# Documentation of framework modules from: “Automated data population using case-like scenarios for training and validation in mobile forensics”

Zentrale Stelle für Informationstechnik im Sicherheitsbereich (ZITiS)

June 5, 2023

# Contents

# 1 Documentation of the scripts

This document serves as a reference guide for the scripts included in the FORMOBILE.eu repository. Each section explains one main script and may include a short description of the script's purpose, usage of other scripts, preconditions, functionality, and additional information. It is important to read each documentation carefully and ensure that the preconditions are met before proceeding.

To begin, activate the developer options and USB debugging on every smartphone. Then, install the required applications onto the devices (subsection 1.1). Set up all the tokens (subsection 1.2). Finally, load the contacts onto the devices (subsection 1.3). Once these steps are completed, you are free to use all the other scripts.

## 1.1 Applications

The repository provides two different methods for installing applications on the smartphones. The first method is to install applications via the Google Play Store. This ensures that the most recent version of an app is installed and mimics the natural way an app would be installed.

The second method is to install pre-downloaded APK files directly onto the devices via ADB. This method is especially useful when the desired application is not available on the Google Play Store, or when a specific version of an app needs to be installed. Additionally, this method allows for faster installation on multiple devices without requiring an internet connection or a Google account.

### 1.1.1 Installation via Google Play Store

*Script:* **playstore_main.py**

*Preconditions:*

Ensure the smartphone is connected to the internet and signed into a Google account.

The script requires a list of applications for each device that should be installed. This list is stored in `formobile/datasets/apps/apps_{PERSON_TOKEN_NAME}` and should contain the IDs of the apps to be installed.

Verify that the app is not already installed on the device. Some phone vendors may have additional stores (e.g., Galaxy Store for Samsung, AppGallery for Huawei), and the phone may prompt the user to choose a store for installation. Select Google Play and choose 'Always'. Manually define the location of the install button in the config file.

Connect the devices to the PC.

*Functionality:*

The program establishes connections to the devices using Android Debug Bridge (ADB). Once connected, `playstore_main.py` opens the corresponding store pages of the apps using the app IDs in the application list. The script then taps the screen at the location of the install button. This process is repeated for each app in the list.

### 1.1.2 Installation via APKs

*Script:* **apps_main.py**

*Uses:*

- `copy_apps_to_folder.py`

- install_apps.py

*Preconditions:*

Before starting the program, ensure the following:

Download all the required applications as APK files. Store the standard apps, which should be installed on every phone, in formobile/datasets/apps/standard/. Store individual apps, which are not required for every phone, in formobile/datasets/apps/individual_apps/. Define the number of individual apps in config.json.

apps_main.py installs standard applications on every Android device listed in config.json and connected to the PC. The script can also install predefined individual apps, but this step is optional.

Connect the devices to the PC.

*Functionality:*

The program establishes connections to the devices using Android Debug Bridge (ADB). Once connected, install_apks.py installs each standard application on the corresponding devices listed in config.json.

Next, the program checks if any devices require individual apps. If not, the program ends. If individual apps are needed, a directory is created for each device that has a positive number of individual APKs specified in config.json. copy_apps_to_folder.py is then used to randomly select and copy the specified number of individual application files from the individual folder to the device-specific folder.

Finally, every application in the device-specific folder is installed on the respective device. If a specific individual application needs to be installed, manually copy the file to the individual folder of the device before running the script. In this case, create the folder manually using the device serial number as the folder name.

To work on multiple phones simultaneously, threading is utilized.

## 1.2 Setup all tokens

*Script:* **setup.py**

*Uses:*

- setup_mail.py
- cal_setup.py
- whatsapp_setup.py

*Preconditions:*

1. Ensure that the preconditions for mail, calendar, and WhatsApp are met. Please refer to their respective sections for more details.

2. This script depends on several other scripts explained later in this document. However, setup.py should be one of the first scripts executed to set up all tokens. It handles the setup of tokens needed for Gmail, Google Calendar, and WhatsApp.

3. If the tokens already exist and have not expired, the program renews the tokens without requiring user input. Otherwise, the tokens need to be claimed one by one.

4. It is crucial to claim the tokens in the exact order that the people are specified in config.json. In case of any problems, it is recommended to delete everything inside the formobile/Token/ directory and rerun setup.py. The devices do **not** need to be connected to the PC.

*Functionality:*

1. The program first claims the Google Calendar token for each account using `cal_setup.py`. An authentication screen appears where you need to login. Repeat this step for each account in the correct order specified in `config.json`.

2. Next, the Google Mail token is stored on the PC using the `setup_mail.py` script. Follow the same steps as before to claim the token.

