconds after the ‘Start stimuli’ button has been pressed. Several files can be played in succesion during one recording, and the play progress of those will be logged in the same file. To separate the logs of different plays, the recording must be stopped between the plays.

Supported sound file formats include flac, wav, mp3, and acc. Supported image file formats include png, jpg, bmp, and gif. Supported video file formats include mp4, 3gp, webm, and mkv. To play a sound and simultaneously show a background image, a file list can be used as in the example ‘setting_and_app_file_examples/stimulus_examples/stimulus_file_list.txt’. The images are stretched to fit the whole view area, and videos have the maximum size while preserving the aspect ratio. To get full screen videos, use the same resolution that is shown on the screen of the app.

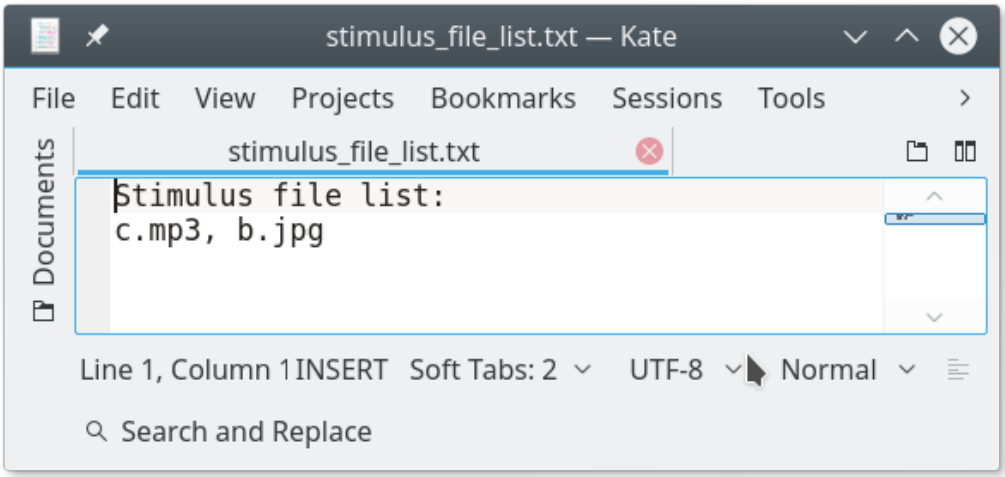

Figure 23: A example of a file list that allows playing a sound while simultaneously showing an image

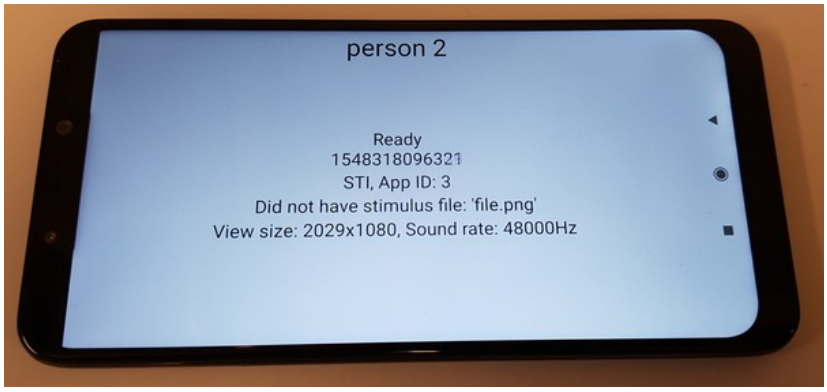

Figure 24: The Stimulus app showing an error

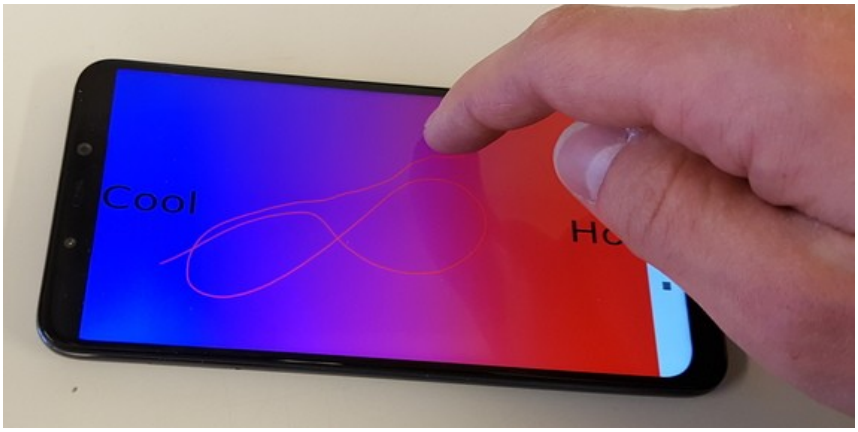

Figure 25:Stimulus app when ‘Traces’ setting has been sent from the ‘Uploads’ tab

