Horizon 2020 PROGRAMME *ICT-01-2014: Smart Cyber-Physical Systems*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 643924

D4.1
Demonstrator 1, Configuration App

1 DOCUMENT INFORMATION

Authors	T. Larmoire (THALES) C. Fedorczak (THALES) A. Pagani (DFKI)
Responsible Author	Alain Pagani e-mail: alain.pagani@dfki.de

Keywords	Demonstrator 1 – Peephole demonstrator
WP / Task	WP4
Nature	Internal document
Dissemination Level	PU

2 DOCUMENT HISTORY

Person	Date	Comment	Version
Alain Pagani	06.06.2018	Delivered version	1.0

3 ABSTRACT

This document is a software description document that accompanies the deliverable D4.2 “Peephole Demonstrator Configuration App”. This deliverable is the software developed on the companion device (here a smartphone running Android) in order to implement the Peephole surveillance demonstrator. The software is made available to the reviewers on a GitLab server. This document first provides a short description of the demonstrator and its features. It then describes the developed software that runs on the companion device.

4 TABLE OF CONTENTS

1 Document Information 2
2 Document History 3
3 Abstract 4
4 Table of Contents 5
5 Introduction 6
6 Short description of the demonstrator 7
6.1 Description of the use-cases 7
6.2 Description of the parts used for the development 9
6.3 Software Architecture 10
6.4 Hardware set-up 13
7 IFOYD App Software Description 18
7.1 Demonstrator requirements 18
7.2 Software description 21
8 IFOYD App Software Documentation 23
8.1 Launching the IFOYD App 23
8.2 Main menu 24
8.3 Configuration 25
8.4 Camera 26
8.5 Live 27
8.6 Detection Event list 28
9 Conclusions 30

5 INTRODUCTION

The initial specification of the peephole requirements was started in April 2015 and was finalised in 2016. It continued to be refined in parallel to the progress of the hardware design (WP2) and the middleware design (WP3). It describes the different use cases associated with the peephole demonstration.
The requirements for the peephole demonstrator have been chosen to fulfil common needs in the security market but also to test features of increasing complexity in terms of processing power for the EoT device, in order to evaluate the limits achievable in terms of edge processing.

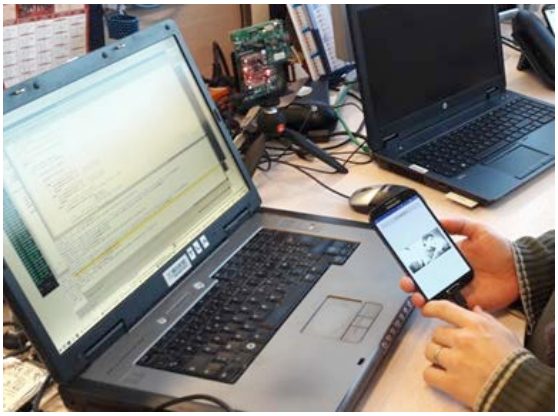


Figure 1 - EoT and IFOYD App development

The demonstrator is composed of the **EoT device** running the application software, developed by DFKI, and the **Android Smartphone** with the **IFOYD (In Front Of Your Door) app,** developed by Thales. This document describes the configuration application software developed for the Android device.

As an additional task, an **EoT device simulator** running on a **laptop** has been developed by Thales in order to start the development of the App while the EoT device was still in development. This software is also described briefly.

The App software has been stored in the Gitlab repository:

https://gitlab.com/espiaran/EoT/tree/peephole/WorkPackage_4/Peephole/android

The reviewers will be able to access the private parts of the code on request.

6 SHORT DESCRIPTION OF THE DEMONSTRATOR

The Peephole demonstrator is composed of an EoT device and an Android Smart Phone with a dedicated App to exchange commands and data with the device.

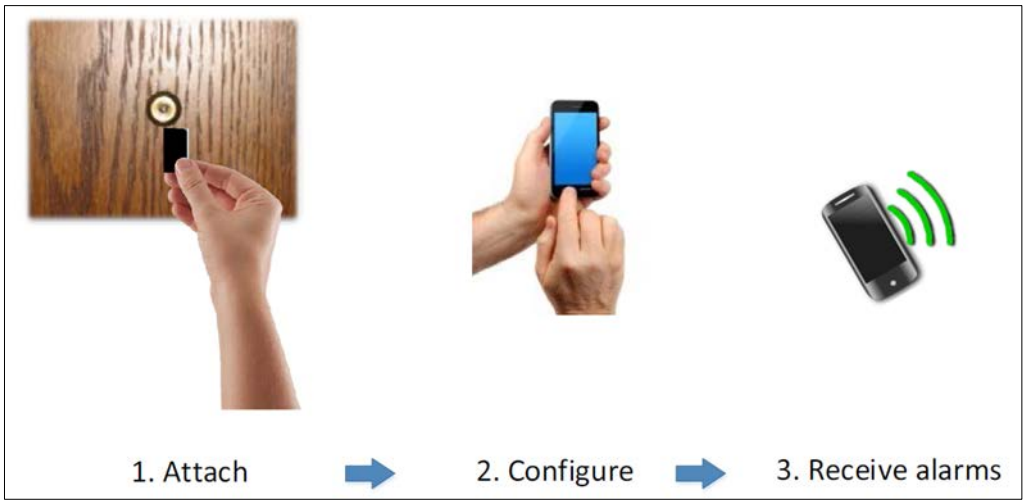


Figure 2 - Peephole demonstrator

After power on, the EoT device connects to the network using its Wi-Fi connection.
The Android phone has to connect to the device through the network to be able to control it and receive data and alarms from it.
The EoT device holds an MQTT broker that will manage the exchanges with the Android phone. Several Android phones can be connected simultaneously to the device and get alarms, images and video clips from the device.

The Android App that has been developed and that will be described in this document is a functional prototype that gives access to the features of the EoT peephole device. No effort was spent on the aesthetic aspect of the different screens, as they are likely to be customised.

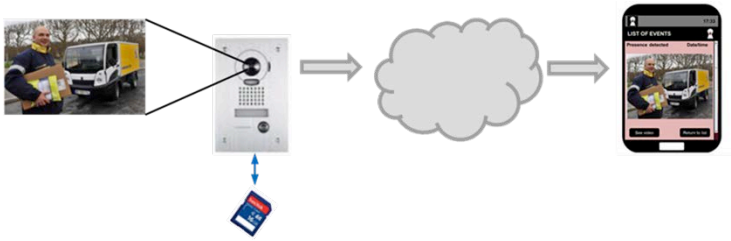
The interface specifications have been elaborated by DFKI and THALES based on the general software architecture and the use of the PULGA broker.

A document called **EOTMQTTProtocol_Vx** has been created by **DFKI** in May 2017 and has been upgraded since then by DFKI and **Thales**. The last version of the document is currently V5 and has been release on May 22, 2018. This document is provided as an annex.