3. The last step is to obtain the WhatsApp security tokens with `whatsapp_setup.py`. This involves scanning a QR code with WhatsApp on the appropriate smartphone, which automatically verifies if the correct person was chosen.

## 1.3 Contacts

*Script:* **contacts_main.py**

*Uses:*

- `v_card.py`
- `delete_contacts.py`
- `send_contacts.py`

`contacts_main.py` generates a vCard (version 3.0) from the data provided in `vcard_data.csv` and loads it onto the Android device. A vCard is a file format standard for electronic business cards and can contain name and address information, phone numbers, email addresses, URLs, logos, and photographs. The devices must be connected to the PC.

*Preconditions:*

Before executing the program, ensure the following:

Manually populate the `vcard_data.csv` file with the required contact data. The format of the data must follow the standard specified in Table 2.

| first_names | last_names | birthday | orgs | phone_h | phone_w | street | plz | city | email | picture |
|---|---|---|---|---|---|---|---|---|---|---|
| John | Doe | 01/01/1980 | Company A | 123-456-7890 | 987-654-3210 | 123 Main St | 12345 | City A | john.doe@example.com | <Base64 encode Image> |
| Jane | Smith | 05/15/1992 | Company B | 555-123-4567 | 888-999-0000 | 456 Elm St | 54321 | City B | jane.smith@example.com | <Base64 encode Image> |
| Mark | Johnson | 11/30/1975 | Company C | 777-888-9999 | 111-222-3333 | 789 Oak St | 98765 | City C | mark.johnson@example.com | <Base64 encode Image> |

Table 1: Example vcard_data.csv as table

*Functionality:*

The program first establishes connections to the devices via ADB and sets the paths to the required data. It then adds all people from `config.json` to `vcard_data.csv` to enable conversations over WhatsApp and other messaging apps. The modified file is named `vcard_data_with_config.csv`.

Next, the vCard is generated using the `v_card.py` script, resulting in the file `vcards.vcf`.

All contacts on the phone are then deleted using `delete_contacts.py` to avoid duplicate contacts. The vCard is loaded onto the device using `send_contacts.py`. To extract data from the vCard, device-specific inputs have to be made, which need to be manually discovered and saved to `config.json`. The uiautomator tool is installed on the phone to automate pressing the required buttons. Finally, the vCard and uiautomator files are deleted from the phone.

To work on multiple phones simultaneously, threading is utilized.

| Field | Description |
|---|---|
| first_names | First name(s) of contact |
| last_names | Last name of contact |
| birthday | Date of birth in the format YYYY-MM-DD |
| orgs | Organization to which contact belongs |
| phone_h | Telephone number (home) |
| phone_w | Telephone number (work) |
| street | Address information |
| plz | Address information (postcode) |
| city | Address information |
| email | Email address |
| picture | Profile picture of contact (PNG, JPG, JPEG) |

Table 2: Contact Information

## 1.4 Pictures

*Script:* **main_pictures.py**

*Uses:*

- `move_pictures_to_folder.py`
- `modify_timestamps.py`
- `send_pictures.py`

The program `main_pictures.py` fills the Android devices with a predetermined number of pictures.

Note: The photograph dataset is not included in the repository due to rights restrictions. You need to change the `src_path` in `main_pictures.py` to point to the dataset you want to use. Additionally, ensure that pictures with metadata are not loaded onto the smartphones. The devices must be connected to the PC.

*Preconditions:*

1. The phones must be connected to the PC and connected via ADB.

2. `config.json` must contain the following information: - Amount of pictures to store - Start and end dates for the timeframe - Name convention for photo names

*Functionality:*

1. The program first establishes connections to the devices via ADB.

2. Then, using `move_pictures_to_folder.py`, the program moves the specified number of images from the source directory to the destination folders on each device. The destination folder's name is the serial number of the device, and it will contain all the images that will be stored on the device. If the destination folder already exists and is not empty, it will be deleted. Note that this means rerunning the script will result in data loss, but existing photos on the phone will not be deleted. You can adjust the number of pictures by changing the value in `config.json`.

3. To make the photos appear as if they were taken using the smartphone, the program uses `modify_timestamps.py` to modify and randomize the timestamps of the pictures. The start and end dates must be set in `config.json`, and the timestamps will be randomly generated within this time frame. The naming convention for photos on each smartphone is also read from `config.json`.

4. Finally, using `send_pictures.py`, the program stores every image in `formobile/datasets/pictures/gallery/<serial number>` on the respective device:

Threading is utilized to work on multiple phones simultaneously.

## 1.5 Phone Calls

*Script:* **phonecall_main.py**

*Uses:*

- `simulate_phone_call.py`

This script simulates a phone call between two devices with a specified duration. At the end of the call, one randomly selected person hangs up. Both devices must be connected to the PC.