The Stimulus app produces the following files:

- meta file
  - View width – Width of the view area in pixels
  - View height – Height of the view area in pixels
- STI.csv – Log of the play progress
  - Timestamp(milliseconds) – Synced server time
  - EventType – Can be ‘started, ‘playing’, ‘stopped’, or ‘cancelled’
  - PlayProgress(milliseconds) – The progress of the in milliseconds from the start
  - FileName – Name of the stimulus file
  - ImageName – Name of the background image
- TOUCH.csv – Screen touches during the recordings
  - Timestamp(milliseconds) – Synced server time
  - TouchIndex – For multitouch: ‘0’ for the first finger to touch the screen, ‘1’ for the second, etc.
  - Xcoordinate – X coordinate in pixels
  - Ycoordinate – Y coordinate in pixels
  - UpDownEvent – ‘-1’ for finger down, ‘0’ for finger move, ‘1’ for finger up

While the Stimulus app tries to start the play at the same time on all the devices, there can be up to 100 millisecond offsets in starts of the play. Also, the latencies of touch screens may vary and they

are not currently compensated in the recordings. Thus, to compare the touch times as accurately as possible, all the participant should use the same model of the Android device.

The touch data can be streamed over OSC. See ‘FSS_visualization_touches’ Processing sketch for an example of receiving the data.

When analyzing the data, the best approximation of the play progress can be achieved by taking the PlayProgress values, removing the values from the first and the last second, fitting a line to the data, and by interpolating the play progress values for the needed timestamps. This usually gives the best results as the start and the end of the logging may have inaccurate values due to, for example, a stuttering start of play. Also, the logged values may have a few milliseconds of noise caused by the way the Android MediaPlayer reports the play progress.

4.7 Cartography (CART)

The Cartography app allows creating questionnaires where the participants move images on the screen of an Android device. The idea is that participants are instructed to use the absolute or the relative positions of the images in conveying information.

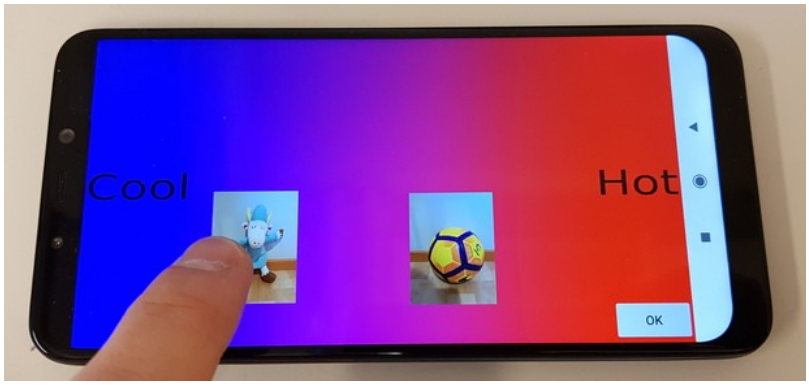

Figure 26: Cartography app running the 'Cool or Hot' experiment with images of toys

The input images for the Cartography are uploaded from the 'Uploads' tab of the Server in the same way as the files for the Stimulus app. The experiments are begun by starting a Cartography definition file using the 'Start stimuli' button in the 'Connected apps' tab of the Server. See example files in folder 'setting_and_app_file_examples/cartography_example'.