6.1 Description of the use-cases

The peephole demonstrator has been developed by DFKI and Thales according to the requirements produced for WP2 and WP3.
To verify the completion of the requirements, five use cases have been selected:

a. **Surveillance**:

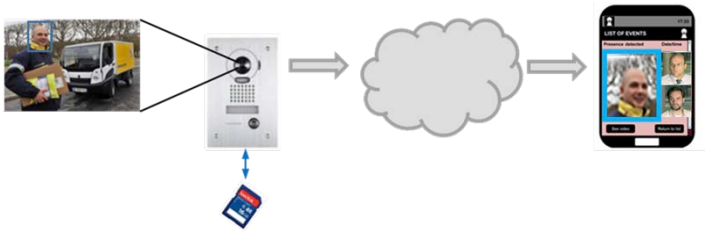


The device is continuously recording in a circular buffer and alarm is sent on motion detection. Live view is available on demand.

b. **Tampering detection**:

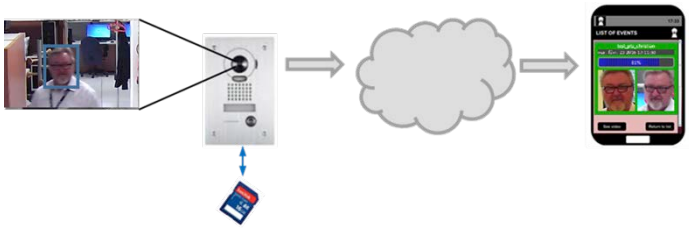
An alarm is sent in case of a failure or a tampering is detected.

c. **Face detection**



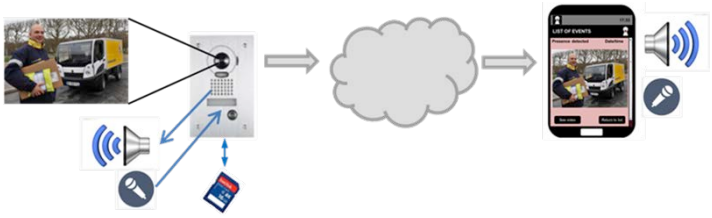
Faces are detected and cropped images of the faces are recorded and transmitted

d. **Face recognition (optional)**



Detected faces are compared to a white list stored in the EoT device, and when a match occurs, an event is generated.

e. **Bi-directional audio transmission (optional)**



A bidirectional audio link is created between the EoT device and the Android Smartphone

The original requirement analysis document mentioned marked two scenarios as optional: Scenario 4: Face recognition, and Scenario 5: Bi-directional audio transmission. These two scenarios were optional and were to be implemented only after the successful implementation of the first three scenarios. After the development of the three first scenarios, the allocated budget was already consumed, and the optional scenarios have not been implemented.

6.2 Description of the parts used for the development

The demonstrator implemented is composed of:

i. A Smartphone with the IFOYD App.

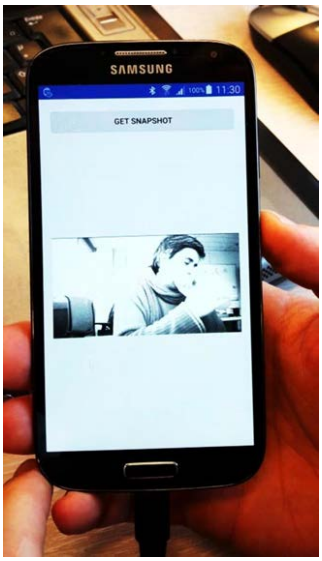


Figure 3: A view of the IFOYD app on the selected Smartphone

The Smartphone used for the test is a Samsung Galaxy 4, Model GT-19505. The Android version installed on the Smartphone is Android 5.01 (Lollipop). The IFOYD App has been developed in JAVA using Android SDK.

ii. A Wi-Fi access point:

Figure 4: Typical router for the Wi-Fi connection

A standard Wi-Fi access point is used in order to allow connection of the EoT device and the Smartphone.

iii. The EoT device:

Figure 5: Early version of the EoT Device (R1)

The development has been done using the different releases of the EoT device.

iv. A laptop for the development

Figure 6: Laptop used for development

Thales does most of the developments on Linux (CentOS or Ubuntu). The EoT App has been developed using Eclipse IDE under Ubuntu environment.

An EoT device simulator has also been developed running on that PC in order to develop and debug more easily the Android App. This simulator will be described below.

6.3 Software Architecture

6.3.1 EoT device

The software architecture for the EoT Device is described in a dedicated document (Deliverable D4.1).
It is mainly composed of an application software and a communication module in the form of an MQTT broker. This broker uses the PULGA library derived from the MOSQUITTO implementation.

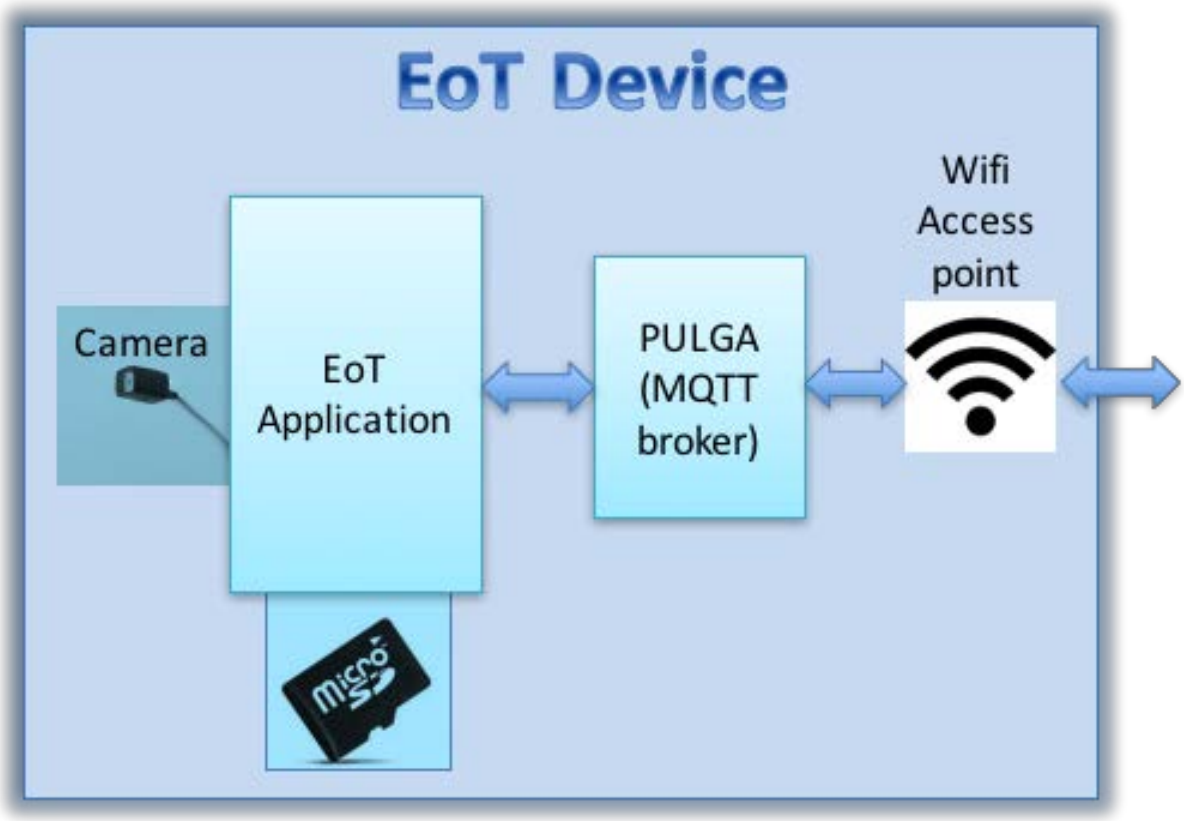


Figure 7: Architecture of the EoT Device Software

The connection is then established using the WiFi_Functions library.