*Preconditions:*

The caller, the called person, and the duration of the call need to be defined in the script.

*Functionality:*

First, the program establishes a connection to the devices via ADB.

Then, the caller and the called person are defined, and their phone numbers are set based on the information in `config.json`.

The `simulate_phone_call.py` script is used to start the automated call and automatically end it after the predefined amount of time.

## 1.6 Browser

*Script:* **chrome_main.py**

*Uses:*

- `populate.py`

The program `chrome_main.py` populates the browser history of every smartphone. The devices must be connected to the PC.

*Preconditions:*

A list of possible websites and Google searches needs to be created. Create a text file with each link on a new line. It's important to include the complete link, including the protocol. For example, a Google search for the PyCharm IDE would be:

`https://www.google.com/search?q=pycharm+ide`

*Functionality:*

First, the program establishes a connection to the devices via ADB.

You can choose to open every link in the text file or a specified number of randomly selected links.

The program opens the links one by one on the smartphones, waiting three seconds after each link to allow Chrome to log the visited website in the browsing history.

Threading is utilized to work on multiple phones simultaneously.

## 1.7 GPS Spoofing

*Script:* **spoofing_main.py**

*Uses:*

- `enjoying.py`
- `working.py`
- `weekend.py`
- `download_delete.py`

This script manipulates the GPS location of the devices using the Appium Settings app. It sends latitude and longitude GPS information to mock the device's location. The devices must be connected to the PC.

*Preconditions:*

1. Set the enjoying, working, and weekend hours, as well as the location for each time period, for each person in the `config.json` file. The *enjoying* hours refer to leisure time after work.

2. Install the Appium Settings app on the devices. Connect the smartphones to the PC and run `download_delete.py`. Ensure that the argument of the `download_delete` function is set to *download*. If you no longer need to spoof locations, you can delete the Appium Settings app from your devices. Set the argument of the `download_delete` function to *delete*.

*Functionality:*

Set the argument of the `location_spoofer` function to *start*. This will establish connections to the devices via ADB.

Appium is loaded into the RAM of each smartphone, and GPS spoofing starts automatically. Google Maps opens automatically to log the location. The program restarts every 30 minutes to keep Appium in the RAM.

If you want to stop GPS spoofing, stop the program and set the argument of the `location_spoofer` function to *stop*. Then, restart the script.

Threading is utilized to work on multiple phones simultaneously.

## 1.8 Reddit Data

Reddit data can be used to populate group chats and generate email content. We used the subreddit comment downloader to collect the data from Reddit.

We provide the scripts we used to format the downloaded Reddit data. Please note that you will need to make changes to the scripts, such as providing your own Reddit secret ID, etc. The devices do not need to be connected to the PC for this process.

`reddit_main.py` utilizes the following scripts:

- `get_mods.py`
- `concat_subreddit_dl_data.py`
- `reddit_comments.py`

*Preconditions:*

- You will need a Reddit account, as well as the Reddit ID, secret, and username. Please refer to the instruction set provided by the authors on how to use these credentials:
`https://github.com/pistocop/subreddit-comments-dl`

- You should know the case-sensitive name of the subreddit from which you want to download comments. You can find subreddits on `https://www.reddit.com/subreddits/`

*Functionality:*

The script begins by downloading the specified subreddit (this may take some time) using the Linux shell. The downloaded files are then concatenated into a single main file using `concat_subreddit_dl_data.py`. `get_mods.py` automatically downloads a JSON file containing the details of all the moderators of the given subreddit. Finally, `reddit_comments.py` prepares the data by deleting comments made by the moderators and formatting it according to our requirements.

Note: The formatting and customization of the data may require additional modifications to the scripts based on your specific needs.

## 1.9 Calendar

*Script:* **calendar_main.py**

*Uses:*

- write_events.py
- clear_calendar.py
- create_event.py

This script is used to create Google Calendar entries that will be synced to the devices.

The calendar entries are stored in the following location: `formobile/google_calendar/events/PERSON_TOKEN_NAME.csv`

To manage calendar entries, manually created Google tokens are required for each account. These tokens expire two weeks after creation, so they need to be regularly updated. The devices do not need to be connected to the PC.

*Disclaimer:* These scripts heavily rely on Google APIs and the Google Cloud Console. The information in this document may become outdated in the future. For problem-solving, please refer to the official Google documentation.

*Preconditions:*

You need to set up a fully configured Google Cloud Platform project and have the `credentials.json` file downloaded. Please follow the instructions [link] for the setup process.

