Escuela Politécnica Superior de Linares

Máster en Ingeniería de Telecomunicación

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Trabajo Fin de Máster

Desarrollo de una Interfaz de usuario en Android de un Visor Móvil de Procedimientos (mobiPV)

Alumno: Luis García Millán

Tutores: José Manuel Pérez Lorenzo
Ulrich Birkel

Depto.: Ingeniería de Telecomunicación

Octubre, 2017

ACKNOWLEDGMENTS

I would first like to thank my supervisors. Antonio Fortunato, who gave me the opportunity of being part of mobiPV project and finish it. José Manuel Pérez Lorenzo for your trip to EAC, your help to review this text and some good ideas. Ulrich Birkel for your visit and your general advice during this time.

I would also like to thank Aidan Cowley for giving me the opportunity of belonging to the incredible team Spaceship EAC. Mikael Wolff for your experience and kindness. David Martínez Oliveira, without your help to understand mobiPV these months, the app development would have been quite harder.

I would also like to acknowledge all the people I shared time with during my internship. You made this time feel like I was at home. Thank you EAC community, specially to Stephen Ennis, Victor Manuel García Alcaide, Mario Fernández Palos, Manfred Thallner, Sarah Eriksson, Maxim L. Mommerency, Íñigo Muñoz Ipotxa, Andrea Casini, Manel Vera Palou, Alanah Knibb, Leonie Beck, Theo Bidard, Alessio Cataudella, Angelo Bellofiore, Oriane Garcia, Patrick Fleith, Paul Hedderman, Sander Coene, Stefan Siarov, Victoria Nash, Hugo Villeneuve, Adelaide Thomas, Stijn Thoolen, Guillermo Rojo Gil, Jan Mueller, Louis Maller and Victoria Da-Poian.

I must express my very profound gratitude to my father, Luis José García García, your advice taught me that there is no ceiling and I am the one who decides what is my limit. To my brother, Jorge García Millán, you are always there for whatever I need and you taught me that family is above all else. To my life partner, Gema Díaz Quesada, without your encouragement I would not be able to fly that far, you are always there giving an unconditional support in good times and bad. To the rest of my family, without you I would not be the person I am now.

Finally, this work is for you, mom, you will always live in my hurt

Resumen

El trabajo de los astronautas en la Estación Espacial Internacional (ISS) consiste en llevar a cabo un amplio rango de tareas al mismo tiempo que siguen unos manuales llamados procedimientos. Los procedimientos les ayudan paso a paso a realizar esas tareas y pueden tener tanto formato digital (seguidos con el uso de portátiles) o formato analógico (seguidos usando documentos físicos). La mayor parte del tiempo, la zona de trabajo donde los astronautas llevan a cabo sus tareas no suele ser cercana al lugar donde se encuentra el portátil adjunto a la pared, o el lugar no es suficientemente accesible para llevar los documentos físicos. En este contexto, el proyecto mobiPV nace para crear un visor de procedimientos que da al astronauta la posibilidad de trabajar con un dispositivo Android adjunto a su antebrazo que les ayuda a ahorrar tiempo y al mismo tiempo crea un espacio de trabajo cooperativo con el equipo de soporte que está en la Tierra

Este proyecto mejora la solución para el dispositivo Android implementando una Interfaz de Usuario que reemplaza la antigua aplicación. Esa antigua aplicación consiste en una app que actúa como un navegador web. Esta nueva aplicación Android mejora la experiencia de usuario haciendo un mejor uso de las principales características de Android e integrando las características del software mobiPV. La app también mejora la eficiencia evitando el uso de algunos componentes de mobiPV como mobiWebGW y el WebSocket para simplemente estar conectada a un componente de control llamado mobiControl y creando un Socket TCP.

El resultado prueba que más y más tecnologías espaciales pueden ser adaptadas a la plataforma abierta Android. Esta app comienza un camino que puede no tener límites en relación a la integración de proyectos y tecnologías en este visor de procedimientos.