6.3.2 Android App: IFOYD

This Application has been developed in JAVA using Eclipse and the Android SDK. Elementary commands have been defined for the dialog between the EoT device and the IFOYD App. They are described in the document: **EOTMQTTProtocol_v5.docx** dated 22/05/2018, available in Annex.

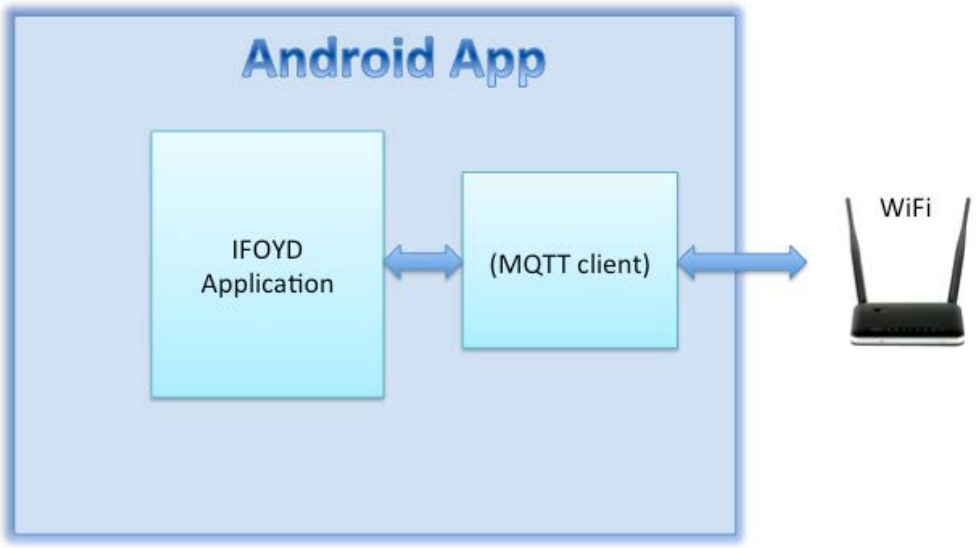


Figure 8: Architecutre of the android app

The app is using a Java MQTT Client library available in the Eclipse environment. The main application is organised in different modules corresponding to the elementary tasks needed for the communication with the EoT Device.

The IFOYD App is compiled to an .apk file that can be directly installed on the Smartphone, once downloaded or copied in its memory.
The App installation process is standard and doesn't require any specific parameters.
An icon for the IFOYD app is created on the Applications pages and can be added to one of the access screens.

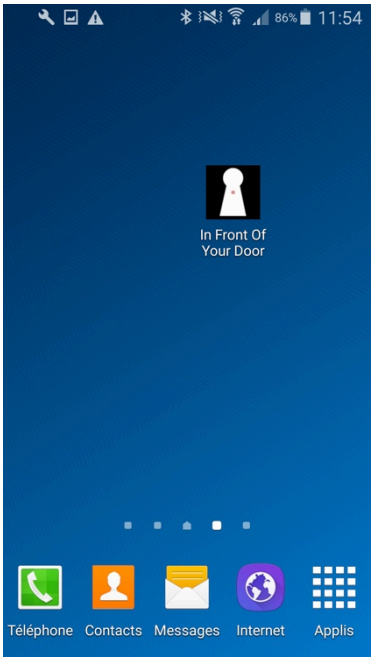


Figure 9: Icon of the IFOYD app on the main screen

6.3.3 EoT device simulator

The EoT device simulator has been developed on a laptop under Ubuntu and tcl programming language.

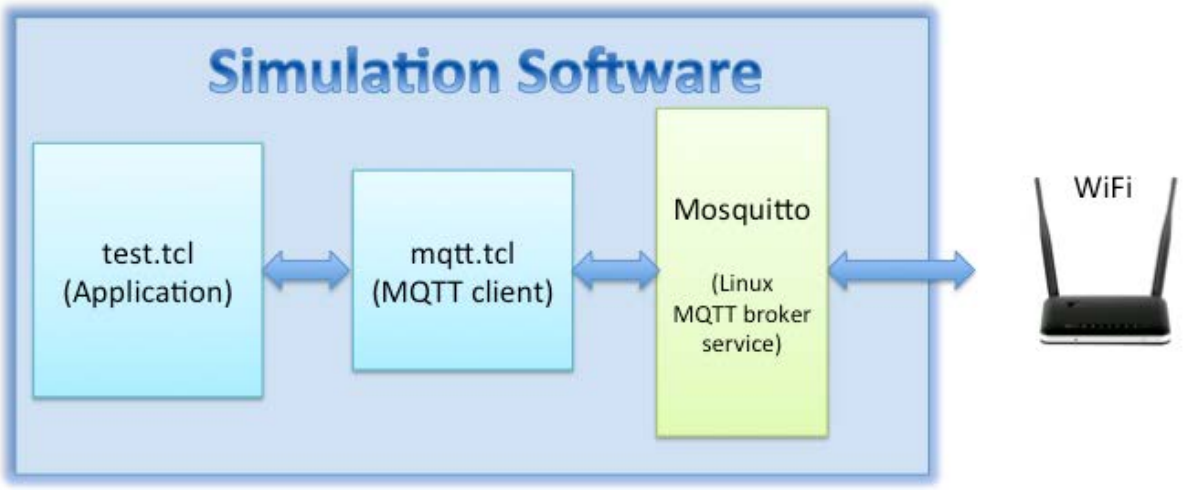


Figure 10: Architecture of the EoT Device Simulator

Two main applications have been developed:

- test.tcl which is the application programme that simulates the EoT device, responds to the commands sent by the IFOYD app and generates some events.
- mqtt.tcl which is an MQTT client. This module is launched directly by the test.tcl programme.

The simulator also uses the MOSQUITTO process as an OS service. Ubuntu has been configured to launch MOSQUITTO at boot.

This simulator uses some images and video sequences stored in the hard disk in order to simulate the different functions and provides a terminal to display the messages received from the Android app as well as the responses.
This simulator has proven very useful as separate teams located on different sites and sharing different networks did the development of the EoT device software and the Android app.