The `PERSON_TOKEN_NAME.csv` files need to be filled with data in the specified format. Each row represents an event, and the columns correspond to different event attributes such as summary, description, location, start time, end time, timezone, organizer, attendees, recurrence, and recurrence details. See the Table 3 for an example and Table 4 for additional information about the format of the entries.

| Summary | Description | Location | Start | End | Timezone | Organizer | Attendees | Recurrence | How Often | Specify Days |
|---|---|---|---|---|---|---|---|---|---|---|
| Meeting 1 | Lorem ipsum dolor sit amet | Office A | 2023-06-01 | 2023-06-01 | UTC+2 | John Doe | Jane Smith | Daily | Every 2 days | Monday, Wednesday |
| Conference | consectetur adipiscing elit | Conference Center | 2023-06-10 | 2023-06-12 | UTC-5 | Sarah Johnson | Mark Davis, Emily Brown | Weekly | Every week | |
| Project Review | Sed do eiusmod tempor incididunt | Online | 2023-07-01 | 2023-07-01 | UTC+1 | Michael Lee | All Team Members | Monthly | Every month | |

Table 3: Example of a calendar table

*Functionality of the program:*

| Field | Description |
|---|---|
| summary | Title of the event |
| description | Further explanation |
| location | Place of meeting |
| start | Start date and time in the format YYYY-MM-DDTHH:MM:SS |
| end | End date and time in the format YYYY-MM-DDTHH:MM:SS |
| timezone | Timezone (e.g., Europe/Berlin) |
| organizer | Head of the meeting |
| attendees | Participants of the meeting |
| recurrence | Recurrence pattern (daily, weekly, monthly, yearly) |
| how often | Number of times the event occurs |
| specify days | Specific days (MO, TU, WE, TH, FR, SA, SU) |

Table 4: Event Information

The script starts by checking for valid Google tokens. If the tokens have expired or do not exist, an authentication screen will appear. You need to log in with the person's Google account, and the token will be automatically stored locally. The token name is set in the `config.json` file.

The `calendar_main.py` script executes the `populate_calendar` function. This function sets the paths to the tokens and the `PERSON_TOKEN_NAME.csv` files. If you want to delete existing calendar entries before creating new ones, pass `True` as an argument to the `populate_calendar` function. This will invoke the `clear_calendar.py` script, which removes all existing entries.

The script then creates events in the Calendar using the data from the `PERSON_TOKEN_NAME.csv` file using the Google Calendar API. The events are automatically synced with the devices.

Note: Make sure to customize the event data in the `PERSON_TOKEN_NAME.csv` file according to your needs and follow the specified format.

## 1.10 Emails

*Script:* **gmail_main.py**

*Uses:*

- **gmail_account.py**
- **get_mods.py**
- **reddit_email.py**

This script is used to send automated emails using Gmail accounts of the participants specified in the script and in `config.json`. To send emails, manually created Google tokens are required for each account. These tokens expire two weeks after creation, so they need to be regularly updated.

In our test runs, the content of the emails originated from the subreddit "SeriousConversation" and was already prepared to fit our needs.

The devices do not need to be connected to the PC.

*Preconditions:*

You need to set up a fully configured Google Cloud Platform project and have the `credentials.json` file downloaded. Please refer to [link] for detailed instructions.

*Functionality:*

1. Upon the first execution of the script, Google tokens need to be claimed. If the tokens have expired or do not exist, an authentication screen will appear. You need to log in with the person's Google account, and the token will be automatically stored locally. The token name is set in the `config.json` file.

2. The script sends new emails to the email partners specified in the `config.json` file. It also attempts to reply to emails received in the past.

3. In the `config.json` file, each email partner of the person receives an individual salutation, and the person itself also has a personal signature.

Note: Please make sure to customize the email content according to your needs. In our case, the content originated from the "SeriousConversation" subreddit, but you can modify it to fit your requirements.

## 1.11 Finite State Machine and WhatsApp

*Script:* **main_fsm.py**

*Uses:*

- `whatsapp_fsm.py`

The program `main_fsm.py` implements a finite state machine (FSM) for realistic automated WhatsApp chats. The FSM determines the current state of the WhatsApp chatbot based on probabilities defined in the transition matrix. The transition matrices for every person defined in `config.json` should be generated fitting your needs. These transition tables should be stored under `/formobile/messenger/transition_matrices/transition_matrix_{PERSON_TOKEN_NAME}`.

The devices do not have to be connected to the PC.

The data sent in private conversations is from the Amazon Alexa AI group and should not be modified.

If you wish to send messages in group chats, make sure that the corresponding dataset is ready. Afterwards, change the variable `REDDIT_DATA` in `definitions.py` to fit your needs.

Each time the script is terminated, the current state of conversations is stored in `~/formobile/messenger/whatsapp/conversations.json`. This data is loaded on restart to ensure that conversations are not broken.