Table of contents

List of figures

List of tables

List of Abbreviations

ADT
Android Development Toolkit, 33
AVD
Android Virtual Device, 33
EAC
European Astronaut Centre, 5
GUI
Graphical User Interface, 15
IDE
Integrated Development Environment, 28
MPU
Main processing Unit, 9
NDK
Native Development Kit, 28
NEEMO
NASA Extreme Environment Mission Operations, 3
OS
Operative System, 7
OVF
Open Virtualization Format, 31
RDP
Remote Desktop Protocol, 32
TOC
Table of Contents, 14
VR
Virtual Reality, 8

1 Introduction

The mobiPV project is thought to help astronauts working in the International Space Station (ISS) when checking the procedures in order to perform their daily work.

The procedures are manuals which describe step by step the different tasks that the astronauts have to do when performing any kind of job. These procedures can be in different formats as paper format or digital format including written text, images or videos.

Problems begin when some of the tasks that are done require working in a narrow space inside the ISS or having both hands free. This is not possible when the astronaut has to be checking continuously a book or fixed laptop on the wall.

As demonstrated by [1], mobiPV implements the solution to that problem by creating a wereable and hands-free environment to follow the procedures in a comfortable way. MobiPV also includes a cooperative environment to allow members from the ground station work together with members of the crew inside the ISS. So with the full schedule that astronauts have in the ISS, the time saving is obvious because they do not have to be continuously moving from the work place to the location of the written manual or laptop and can cooperate on live with the crew members on the ground control centre.

MobiPV is compound by software and hardware components. The hardware part is a smartphone attached to the forearm of the astronaut, an eye camera and a microphone and headphones and also an iPad nearby for supporting, in the crew area. On the other side, the ground members have a laptop with which they follow crew work. The software is running in a Linux OS; the smartphone connects to a WebApp in the ISS server which is linked to the ground. On the ground side the laptop is also running Linux where mobiPV is installed.

With the current state of art, it could be possible to use mobiPV, in fact it has been successfully tested three times, one of them in the ISS, although these tests have proved that the final experience using the smartphone could be improved by creating a native code Android app. The task of this project is to develop an Android app for the crew members of the ISS to handle the mobiPV software. Instead of running a WebApp which is not very comfortable in a small screen as the one in a smartphone, the best solution is to create a native software for this device using android which achieve a better final experience for the user.

This Android app has to be able to open a connection with the mobiPV control process through TCP/IP sockets and, by sending commands to it, to parse the responses to perform the different functionalities. This control process is responsible of sending the different commands that arrive to the rest of the processes of mobiPV. These other processes controls functionalities such as the login function, the procedure viewer, the multimedia exchange, the chat, the state of the different screens in the application, etc.

For the development of this task a Windows laptop is going to be used, in this hardware, a virtual Linux machine will be working with mobiPV running on it and an Android virtual machine is also going to be running and there is here where the new Android app is installed, so the connection of both devices is made with a virtual network between the two simulators and using windows as a meeting point. Finally, the app is going to be developed using Android Studio windows as a meeting point.

1.1 Current state of mobiPV

As mentioned before mobiPV project is fully working and in the next figure (Figure 1) the main components are shown, which are the smartphone, the wearable, a laptop and a supporting Table.