6.4 Hardware set-up

6.4.1 EoT device configuration

During the initial configuration of the EoT Device, it is necessary to configure the IP address of the device. This is done using the bootloader and the software Pulga for a first configuration step. The IP address of the device is configured by default to 192.168.1.1 and the broker can be accessed on port 1883.

A change of the IP address and the broker port will be possible from the Android device. This will need the “config” switch located on the EoT board to be pressed for a least 3 sec. The EoT board will switch to configuration mode. Using the

Configuration menu, the Android App will allow changing the IP address and the broker port. This change will be effective when the “Apply” button of the App will be pushed.
At the date of the report, this configuration is still in development.

6.4.2 Android Smartphone configuration

The Android Smartphone running the IFOYD app needs to be connected using its Wi-Fi connection to the EoT device’ WiFi access point.
This is done directly in the settings of the phone. DHCP can be used as the EoT device supports this feature.

Once connected to the Wi-Fi, the IFOYD app can be launched. The first screen shows the different tasks that can be accessed.

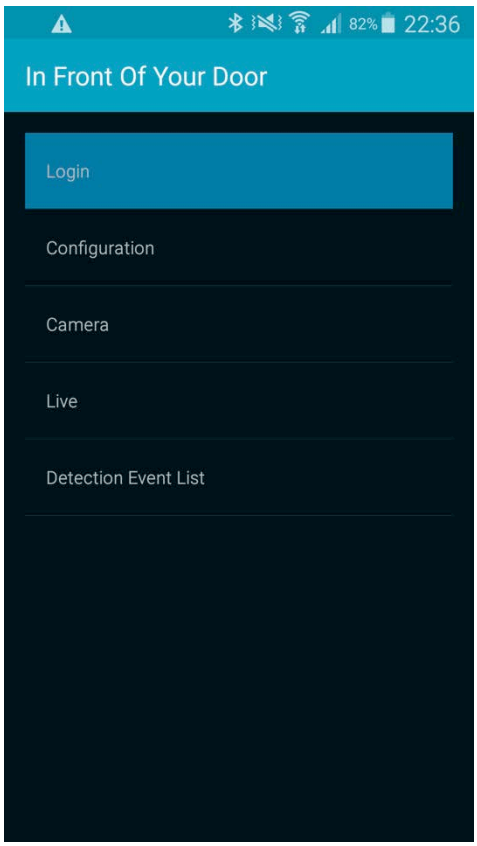


Figure 11: First screen with the different tasks

The focus is given on the login menu. As long as the connection hasn't been initialised with the EoT Device, the other menus cannot be accessed.

Once the "Login" button has been activated, a new page appears asking for the EoT Device IP address (in fact the MQTT broker address).

The initial value for the broker has been defined to be 192.168.0.8. If the device is not using this address, it can be edited directly on the address window.

The MQTT port is initialised by default to 1883 as TCP/IP port 1883 is reserved with IANA for use with MQTT. TCP/IP port 8883 is also registered, for using MQTT over SSL.

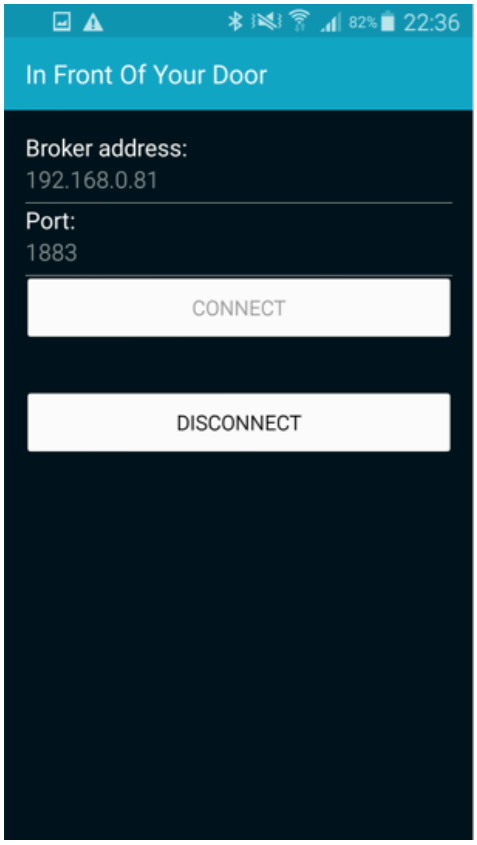


Figure 12: Configuration of the IP address and port

In the current version, the broker is embedded inside the device.

6.4.3 Simulator configuration

The laptop, running the simulator, needs to be connected to the same network as the EoT Device, with access to the same segment. This connection is usually done in Wi-Fi, but an Ethernet wired connection could also be used as the Wi-Fi access point used for the Android Smartphone can be accessed.

By default, the MOSQUITTO service is launched at boot if Upstart is installed (by default on Ubuntu).
The initialisation file “mosquitto.conf” is located in the /etc/init folder and contains the MOSQUITTO setup. The default .conf file, provided with the MOSQUITTO package, has been used.

The next step is to move to the simulator folder and to launch the ./test.tcl programme.

The terminal will display:

```
cfe@cfe-HP-ProBook-640-G1 /data/eot/EoT-peephole/WorkPackage_4/Peephole/android $
./test.tcl
2002       0x20        2 0000
9003       0x90        3 000100
9003       0x90        3 000200
9003       0x90        3 000300
9003       0x90        3 000400
9003       0x90        3 000500
9003       0x90        3 000600
```

The simulator is now ready to respond to commands sent by the IFOYD app.

7 IFOYD APP SOFTWARE DESCRIPTION

In this section, we will describe first the requirements of the demonstrator. Then we will describe the software, in the configuration where the IFOYD app is connected to the simulator as it allows monitoring the messages exchanged.

7.1 Demonstrator requirements

A requirements analysis has been conducted for all demonstrators in the first phase of the project. The result of the analysis has been documented in a report that has been submitted as an annex for the mid-term review ("Annex 2").

For the Peephole Surveillance demonstrator, the following requirements have been identified. In the following table, we provide the name of each requirement, and indicate if this requirement has been covered by the implementation or not. In case the requirement was not implemented, we justify the modification of the development plan in the last column.