A Cartography definition file is a CSV file containing on each line the data for one trial:

- BackgroundImage – The name of the backgroud image file
- OverlapAllowed – true/false, whether images can overlap
- SelfEnded – true/false, whether the trial is ended by pressing 'OK' button
- MultiTouch – true/false, whether multitouch is allowed
- TrialDuration – Duration of the trial in milliseconds, if it is not self ended
- BreakBeforeTrial – Duration of the pause between the trials in milliseconds. Use 1000 to have enough time to load the images of the next trial
- ImgScale – Scale of the images. 1.0 mean 100% of the screen size, 0.1 means 10% of the screen, etc.
- Img0... – Name of the image file. Tens of image files can be added

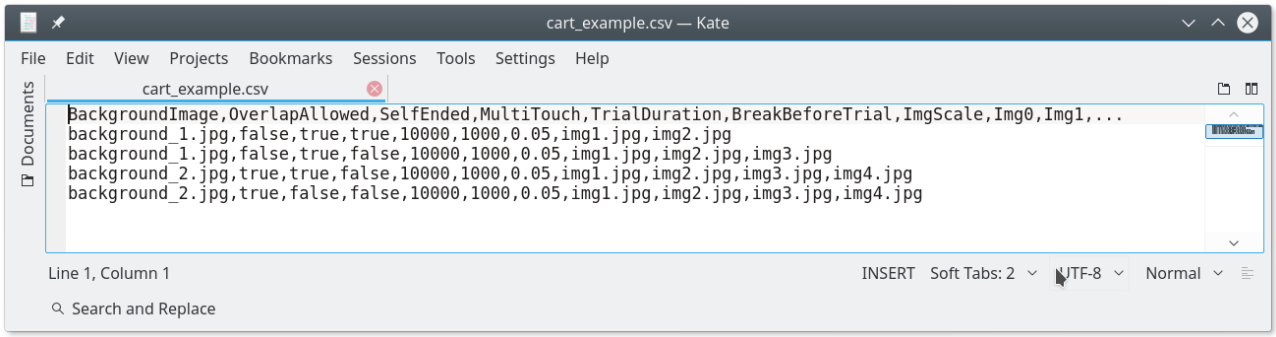

Figure 27: An example Cartography definition file

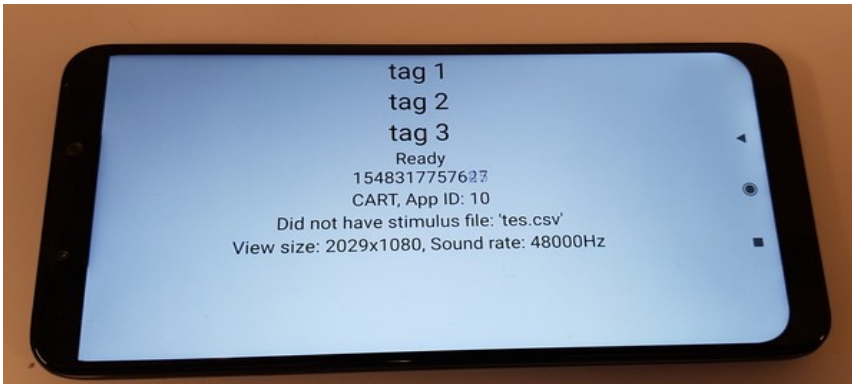

Figure 28: Cartography app showing an error message

The Cartography app produces the following files:

- meta file
  - View width – Width of the view area in pixels
  - View height – Height of the viewarea in pixels
- TOUCH.csv – Screen touches during the recordings
  - Timestamp(milliseconds) – Synced server time
  - TouchIndex – For multitouch: ‘0’ for the first finger to touch the screen, ‘1’ for the second, etc.
  - Xcoordinate – X coordinate in pixels
  - Ycoordinate – Y coordinate in pixels
  - UpDownEvent – ‘-1’ for finger down, ‘0’ for finger move, ‘1’ for finger up

- CART.csv
  - Timestamp(milliseconds) – Synced server time
  - ImgIndex – Number of the image on the line in the Cartography definition file
  - CenterX – X coordinate for the center of image after a move or initial/ending position.
  - CenterY – Y coordinate for the center of image after a move or initial/ending position.
  - Width – Image width on the screen in pixels
  - Height – Image height on the screen in pixels
  - TouchIndex – This is meant for finding the corresponding touch data from the TOUCH.csv file when multitouch is enabled. The TouchIndex in the CART file is the same as in the TouchIndex in the TOUCH file. For the TrialPhase 0 and 1, the TouchIndex is always -1 as the initial and final positions of the images do not have corresponding touch data.
  - TrialPhase – 0 means initial randomly assigned positions. 1 means a new position after user has moved an image. 2 means final positions after all the moves when the trial has ended.
  - TrialNum – This correspond to the line number in the original definition file. The first trial is number 0.