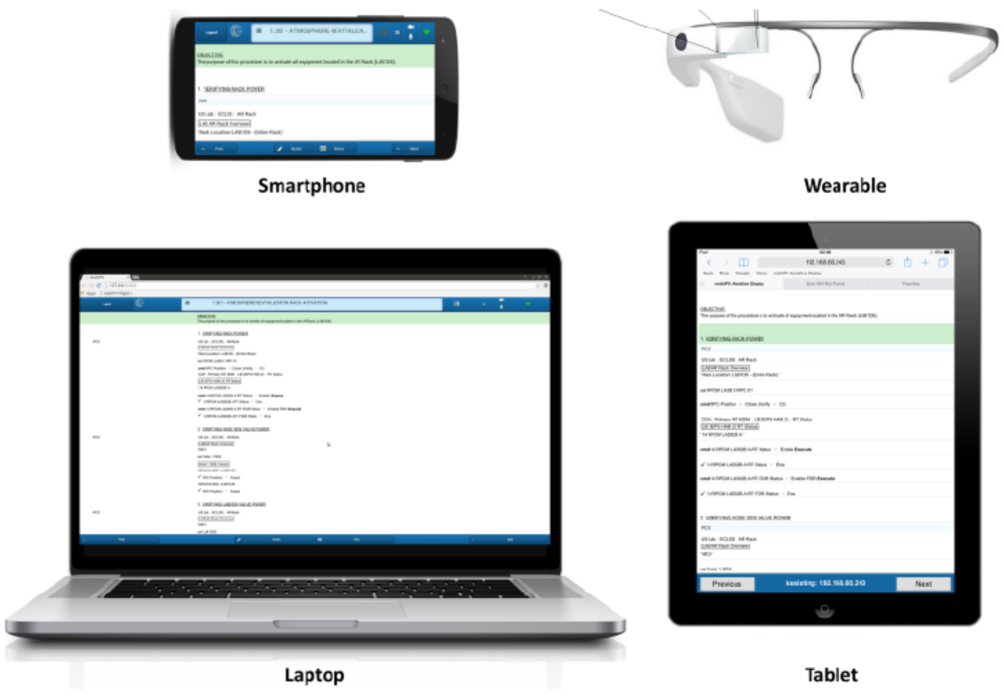


Figure 1: MobiPV main devices. From [1]

Two possible configurations for mobiPV have been tested:

- On one hand, the first way can be configured either with the smartphone on the cuff and complements it with Google Glass.
- Otherwise the smartphone is complemented with a head-worn webcam and a simple mono headset with speaker and microphone.

To get to this state several tests have been done in the last years, since 2012, when, mobiPV project was born. Two special user evaluations were run by astronauts and ground members inside Aquarius, an underwater habitat twenty miles off the coast of Florida, in the framework of the NASA Extreme Environment Mission Operations (NEEMO) program. A first test was conducted in summer 2014 during NEEMO 19, followed a year later by a second one during NEEMO 20. Another special user evaluation was made in the ISS in September 2015.

As detailed explained in [2], NEEMO program sends groups of astronauts, scientists and engineers to live to a maximum of 3 weeks to Aquarium, the unique research environment under the sea level. It is operated by Florida International University and is located 5.6 kilometres off Key Largo in the Florida Keys National Marine Sanctuary. In this environment aquanauts are able to simulate a great number of test as if they were staying in the surface of an asteroid.

The very first big test of mobiPV was precisely in NEEMO 19. For this set-up a delay of ten minutes was implemented. Its results were really encouraging, the chosen configuration was the smartphone, the Google Glass acting as head camera and the IPad providing an assistive display. The activity that the aquanaut had to do followings the procedure using mobiPV was Skin-B. With the help of this experiment [3], the aquanauts are able of measuring the ageing of the skin, which is slow on the Earth but is accelerated in the space due to the different gravity conditions.

In Figure 2 and Figure 3 the Danish astronaut Andreas E. Mogensen is performing Skin-B experiment while wearing Google Glass, in Figure 2, and the ESA astronaut trainer Hervé Stevenin is wearing the complete set up with the attached smartphone in the forearm included, in Figure 3.

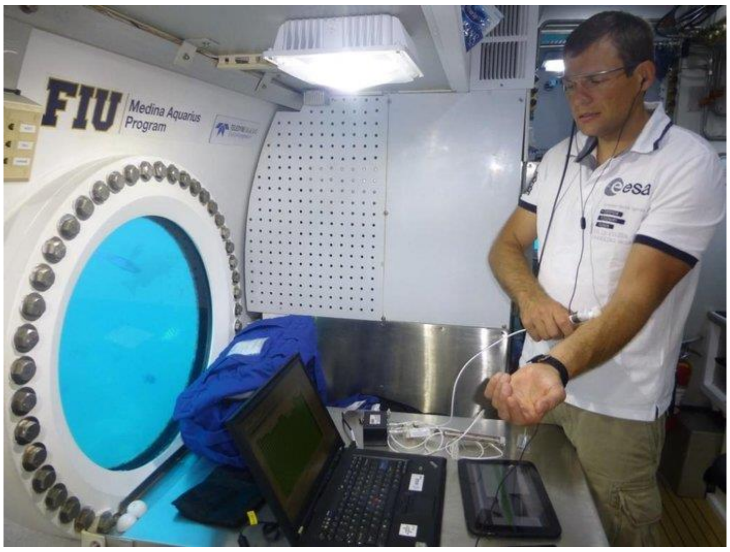


Figure 2: NEEMO 19 Aquarius underwater habitat. Andreas Mogensen using mobiPV first configuration to perform Skin-B. [4]