ID	Name	Description	Achieved
REQ001	Connect to WiFi hot spot	Connection to the local network/Internet	Yes, Broker on the EoT device
REQ002	Connect to the PC or Smatphone app	Connection to the app	Yes
REQ003	Module set-up	Set cloud address, login, password, application	Yes
REQ004	Start operation	Start the app on the module	Yes
REQ005	Stop operation	Stop the app on the module	Yes
REQ006	Status	Send the module status: • Active /Standby • Event detected or not • Last events • Memory used/remaining • Battery level • I/O status • Camera status (average light level)	Partially (Not implemented: Memory used, battery level, camera status – justification: not needed)

REQ007	Take photographs	Take a picture with the camera and send it over wifi	Yes
REQ008	Loop recording at 1 fps	Record 60 frames at 1 fps in a circular buffer	Yes
REQ009	Circular buffer freezing	Freeze the circular buffer	Yes
REQ010	Recording @ 25 fps or 12 fps	Record during the alarm duration	Yes
REQ011	Send an alarm to the cloud app	Send the alarm flag and alarm type to the cloud server	Partially – no cloud server used, but direct connection with the companion device
REQ012	Send the picture of the alarm	Send the picture corresponding to the triggering of the alarm to the cloud server	Partially – no cloud server used, but direct connection with the companion device
REQ013	Send the pre-alarm buffer	Send the circular buffer to the cloud server	Partially – no cloud server used, but direct connection with the companion device
REQ014	Send the post alarm buffer	Send the post alarm buffer to the cloud server	Partially – no cloud server used, but direct connection with the companion device
REQ015	Send the live video	Send the live video to the cloud app	Partially – no cloud server used, but direct connection with the companion device
REQ016	Return to standby mode	• Transfer all buffers to the cloud server • Clear buffers • Resume circular buffer recording	Yes

REQ017	Presence detection	Detect one or several objects moving in front of the camera. Size of the objects to be determined (1 person at 5 meters)	Yes, face detection when a person is in front of the camera
REQ018	Tampering detection	• Detection of a dark picture for more than 5 sec • Detection of a partially occulted picture for more than 5 sec • Detection of a highly blurred picture for more than 5 sec	Yes
REQ019	Face detection	• Detect the presence of a face with a width ranging between x and y pixels (tbd) • Detect a face oriented between + and – X degrees horizontally (tbd) • Detect a face oriented between + and – X degrees vertically (tbd)	Yes
REQ020	Face contrast	Detect a face with a minimum contrast of (10%) of the full scale level (tbd)	Yes
REQ021	Scene Illumination	Operation with a minimum illumination of 1 lux on the scene	Not tested
REQ022	Face thumbnail extract and send	Extract a thumbnail image of the face detected in the picture to the cloud server	Yes (direct connection)
REQ023	Upload face patterns to the module	Upload a list of known face patterns	No (optional scenario, not implemented)

REQ024	Send positive face recognition event	In case a facial match has been detected, send the event, face thumbnail and ID to the cloud server	No (optional scenario, not implemented)
REQ025	Start bi-directional audio	Start audio communication with the module	No (optional scenario, not implemented)
REQ026	Stop bi-directional audio	Stop audio communication with the module	No (optional scenario, not implemented)

7.2 Software description

The IFOYD app was developed in Java. At high level, it is composed of the main module, called MainActivity.java, and the MQTT client, called MQTT_Client.java

The arborescence of the IFOYD project is as follows:

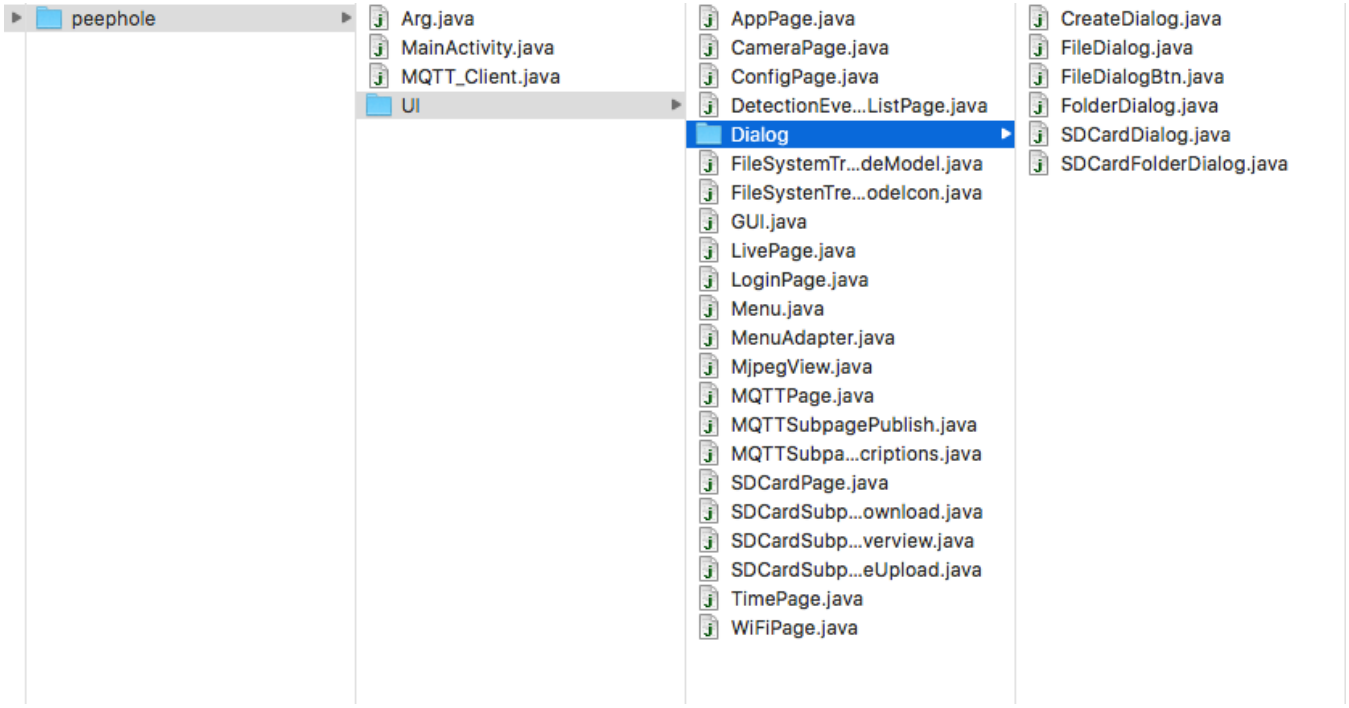


Figure 13: Arborescence of the IFOYD project

All User Interfaces are located in the subfolder UI. Modules corresponding to each of the functions can be called through the interface.

The following table shows the list of the controls and the corresponding modules:

Function/Control	Module
Main App	MainActivity.java
Start page	AppPage.java
MQTT client	MQTT_Client.java
Configuration	ConfigPage.java
Set Clock	ConfigPage.java
Enable/Disable Live Video Streaming	LivePage.java
Activate/Deactivate Face Detection	ConfigPage.java
Live Detection Events	DetectionEventListPage.java
Request Event List	DetectionEventListPage.java
Response Detection List	DetectionEventListPage.java
Request: Frame of a Detected Event	DetectionEventListPage.java
Response: Frame of a Detected Event	DetectionEventListPage.java
Live Video Streaming	LivePage.java
SD Card management	SDCardPage.java

8 IFOYD APP SOFTWARE DOCUMENTATION

The results of the different functions available from the IFOYD app are displayed below.

8.1 Launching the IFOYD App

When the IFOYD app is launched, the following screen appears.