The program attempts to send automatic voice messages via text-to-speech for each person with a nonzero probability of sending voice messages, as defined in the `config.json` file. To enable this functionality, you will need an external sound card and an AUX cable. Connect one end of the AUX cable to the input and the other end to the output of the sound card. Plug the sound card into a USB port on your PC, as shown in Figure 1.

When you start the program for the first time, WhatsApp Web will open, and a QR code will be displayed. Scan the QR code with the corresponding device's WhatsApp app. The security token, generated by scanning the QR code, will be stored locally. This means that authentication will not be required for this device in the future. However, the device still needs an active internet connection for WhatsApp Web to function properly.

*Functionality:*

The script uses Selenium and Google Chrome with WhatsApp Web to automatically send WhatsApp messages. These messages are automatically synchronized with the WhatsApp installation on the connected phones.

DISCLAIMER: The script relies on CSS selectors/XPATH of the WhatsApp Web client, which can become outdated quickly. If WhatsApp makes changes, these selectors will need to be updated.

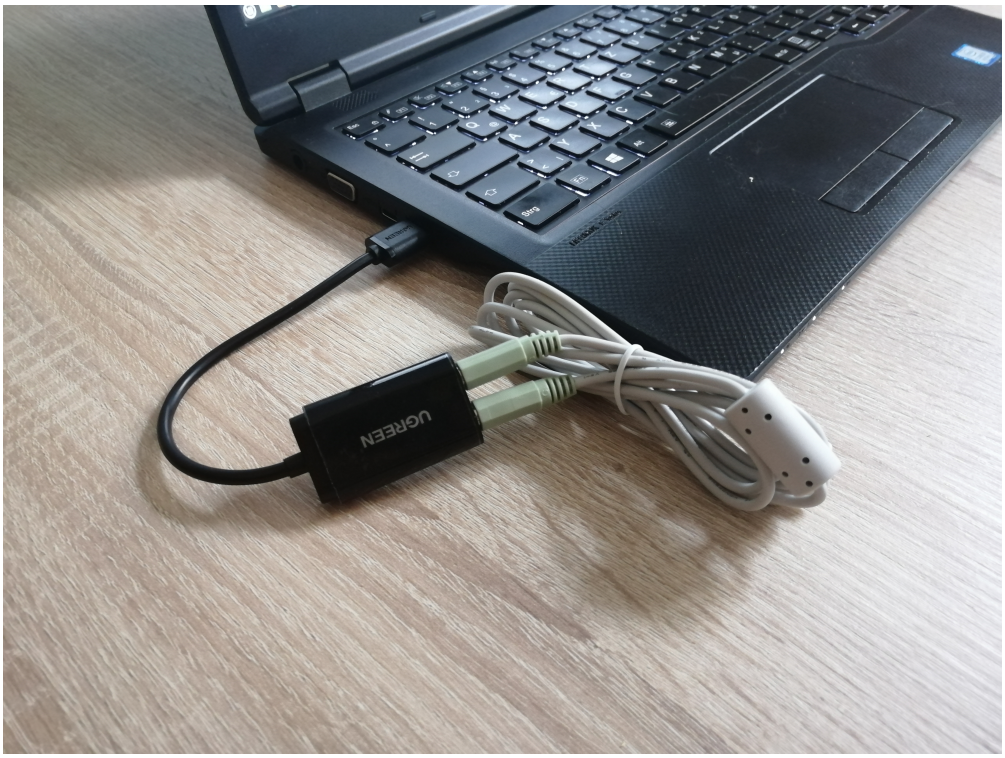


Figure 1: External soundcard

The program creates objects from the *Person* class for each person defined in the `config.json` file and sets the required attributes. Afterwards, *start* is called for each person. Within this function, an endless loop calls `whatsapp_fsm.py`. This script determines the current state of each person and handles state transitions based on the defined probabilities. The possible states and their purposes are described in Table 5.

There was an atttempt to make the Finite State Machines modular allowing the addition and removal of modules such as email or location spoofing based on your needs. However, due to difficulties, this approach could not be fully explored. As a result, `whatsapp_fsm.py` currently handles all branches of the FSM, as shown in Figure 2.

Threading is implemented to enable working on multiple browsers simultaneously.