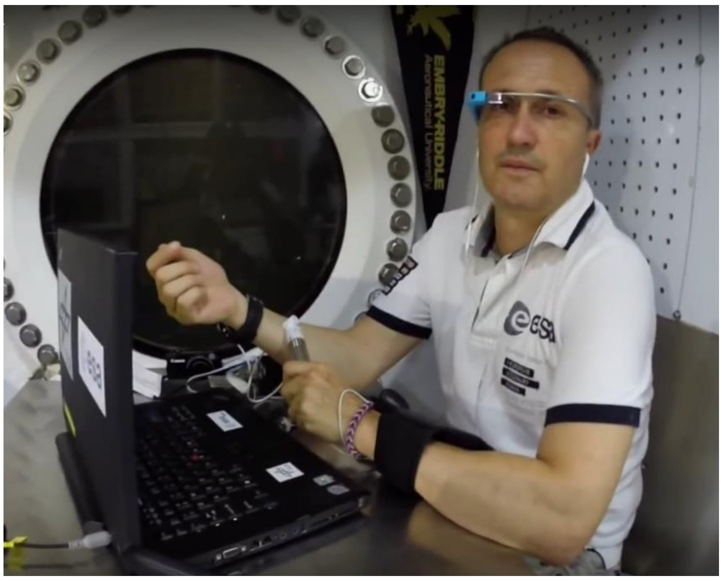


Figure 3: NEEMO 19 Aquarius underwater habitat. ESA astronaut trainer Hervé Stevenin is wearing the complete set up. From [5]

Even though, the results were encouraging, some areas were found to be improved: voice commanding, Google Glass limited battery life and fast heating that made them not really comfortable, as well as the need for a networking protocol that would better cope with limited bandwidth and frequent outages.

Moving on, some months later in NEEMO 20 the Italian astronaut Luca Parmitano, together with NASA astronauts Serena Aunon and Dave Coan evaluated in the summer of 2015 an updated version of mobiPV.

This time the second configuration was chosen, using the smartphone together with a head mounted camera, a wired headset and an IPad. The activity was the same than in the previous session: Skin-B. Figure 4 shows Luca Parmitano testing mobiPV while following Skin-B procedure, in this case the head mounted camera is the main difference from NEEMO 19

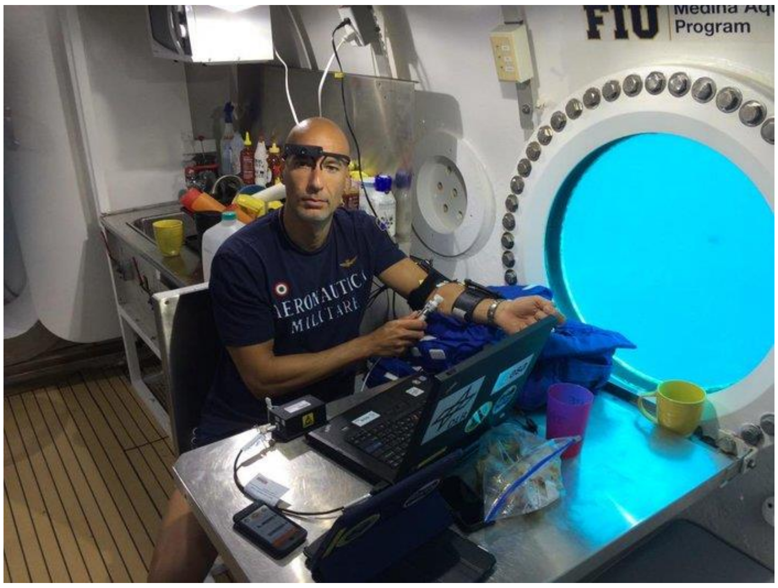


Figure 4: NEEMO 20 Aquarius underwater habitat. Luca Parmitano mobiPV second configuration to perform Skin-B. From [6]

The results were again encouraging, the system software was updated and in this case the verification of a simulated multi-centre collaboration setup was carried out. The aquanauts could perfectly interface with a ground team placed in the European Astronaut Centre (EAC) in Cologne. On the other hand, problems still persisted with the voice recognition and the cables connecting the various hardware resulted to be quite hard to carry.