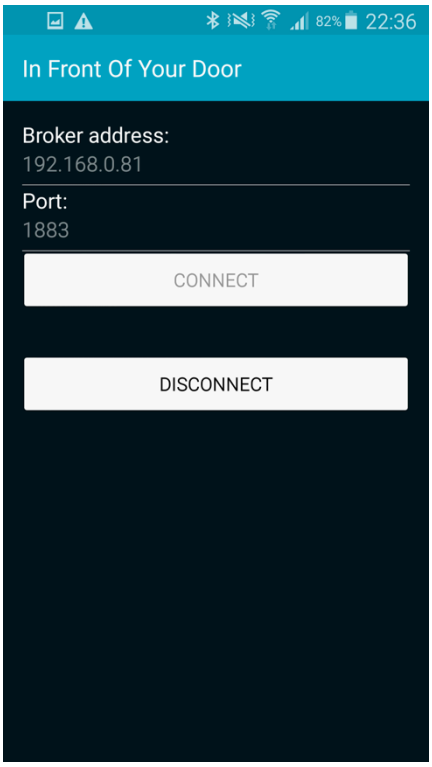


Figure 14: Configuration screen after the launch of the app

The first task is to select the IP address of the EoT device, in our case, 198.168.0.81. The port is already configured on its actual value: 1883. Then the user pushes the “CONNECT” button. If the connection is successful, the “CONNECT” button becomes grey and turns inactive, meaning that the connection is already active. Pushing the “DISCONNECT” button will stop the connection. To come back to the main menu, the Android “RETURN” button has to be used (bottom right below the screen).
No security feature has been developed at this stage. Security relies on access to the Wifi access point of the EoT device which is protected by a password. Additional security could be provided by hiding the ssid of the device.

On the simulator side, the following messages are displayed:

```
./test.tcl
2002       0x20        2 0000
9003       0x90        3 000100
9003       0x90        3 000200
9003       0x90        3 000300
9003       0x90        3 000400
9003       0x90        3 000500
9003       0x90        3 000600
3029       0x30       41 000c636c6f636b436f6e74726f6c7b22
```

8.2 Main menu

The main page gives access to the different functionalities:

- **Login** gives access to the page seen in section 8.1
- **Configuration** gives access to clock setting, face detection ON and OFF (see section 8.3)
- **Camera** gives access to the generated (see section 8.4)
- **Live** shows the live stream of the camera (see section 8.5)
- **Detection Event List** gives the last events with a picture of each event. The recorded video can be downloaded and played back. (see section 8.6)

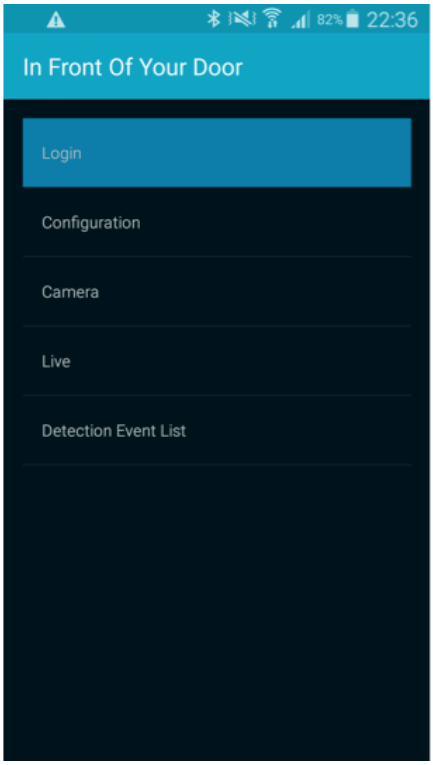


Figure 15: IFOYD main menu

The monitoring screen of the simulator keeps unchanged.

```
./test.tcl
2002       0x20        2 0000
9003       0x90        3 000100
9003       0x90        3 000200
9003       0x90        3 000300
9003       0x90        3 000400
9003       0x90        3 000500
9003       0x90        3 000600
3029       0x30       41 000c636c6f636b436f6e74726f6c7b22
```

8.3 Configuration

The configuration page gives access to three functions:

- **SEND CLOCK** will send the Smartphone time to the EoT device to setup its clock. As the Android Smartphone synchronises its clock through the network or using GPS, it can be considered as a reference.
- **START FACE DETECTION** will launch the detection of faces by the EoT device and the generation of cropped images.
- **STOP FACE DETECTION** stop the continuous face detection process.

Figure 16: Configuration page of the app

On the simulator side, the following commands are received to the **SEND CLOCK** command:

```
./test.tcl
2002       0x20        2 0000
9003       0x90        3 000100
9003       0x90        3 000200
9003       0x90        3 000300
9003       0x90        3 000400
9003       0x90        3 000500
9003       0x90        3 000600
3029       0x30       41 000c636c6f636b436f6e74726f6c7b22
clockControl {"currentClock":1527626417}
currentClock 1527626417
```

The message corresponding to the **START FACE DETECTION** command is the following:

```
3020       0x30       32 000b66616365436f6e74726f6c7b2263
faceControl {"command":"start"}
faceControl start
```

The message corresponding to the **STOP FACE DETECTION** command is the following:

```
stream::one 1528060694537 5127
301f       0x30       31 000b66616365436f6e74726f6c7b2263
faceControl {"command":"stop"}
faceControl stop
stream::one 1528060694617 5617
stream::one 1528060694697 5128
stream::one 1528060694777 5156
stream::one 1528060694857 5648
stream::one 1528060694938 5633
```

8.4 Camera

The following screen appears when the CAMERA button is pressed.
Cropped images of the faces are sent by the EoT Device and displayed on the screen.
At date of this report, a new version of the app is under development which will keep a module listening to the broker as a background task and when a notification is received, the corresponding image will be automatically displayed.

Note that the camera images in Figure 18, 19 and 20 have been generated during a test of the Configuration App using the Simulator running as EoT Device. It has been acquired using a standard webcam and is therefore not representative of the image quality received by one of the cameras connected to an EoT Device (Awaiba Camera, Sony Camera).