The latencies of touch screens may vary, and they are not currently compensated in the recordings. Thus, to compare the touch times as accurately as possible, all the participant should use the same model of the Android device.

4.8 Video (VID)

The Video app allows recording videos that are synced with +-1 frame accuracy with all other FSenSync data. The final synchronization and compositing is done with the Python scripts ‘composite_videos_with_ffmpeg.py’ and ‘composite_videos_with_sounds.py’. See also the example ‘composite_videos_acceleration.py’ that combines accelerometer visualization with videos.

The frame rate allowed by the devices varies. See the FSenSync Device Test Table for more information on hardware support.

Figure 29: The Video app

The Video produces VID.csv files containing timing information, and mp4 files containing the video data. The Python scripts produce mp4 files stored in the ‘synced’ folder that have the starting timestamp in the file name.

Most Android devices have a 4 gigabyte file size limit. This means that recording very long videos is not possible in one take. It can be practical to stop the video recording when the data is not needed as transferring large files may be slow, and Android devices have limited amount of storage space. The fastest way to transfer large files is usually to download them over a 5GHz Wifi.

4.9 Video Annotation (VANNO)

The Video Annotation app allows participants to play videos and annotate them in the same style as in the Stimulus app with a background image. The files are uploaded from the Server in the same way as in the Stimulus app.

Examples of Video Annotation definition files can be found from folder ‘setting_and_app_file_examples/video_annotation_examples’.