The last and most important test took place on the ISS. The ISS is a researching environment in low Earth orbit. Its first component, the Russian module Zaryá, was launched on 20th November 1998 and on the 2nd November 2000 the first astronauts started to live and work there. Since then more and more modules have been added becoming the size of the ISS similar to the size of a football field. Many experiments and researching projects have been done taking advantage of the unique microgravity conditions on board of the ISS. The methods of all of these works, documented and followed by the astronauts following the procedures and the procedure viewers, such as the one described in this project, are gaining importance in order to save the time and improve the experience of the astronauts.

Andreas Mogensen during his 10 days' mission called Iriss in September 2015, was again the responsible of testing the improvements and new features of mobiPV.

For the test only the second configuration was chosen, due to schedule problems. The day before, two smartphones were completely charged, in case that one failed and a cuff worn mount was printed using a 3D printer.

Next day Andreas Mogensen tested the device. He connected the wired audio headset to the audio adapter cable jacks and inserted the adapter cable to the smartphone. The head mounted webcam was connected via USB to the smartphone and all the cabling was secured with a Velcro strap to the crew arm to keep it out of the way.

Afterwards, Mogensen started the mobiPV application. The smartphone was set as Master device and on ground the terminal instantly configured itself as slave. When Mogensen started to navigate through the procedure, the ground terminal showed perfectly in which point he was. The additional iPad was also synchronized. This moment can be seen in Figure 5.

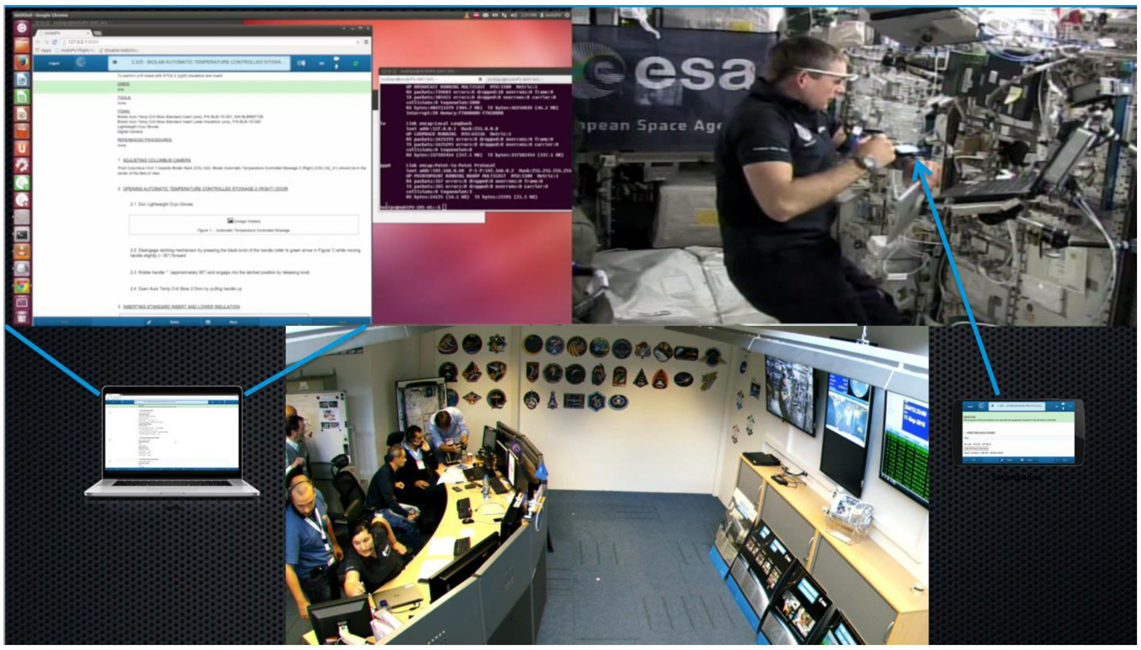


Figure 5: Live space to ground mobiPV operations. Ground terminal and ISS devices synchronized. mobiPV software running on Linux is on the top left part of the figure. Andreas Mogensen is on the top right part while testing on the ISS. The ground crew members in EAC in Cologne are on the bottom image. From [1]

Then, Mogensen enabled the speech recognition for testing. This test unfortunately failed due to issues with the wired audio headset that was handed down from a previous ISS experiment, so the speech recognition was shut down again.