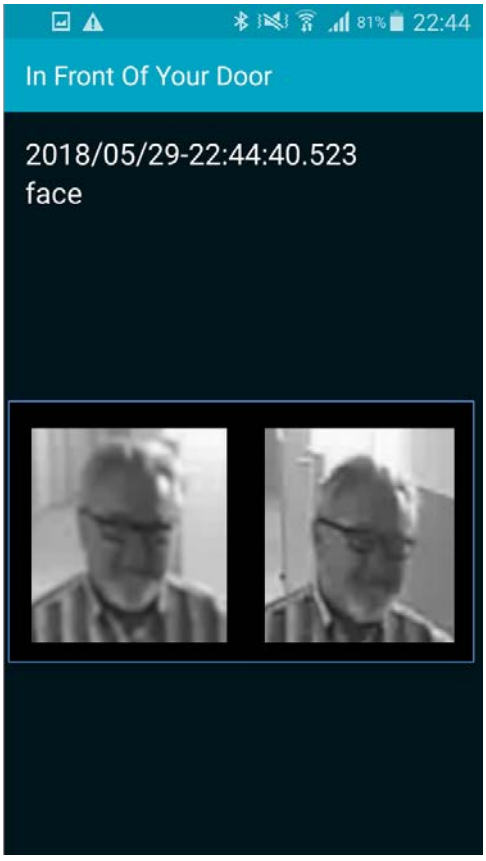


Figure 17: Camera page of the app with cropped images of the collected faces (Configuration app running with the EoT Simulator)

The message received by the simulator is the following:

```
requestFrame {"eventId":212164,"type":"face","frameIndex":0}
requestFrame 212164 0
303d       0x30       61 000c726571756573744672616d657b22
requestFrame {"eventId":212165,"type":"face","frameIndex":0}
requestFrame 212165 0
303d       0x30       61 000c726571756573744672616d657b22
requestFrame {"eventId":212166,"type":"face","frameIndex":0}
requestFrame 212166 0
303d       0x30       61 000c726571756573744672616d657b22
requestFrame {"eventId":212167,"type":"face","frameIndex":0}
requestFrame 212167 0
303d       0x30       61 000c726571756573744672616d657b22
requestFrame {"eventId":212168,"type":"face","frameIndex":0}
requestFrame 212168 0
303d       0x30       61 000c726571756573744672616d657b22
requestFrame {"eventId":212169,"type":"face","frameIndex":0}
requestFrame 212169 0
```

8.5 Live

The live video stream can be displayed (left) or stopped (right) using the command above the video.

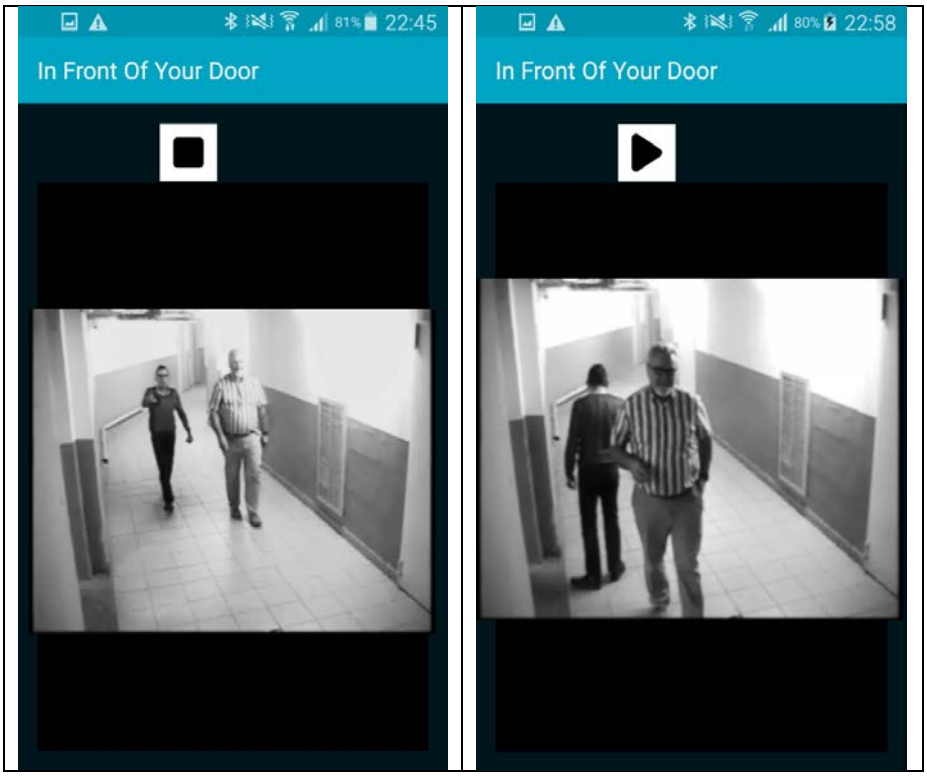


Figure 18: Live page of the app with captured images (Configuration app running with the EoT Simulator)

The commands received by the EoT Device are the following:

```
stream::one 1528062287320 5112
3029       0x30       41 0015766964656f53747265616d696e67
videoStreamingControl {"command":"stop"}
302a       0x30       42 0015766964656f53747265616d696e67
videoStreamingControl {"command":"start"}
stream::one 1528062293121 5646
stream::one 1528062293201 5599
stream::one 1528062293281 5648
stream::one 1528062293362 5108
stream::one 1528062293442 5122
stream::one 1528062293522 5145
stream::one 1528062293602 5632
stream::one 1528062293683 5649
3029       0x30       41 0015766964656f53747265616d696e67
videoStreamingControl {"command":"stop"}
```

8.6 Detection Event list

When the **DETECTION EVENT LIST** button is pressed, the list of events recorded in the EoT device is sent to the IFOYD app. A snapshot of each event is sent with the event and displayed on the Smartphone. A specific event can be selected and replayed. (left image).

Additional controls are available when a clip is replayed (right image) that allow (from left to right):

- back to beginning of the clip
- one image back
- play
- one image forward
- Go to the end of the clip and to the next clip.

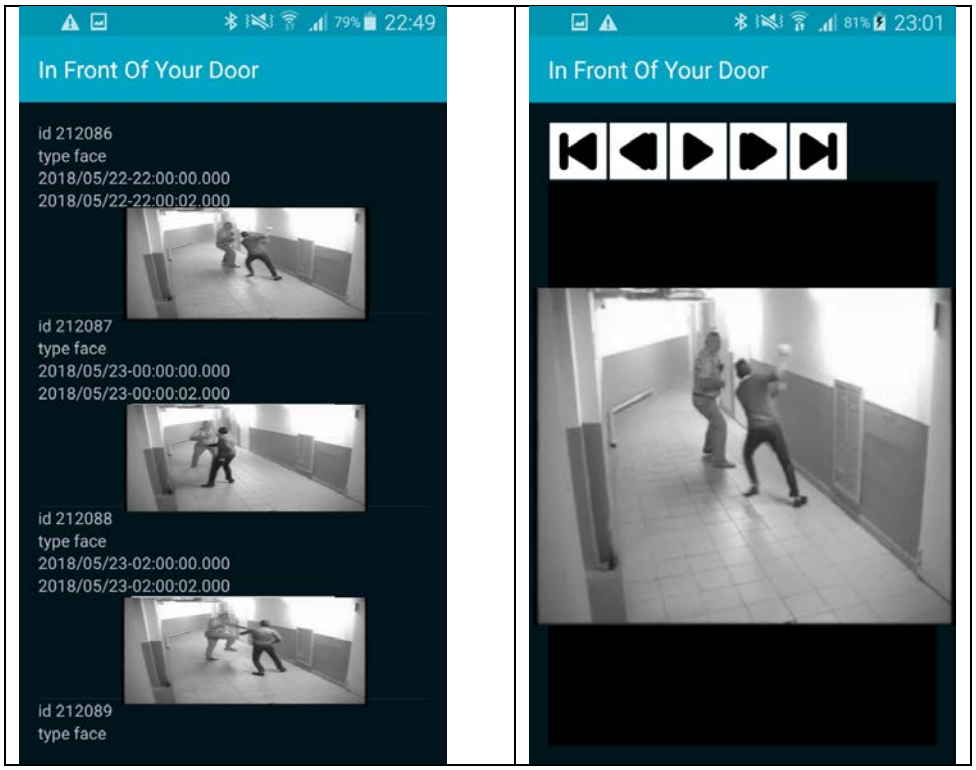


Figure 19: Detection Event List page with replay of stored videos (Configuration app running with the EoT Simulator)