Please note that you may need to adjust the script if there are any changes in the WhatsApp Web interface.

| Action | Description |
|---|---|
| idle | Person waits for command |
| whatsapp_idle | Person is online, but inactive |
| whatsapp_read_message | Marks the message as read (blue checkmarks) |
| whatsapp_start_new_chat | Starts a chat with someone new |
| whatsapp_send_message | Sends a text or voice message |
| email_idle | Person is online, but inactive |
| email_read | Marks the message as read |
| email_write | Sends a text message |
| mock_location | Spoofs the GPS location |
| browse | Creates a browsing history |
| take_photo | Saves a photo in the gallery |

Table 5: Current states and their description

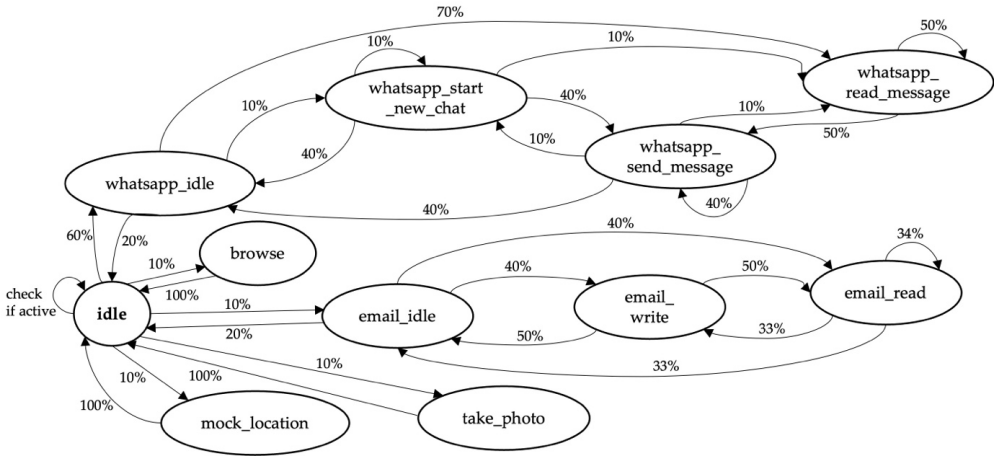


Figure 2: Current transitions and their probability

# 2 Configuration file explanation

## 2.1 Person information

This part of the configuration file contains personal information about the person.

```
{
  "First name": "John", # First name of the person
  "Surname": "Doe", # Surname of the person
  "Birthday": "01.01.1960", # Birthday of the person
  "Home phone": "XXXXXXXX", # Home phone number of the person
  "Street": "Main street 1", # Street of the persons address
  "Postal code": "XXXXX", # postal code of the persons address
  "City": "Capital", # city of the persons address
  "Organisation": "Organisation Name", # Organisation where the person works
}
```

This information is added to the `vcard_data_with_config.csv` from which the vCard is generated. Only one of the above fields must have data in it in order to add the information to the CSV and thus to the vCard. However, in order to properly run the WhatsApp script, it is necessary to store at least:

- First name
- Surname
- Phone number

Otherwise, the script will not be able to select chats properly.

## 2.2 Device specific information

```
"Mobile devices": [
            {
                "Brand": "Huawei",
                "Model": "SNE-LX1",
                "nr of pics": 142,
                "timeframe start": "2021-07-13T08:59:37.512752Z",
                "timeframe end": "2021-09-07T08:59:37.512752Z",
                "individual apks": 1,
                "Serial number": "HYFDUXXXXXXXXXX",
                "Phone number": "+ 4915XXXXXXX",
                "active times": {
                    "6:10": 6,
                    "12:00": 7
                },
                "Whatsapp token": "PERSON_A",
                "Conversation partners": ["PERSON_B", "PERSON_C"],
                "typing speed": 0.6,
                "voice message probability": 0.05
            }
          ],
```

**Brand:** Brand of the phone. Provides a hint about the app names needed to store contacts on the device, but requires further investigation.

**Model:** Model of the phone. Used to identify the phone model when importing contacts to the device.

**nr of pics:** Number of pictures that should be stored on the device. Used in ‘setup_phones.py‘ when copying pictures from the dataset.

**timeframe start/end:** Lower and upper bounds of timestamps of the pictures. Used in ‘setup_phones.py‘ to modify the timestamps of the pictures before storing them on the device. The format should be: `%Y-%m-%dT%H:%M:%S`.

**individual apks:** Number of individual apps that should be stored on the device. The apps in the ‘datasets/apps‘ folder are split into two different folders:

- **standard:** Apps that should be installed on every phone.
- **individual:** Apps that should not be installed on every phone.

The number specified for ”individual apks” indicates how many random apps from the individual folder should be installed.

**Serial number:** Serial number of the device. Used for identifying devices when using ADB (pictures, apps, contacts) and naming corresponding folders.

**Phone number:** Phone number of the device. Used for vCard generation and future phone calls/SMS.

**active times:** Dictionary of times when the person should be active. Key represents the start time of the active phase, and the value represents the duration of the active phase in hours. Used to check whether the person is active in ‘fsm.py‘.