The next step was to send text, image and video notes back and forth. Of course, because of the abovementioned audio issue, the crew video note was without audio and we had to skip the recording of audio notes on the crew device. A real time two-way video link was also established between space and ground and crew (shown in Figure 6) were able to watch the flight controllers and vice versa with fairly good quality.

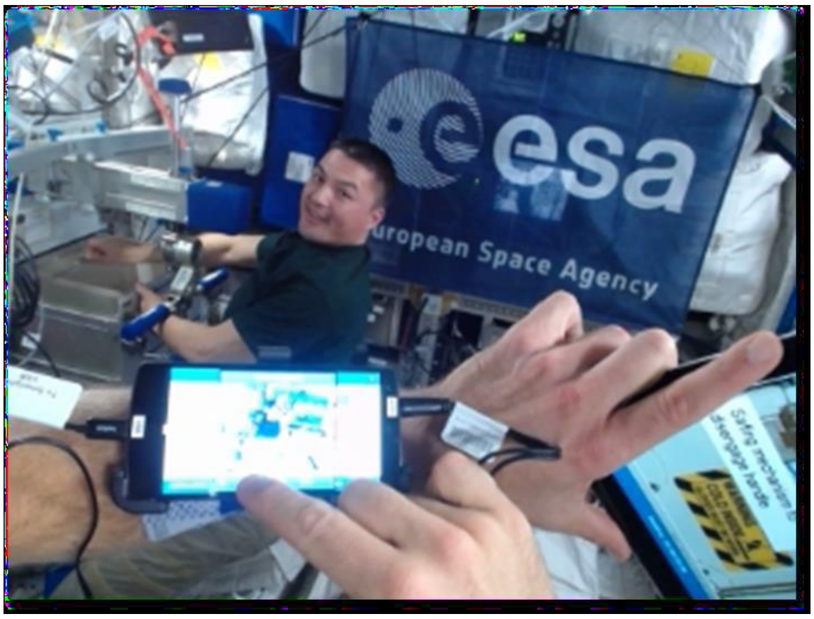


Figure 6: "Astronaut Eye View" from the head-worn webcam. From [1]

1.2 Main targets of this project

After those many tests previously explained, mobiPV team learned that the final user experience when using the attached smartphone by the astronaut could be improved.

As it is going to be explained in deeper in the next section mobiPV works in a Linux Operative System (OS) and creates a web service that can be opened by using a standard web browser. Focussing on the smartphone side mobiPV is installed in the core of the smartphone (in which Linux is running) and then the web service is accessed with a simple Android app that only includes a WebView [7].

At the end, in the smartphone mobiPV is the same than in a laptop but with the only difference that the screen is smaller therefore the quality of the service is also reduced.

With this scenario the creation of a native Android app where the user could use mobiPV becomes necessary. Besides, the visual and easy handling properties of Android can be implemented, resulting in a better and easy experience for the astronaut.

This project is going to answer questions as:

- Whether it is possible to enhance the user experience by adding visual animations or improving the screen resolutions.
- How the Android app is going to interact and communicate the Android app with the core, whether using the same method than the Web App based on WebSockets technology or different one.
- If there is an enhancement with the Android app in relation of the usage of the tools that Android OS provides.

Summing up, as a general objective this project tries to explore new ways of a better adaptation to Android of the mobiPV system, thought to be executed in a wide range of platforms, with the main purpose of be supported in future technology.

Adaptation and opening new paths so space technology does the best of a world spread technology as Android. Proving that the adaptation of mobiPV to an Android can be successful, maybe new space solutions can migrate and have an opportunity with this OS.

In the next sections, both mobiPV original software and Android app will be explained and the results and conclusions will be presented.

2 mobiPV++ System

This sections provides an overall description of the mobiPV++ system as a summary of the contents in [8] and [9]. It will provide also valuable information to the correct understanding of how the communications between the Android GUI and the mobiPV software is achieved.