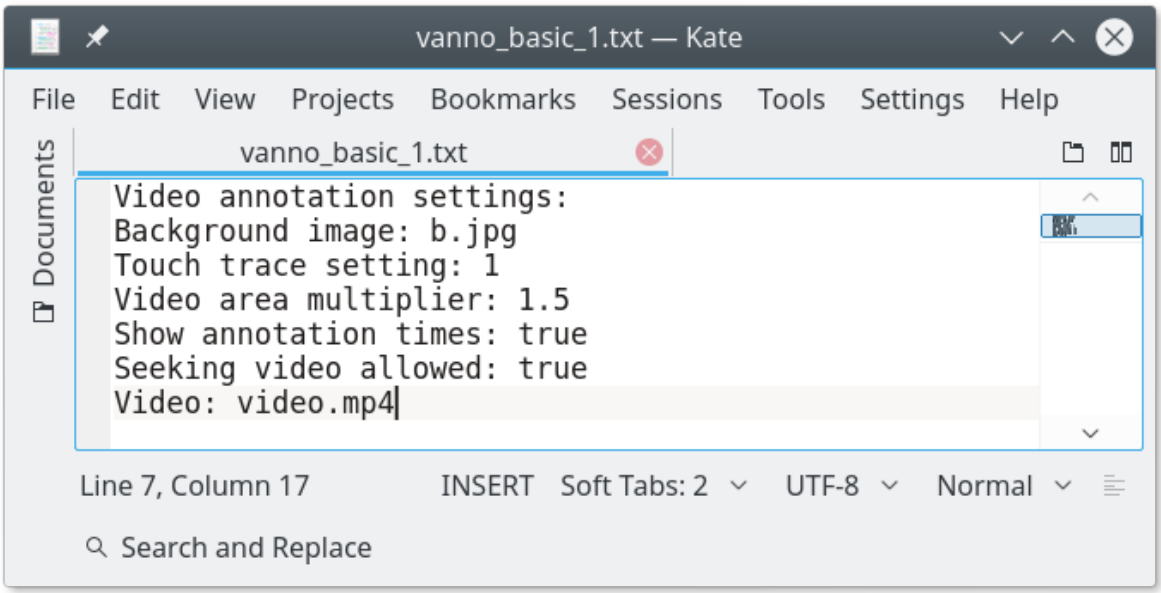

Figure 30: Example of a basic video annotation definition file

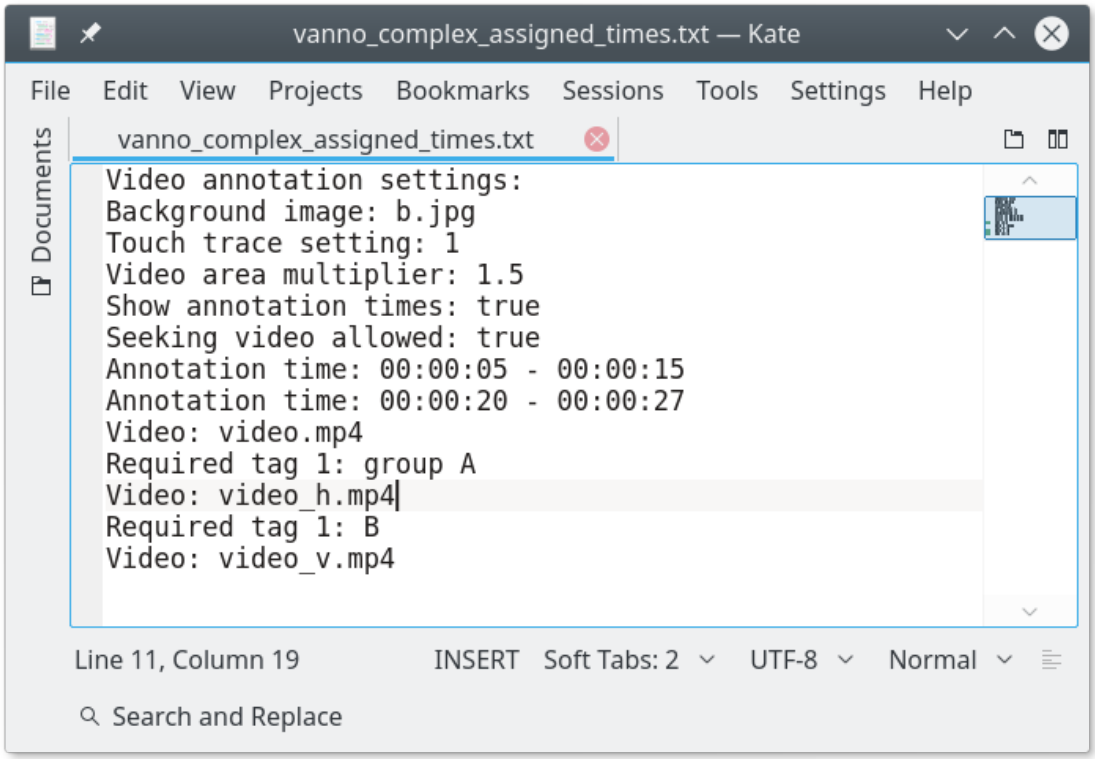

Figure 31: An example of a complex video annotation definition file with suggested annotation times, and video chosen with tags of the apps