The message received on the simulator is the following:

```
requestDetectionEventList {"timeStart":1527450152,"timeStop":1528054952}
requestDetectionEventList 1527450152 1528054952
84 events
stream::one 1528054952311 5173
stream::one 1528054952404 5621
303d       0x30       61 000c726571756573744672616d657b22
requestFrame {"eventId":212145,"type":"face","frameIndex":0}
requestFrame 212145 0
303d       0x30       61 000c726571756573744672616d657b22
requestFrame {"eventId":212146,"type":"face","frameIndex":0}
requestFrame 212146 0
303d       0x30       61 000c726571756573744672616d657b22
requestFrame {"eventId":212147,"type":"face","frameIndex":0}
requestFrame 212147 0
```

9 CONCLUSIONS

The IFOYD (In Front Of Your Door) Android App has been developed as part of the peephole solution to control the EoT Device and to get alarms, video clips and images from the device. The MQTT protocol used for the communication is easy to implement and customised Apps could be easily developed using the canvas provided by the demonstrator.

Annex 1 : MQTT protocol

EOT Board ↔ IFOYD App Protocol

- Document History

Date	Version-Type	Editor	Changes
15.05.2017	Initial Draft	Stephan Krauß	initial draft
16.05.2017	Updated draft	Thierry Larmoire	Added audio and video stream specification
17.05.2017	Updated draft	Stephan Krauß	Updated video stream specification
15.12.2017	Updated draft	Thierry Larmoire	Added record handling of detection events and video
31.01.2018	Updated draft	Ruben Reiser	Updated specification
22.05.2018	Final draft	Stephan Krauß	Fixed issues in examples and wording

Table of Contents

1. Control ..... ¡Error! Marcador no definido.
Set Clock ..... ¡Error! Marcador no definido.
Enable/Disable Live Video Streaming ..... ¡Error! Marcador no definido.
Activate/Deactivate Face Detection ..... ¡Error! Marcador no definido.
2. Detection Events ..... ¡Error! Marcador no definido.
Live Detection Events ..... ¡Error! Marcador no definido.
Request Event List ..... ¡Error! Marcador no definido.
Response Detection List ..... ¡Error! Marcador no definido.
3. Video Streaming ..... ¡Error! Marcador no definido.
Request: Frame of a Detected Event ..... ¡Error! Marcador no definido.
Response: Frame of a Detected Event ..... ¡Error! Marcador no definido.
Live Video Streaming ..... ¡Error! Marcador no definido.

1. Control

- ## Set Clock

Publisher:
IFOYD App

Topic:
clockControl

Message Payload:
`currentClock (integer 32bit)  seconds since 1 Jan 1970`

Message Format:
JSON

Example:

```
{
    "currentClock":  1480082281
}
```

- ## Enable/Disable Live Video Streaming

Publisher:
IFOYD App

Topic:
videoStreamingControl

Message Payload:
`command  (string)  streaming control command: "start”,”stop”`

Message Format:
JSON

Example:

```
{
    "command":  "start"
}
```

Note:
All detections are disabled if video streaming is enabled.

- ## Activate/Deactivate Face Detection

Publisher:

IFOYD App

Topic:

faceControl

Message Payload:

`command (string) face detection command: "start”,”stop”`

Message Format:

JSON

Example:

```
{
    "command": "start"
}
```

Note:

- Motion-/Tampering- detection is disabled if face detection is active.
- Motion-/Tampering- detection is enabled if face detection is inactivate.
- Default: Motion-/Tampering- detection is enabled.

2. Detection Events

• Live Detection Events

Publisher:

EOT Board

Topic:

detectionEvent

Message Payload:

```
eventId      (integer 32bit) event identifier
type         (string) the type of event (motion, person, face,
             tampering)
timestamp    (integer 64bit) milliseconds since 1 Jan 1970
```

Message Format:

JSON

Example:

```
{
    "eventId": 7,
    "type": "face",
    "timestamp": 1480082281234
}
```

• Request Event List

Publisher:

IFOYD App

Topic:

requestDetectionEventList

Message Payload:

```
timeStart    (integer 32bit) seconds since 1 Jan 1970, query slice
             start (included)
timeStop     (integer 32bit) seconds since 1 Jan 1970, query slice
             stop (excluded)
```

Message Format:

JSON

Example:

```
{
    "timeStart": 1480082280
    "timeStop": 1480082291
}
```

- ## Response Detection List

Publisher:

EOT Board

Topic:

responseDetectionEventList

Message Payload:

`events (tablearray)`

- `eventId (integer 32bit) event identifier`
- `type (string) the type of event (motion, face, tampering)`
- `fps (integer 32bit) frame rate of the sequence`
- `countFrames (integer 32bit) number of frames`
- `timeStart (integer 64bit) milliseconds since 1 Jan 1970`
- `timeStop (integer 64bit) milliseconds since 1 Jan 1970`

Message Format:

JSON

Example:

```
{
    "events":[
        {
            "eventId": 7,
            "type": "face",
            "fps":15,
            "countFrames": 300,
            "timeStart": 1480082281234,
            "timeStop": 1480082291234
        },
        {
            "eventId": 8,
            "type": "tampering",
            "fps":30,
            "countFrames": 100,
            "timeStart": 1480082281234,
            "timeStop": 1480082285000
        }
    ]
}
```

3. Video Streaming

- ## Request: Frame of a Detected Event

Publisher:

IFOYD App

Topic:

requestFrame

Message Payload:

```
eventId      (integer 32bit) event identifier
type         (string) the type of event (motion, face, tampering)
frameIndex   (integer 32bit) select frame of the sequence
```

Message Format:

JSON

Example:

```
{
    "eventId": 7,
    "type": "face",
    "frameIndex": 0
}
```

- ## Response: Frame of a Detected Event

Publisher:

EOT Board

Topic:

responseFrame

Message Payload:

```
image        (binary) JPEG encoded image
```

Message Format:

BINARY

Example:

FF D8 FF E0 00 10 4A 46 49 46 …

Note:

Returns an empty message if no image was found.

- ## Live Video Streaming

Publisher:

EOT Board

Topic:

videoStream

Message Payload:

```
image        (binary) JPEG encoded image
```

Message Format:

BINARY

Example:

FF D8 FF E0 00 10 4A 46 49 46 …

Note:

Messages (frames) are sent if the video streaming is enabled.

- End of document -