**WhatsApp token:** Name of the WhatsApp token folder. Specifies the folder where the WhatsApp token is saved. Used for identification (see conversation partners) and when loading the WhatsApp token. Also used to specify the subfolder of the person in the Token directory.

**Conversation partners:** List of conversation partners. Contains the WhatsApp token names of other people defined in the configuration file. Used to specify with whom the person should communicate over WhatsApp.

**typing speed:** Typing speed of the person. Used to calculate the time it takes to write a message. Given in words per second.

**voice message probability:** Probability for a voice message. Used to determine whether the message will be sent as a text or voice message.

## 2.3 Gmail account information

This part of the configuration file contains information about the Gmail accounts of the person. One person could theoretically have more than one Gmail account, but currently the scripts expect each person to have only one account.

```
"Gmail account": [
            {
                "Gmail address": "wilhelmborange@Gmail.com",
                "Gmail password": "",
                "Gmail token": "orange_Gmail.json",
                "Calendar token": "orange_calendar.json",
                "Signature": "Best regards\nWilhelm",
                "E-mail partners": {
                    "yorkr757@Gmail.com": "Hey Rob\n\n",
                    "thomasalpin83@Gmail.com": "Dear Mr. Alpin\n\n"
                }
            }
        ]
```

**Gmail address:** E-mail address of the person's account. Currently not used in the codes but documented for reference, especially for e-mail partners.

**Gmail password:** Password to the e-mail address. Currently not used, but may be used for account generation in the future.

**Gmail token:** Name of the Gmail token folder. Specifies the folder where the Gmail token is saved. Used for loading the Gmail token. The file should be in the '.json' format.

**Calendar token:** Name of the Google Calendar token folder. Specifies the folder where the Google Calendar token is saved. Used for loading the Google Calendar token. The file should be in the '.json' format.

**Signature:** Signature for e-mails. The signature is appended to the end of every e-mail the person sends.

**E-mail partners:** Dictionary of all the e-mail conversation partners.

- Key: E-mail address of a possible e-mail conversation partner.
- Value: Salutation for that person.

## 2.4 WhatsApp groups information

This part of the configuration file contains information about WhatsApp groups.

```
"Whatsapp groups": {
    "New York": ["PERSON_A", "PERSON_B", "PERSON_C"],
    "Old York": ["PERSON_A", "PERSON_B", "PERSON_C"],
    "New NEW Group": ["PERSON_A", "PERSON_B", "PERSON_C"]
  },
```

**Whatsapp groups:** Dictionary with group name and participants.

- Key: Name of the group.
- Value: WhatsApp token filenames of the members.

If the configuration file only contains two people, this section is best kept empty.

## 2.5 Information about datasets

This part contains information about where the picture dataset is stored on the computer.

```
"Picture dataset": "/home/XXXXXXX/Downloads/VG_100K",
```

In this example the visual genome dataset was downloaded and stored in the Downloads folder.

## 2.6 Phone specific information

This part of the configuration file contains information that is specific to the devices.

Each android phone that's currently supported has such an entry in the configuration file. This example is for the Samsung S9 duos.

```
"SM-A405FN": {
    "Buttons contacts": [
        "Save",
        "Phone",
        "Just once"
      ],
    "Contacts store": "com.samsung.android.app.contacts",
    "Contacts delete": "com.samsung.android.providers.contacts",
    "Photo names": "%Y%m%d_%H%M%S"
},
```

**Buttons contacts:** List of buttons that need to be pressed in order to import the contact information from the vCard. For this phone, after the .vcf file is read, the buttons 'Save', 'Phone', and 'Select' need to be pressed. These buttons can change between different Android versions for this phone. In that case, the buttons in the configuration file must be adjusted. The button instructions are case sensitive!

**Contacts store:** App that is responsible for storing contacts. The app that is responsible for storing contacts is different for each phone model. Used when storing contacts.

**Contacts delete:** App that is responsible for the deletion of contacts. The app that is responsible for the deletion of contacts is different for each phone model. Used when deleting contacts.

**Photo names:** Format string for naming of photos. Each phone has a specific way of naming photos that are taken with the camera. This string is the format specification that, when used, formats the randomly generated timestamp to the device-specific string. Used when naming the photos.