The mobiPV++ project main goals are to become a distributed platform that combines multiple adaptive systems and is open to the integration of new hardware and software which varies from simple peer to peer communication to Virtual Reality (VR) services. Although the main functionality, as its name indicates, is to provide a mobile Procedure Viewer specially targeted to astronaut operations on the ISS.

The mobiPV++ application, until this moment, provides the following features:

- Possibility of running on different Linux based platforms as GNU/Linux Debian, Ubuntu and Android.
- Execution checklist procedures intended for the evaluation of experimental features.
- Multi-party communications including, peer to peer video conferencing, audio conferencing, and dynamical switch to video sources for the units connected to a conference.
- Short text message communications
- Note taking and its association to procedure steps as audio, video, photo and text notes.
- Collaborative operations between different units in a master slave mode.
- Access to documents (e.g. procedures) in remote devices.
- Other minor functions such as automatic logbook generation, voice commanding, system diagnostics, ...

The mobiPV++ system is focused in achieving a multidevice distributed system in which all the components are interacting through a network. The motivations leading this idea are improving aspects as the software architecture by using modular design; the scalability due to the fact that computers can be added; and the interoperability between different systems.

2.1 Hardware

The hardware is divided into the following elements:

- Main processing Unit (MPU): This is the computer running the mobiPV++ software.
- Video System: The camera enables the user to record the video notes or to perform the video communication.
- Audio System: The audio capabilities use the default audio system provided by the MPU. Moreover, it is recommended to use an external headset to improve audio quality

2.2 Configuration

Three ways of configuring the mobiPV system are available, the Standalone Configuration, the Peer to Peer Configuration and the Multiparty Configuration.

In the Standalone Configuration the unit work on its own without communicating or collaborating with any other.

There are two possible configurations for the hardware as shown in Figure 7:

- A single device usually one smartphone, tablet or laptop where the whole system is deployed and is used autonomously. If there is no network connection the MPU can be used making use of locally stored procedures.
- Dual device. Assistive Display, multiple devices connected to the same wireless network and used to complete the service provided by the main device in case a better screen to check photos and videos or a bigger keyboard to type are needed. For this option the wireless network connectivity is required (see Figure 7).

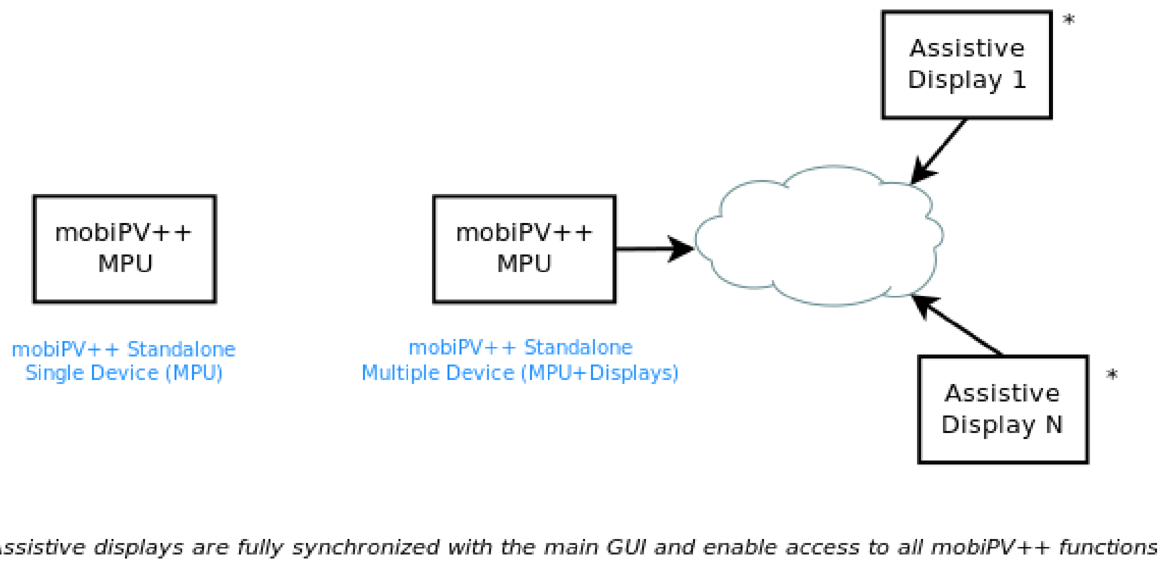

Assistive Display 1
*
mobiPV++ MPU
mobiPV++ MPU
mobiPV++ Standalone Single Device (MPU)
mobiPV++ Standalone Multiple Device (MPU+Displays)
Assistive Display N
*
* Assistive displays are fully synchronized with the main GUI and enable access to all mobiPV++ functions