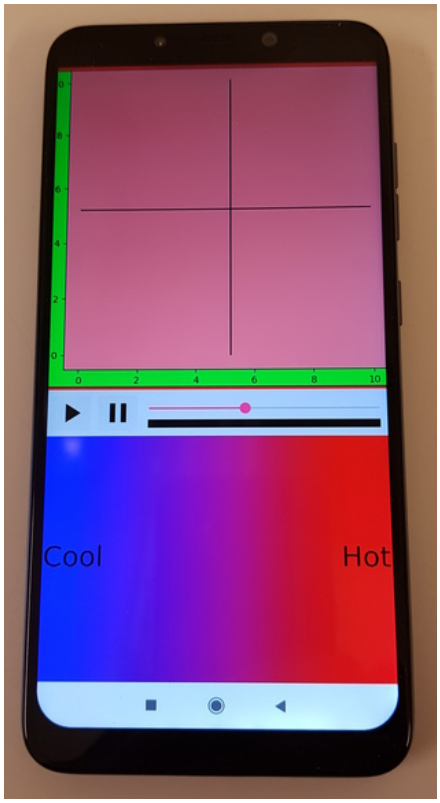

Figure 33: Video Annotation app

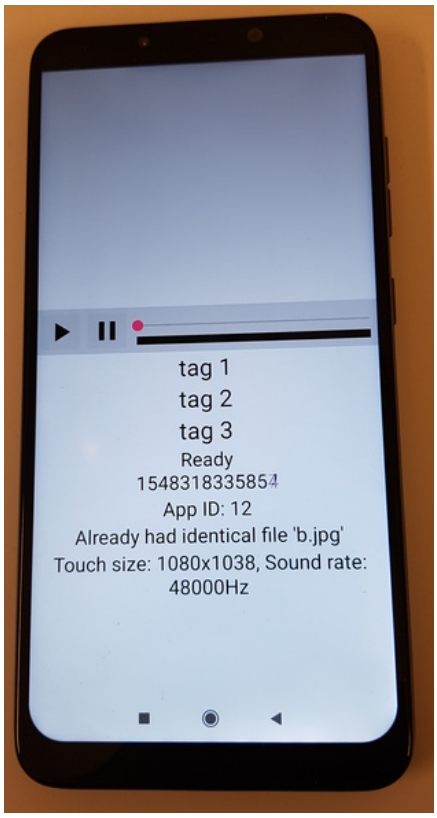

Figure 32: Video Annotation app showing an error

4.10 Serial Reader (SERIAL)

The Serial Reader app runs on desktop computers, and allows recording triggers sent over serial port. To find the correct port easily, first start the app, and after that connect the USB-to-serial cable. Windows, Linux and Mac computers are supported by the app. USB-to-serial cables may require installing drivers manually at least on Macs.

The settings supported by the app are: Rate: 9600 bauds, Parity: None, Data bits: 8, Stop bits: 1

As the app receives single bytes, all the settings do not have an actual effect.