## 2.7 Example config.json

```
{
  "People": [
    {
      "First name": "Anna",
      "Surname": "Kiwi",
      "Birthday": "",
      "Home phone": "+420731872724",
      "Street": "",
      "Postal code": "",
      "City": "",
      "Organisation": "",
      "Mobile devices": [
        {
          "Brand": "Huawei",
          "Model": "SNE-LX1",
          "nr of pics": 2,
          "timeframe start": "2017-01-12T14:12:06",
          "timeframe end": "2018-01-12T14:12:06",
          "individual apks": 0,
          "Serial number": "HYFDU19823003328",
          "Phone number": "+420731872724",
          "x coordinate": 550,
          "y coordinate": 900,
          "active times": {
            "0:00": 1439
          },
          "working time": {
            "11:00": 600
          },
          "spare time": {
            "22:00": 180
          },
```

```
          "weekend activity time": {
            "12:00": 360
          },
          "Whatsapp token": "PERSON_A",
          "Conversation partners": ["PERSON_B"],
          "typing speed": 0.5,
          "reading speed": 0.2,
          "voice message probability": 0.05,
          "attachment probability": 0.02,
          "google probability": 0.3,
          "home location": {
            "latitude": 1,
            "longitude": 1
          },
          "work location": {
            "latitude": 2,
            "longitude": 2
          },
          "spare time location": {
            "latitude": 3,
            "longitude": 3
          },
          "weekend activity location": {
            "latitude": 4,
            "longitude": 4
          }
        }
      ],
      "Gmail account": [
        {
          "Gmail address": "a.kiwi1994@gmail.com",
          "Gmail password": "",
          "Gmail token": "anna_gmail.json",
          "Calendar token": "anna_calendar.json",
          "Signature": "Bye\nAnna",
          "Email partners": {
            "benjam199695@gmail.com": "Hey Ben\n\n"
          }
        }
      ]
    },
    {
      "First name": "Benjamin",
      "Surname": "",
      "Birthday": "",
      "Home phone": "+420731209687",
      "Street": "",
      "Postal code": "",
      "City": "",
      "Organisation": "",
      "Mobile devices": [
        {
          "Brand": "Samsung",
          "Model": "SM-A405FN",
          "nr of pics": 0,
          "timeframe start": "",
          "timeframe end": "",
          "individual apks": 0,
          "Serial number": "R58MA2M7C8D",
          "Phone number": "+420731209687",
          "x coordinate": 500,
          "y coordinate": 800,
          "active times": {
            "0:00": 1439
          },
```

```
          "working time": {
            "11:00": 600
          },
          "spare time": {
            "22:00": 180
          },
          "weekend activity time": {
            "12:00": 360
          },
          "Whatsapp token": "PERSON_B",
          "Conversation partners": ["PERSON_A"],
          "typing speed": 0.5,
          "reading speed": 0.2,
          "voice message probability": 0.05,
          "attachment probability": 0.02,
          "google probability": 0.2,
          "home location": {
            "latitude": 1,
            "longitude": 1
          },
          "work location": {
            "latitude": 2,
            "longitude": 2
          },
          "spare time location": {
            "latitude": 3,
            "longitude": 3
          },
          "weekend activity location": {
            "latitude": 4,
            "longitude": 4
          }
        }
      ],
      "Gmail account": [
        {
          "Gmail address": "benjam199695@gmail.com",
          "Gmail password": "",
          "Gmail token": "benjamin_gmail.json",
          "Calendar token": "benjamin_calendar.json",
          "Signature": "Bye\nBenjamin",
          "Email partners": {
            "a.kiwi1994@gmail.com": "Hi Anna\n\n"
          }
        }
      ]
    }
  ],
  "Whatsapp groups": [
    {
      "Just talking": ["PERSON_A", "PERSON_B"],
      "group_conversation_path": "/datasets/CasualConversation/group_chats.json"
    },
    {
      "Harry Potter - no muggles": ["PERSON_A", "PERSON_B"],
      "group_conversation_path": "/datasets/harrypotter/group_chats.json"
    }
  ],
  "Picture dataset": "",
  "SM-A405FN": {
    "Buttons contacts": [
      "Save",
      "Phone",
      "Just once"
    ],
```

```
    "Contacts store": "com.samsung.android.app.contacts",
    "Contacts delete": "com.samsung.android.providers.contacts",
    "Photo names": "%Y%m%d_%H%M%S"
  },
  "S20 Lite": {
    "Buttons contacts": [
      "Phone"
    ],
    "Contacts store": "com.android.contacts",
    "Contacts delete": "com.android.providers.contacts",
    "Photo names": "IMG_%Y%m%d_%H%M%S"
  },
  "SNE-LX1": {
    "Buttons contacts": [
      "OK",
      "Phone"
    ],
    "Contacts store": "com.android.contacts",
    "Contacts delete": "com.android.providers.contacts",
    "Photo names": "%Y%m%d_%H%M%S"
  },
  "SM-A505FN": {
    "Buttons contacts": [
      "Save",
      "Phone",
      "Just once"
    ],
    "Contacts store": "com.samsung.android.app.contacts",
    "Contacts delete": "com.samsung.android.providers.contacts",
    "Photo names": "%Y%m%d_%H%M%S"
  }
}
```