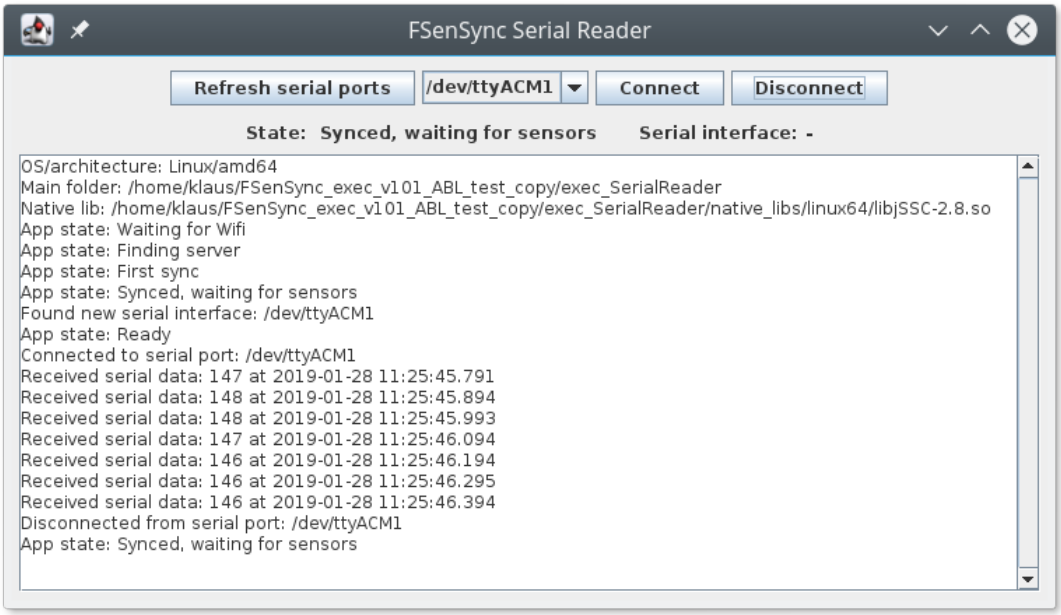

Figure 34: The Serial Reader app

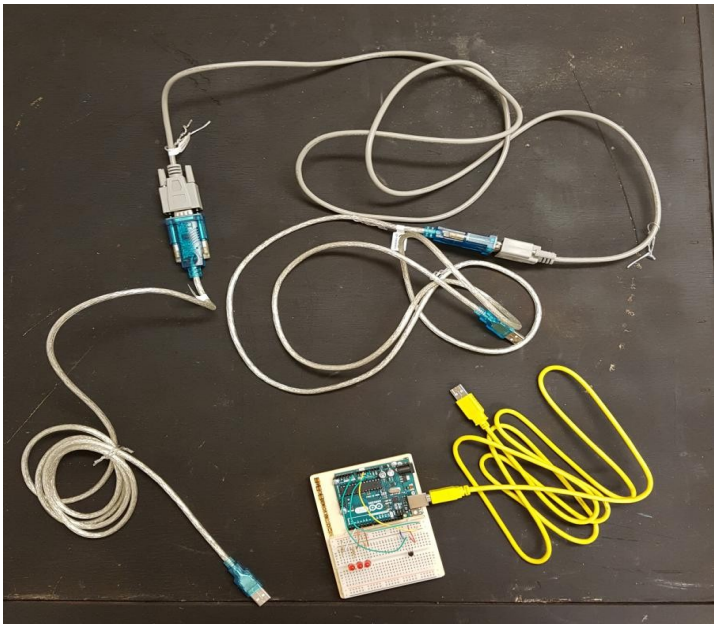

Figure 35: The Serial Reader can read data for example from an Arduino or another computer over USB-to-Serial -> null modem -> Serial-to-USB cables

The data is stored in a SERIAL.csv file:

- Timestamp(milliseconds) – Synced server time
- SerialEvent – Value of the received byte as an integer

4.11 Desktop Player (DEPLAY)

The Desktop Player app allows playing sound and video stimuli on desktop computers. The use of the app on the Server is similar to the use of the Stimulus app.

The DesktopPlayer app requires native GStreamer libraries that work on Linux simply by installing GStreamer 1.x for your distribution. For Windows and Mac, the libraries should be bundled (in the same way as in the Processing Video library), but they are not currently included due to licensing issues (the source code is not easily available).

The resulting DEPLAY.csv file contains:

- Timestamp(milliseconds) – Synced server time
- EventType – Can be ‘started, ‘playing’, ‘stopped’, or ‘cancelled’
- PlayProgress(milliseconds) – The progress of the in milliseconds from the start
- FileName – Name of the stimulus file

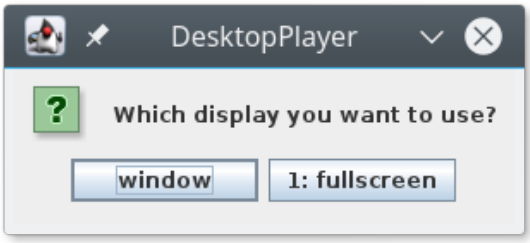

Figure 36: Selection of the view type

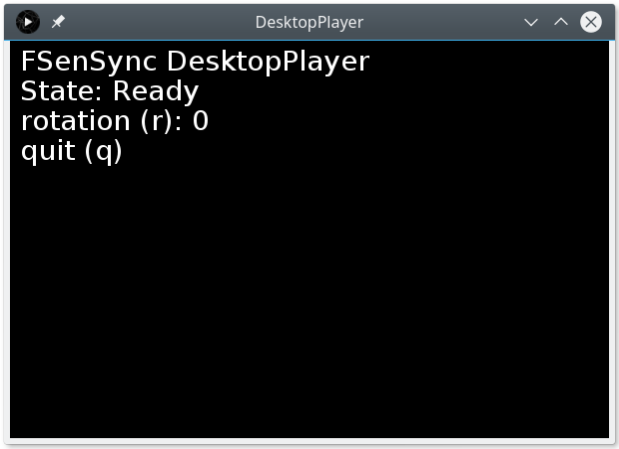

Figure 37: The Desktop Player app

The app may not play all videos with rotation meta information correctly due to GStreamer issues. Use the ‘r’ key on keyboard to fix the rotation.

When analyzing the data, the best approximation of the play progress can be achieved by taking the PlayProgress values, removing the values from the first and the last second, fitting a line to the data, and by interpolating the play progress values for the needed timestamps.

4.12 Synced Processing sketches (PROC)

The FSenSync Desktop Library can be used in Processing sketches for getting synced timestamps, and logging/uploading data files to the FSenSync Server. See examples in ‘sketchbook/FSenSyncRecorderExample’ and ‘sketchbook/SimpleSyncExample’.

The FSenSync Processing sketches also include several examples of how to receive streamed data from the FSenSync apps.