Figure 7: mobiPV Standalone Configuration. From [9]

In Peer to Peer mode, two users can be using mobiPV++ simultaneously with different units and the collaboration and communication is possible. This Configuration can be seen as the Standalone Configuration plus the possibility of stablishing Peer to Peer Voice and Video communications, sending text messages and collaborating when executing a procedure. In Figure 8 this architecture is shown.

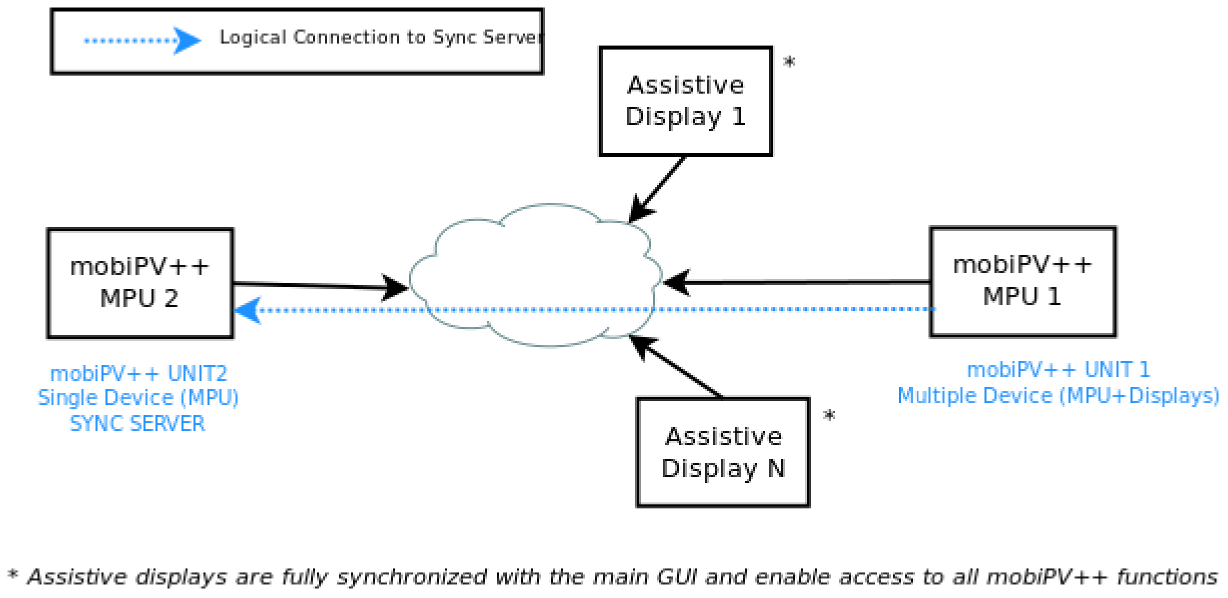

Logical Connection to Sync Server
Assistive Display 1
*
mobiPV++ MPU 2
mobiPV++ MPU 1
mobiPV++ UNIT2 Single Device (MPU) SYNC SERVER
mobiPV++ UNIT 1 Multiple Device (MPU+Displays)
Assistive Display N
*
* Assistive displays are fully synchronized with the main GUI and enable access to all mobiPV++ functions

Figure 8: mobiPV++ Peer to Peer Configuration. From [9]

The third mode is the Multiparty Configuration that includes the last feature: the audio and video multiconferencing.

mobiPV++ has been designed to work in a distributed way. However, for the collaboration, one of the units starts to work as a server and all the information is delivered by this MTU.

In order to maintain the possibility of building different kind of applications, the collaboration and communication capabilities have been split into two different components.

- Synchronisation Server: It is running in one of the units and allows the other units to register themselves. Once a unit is registered, it can query which other units are online, to join ongoing collaboration sessions or to send messages and commands to the others. One example is shown in Figure 9.

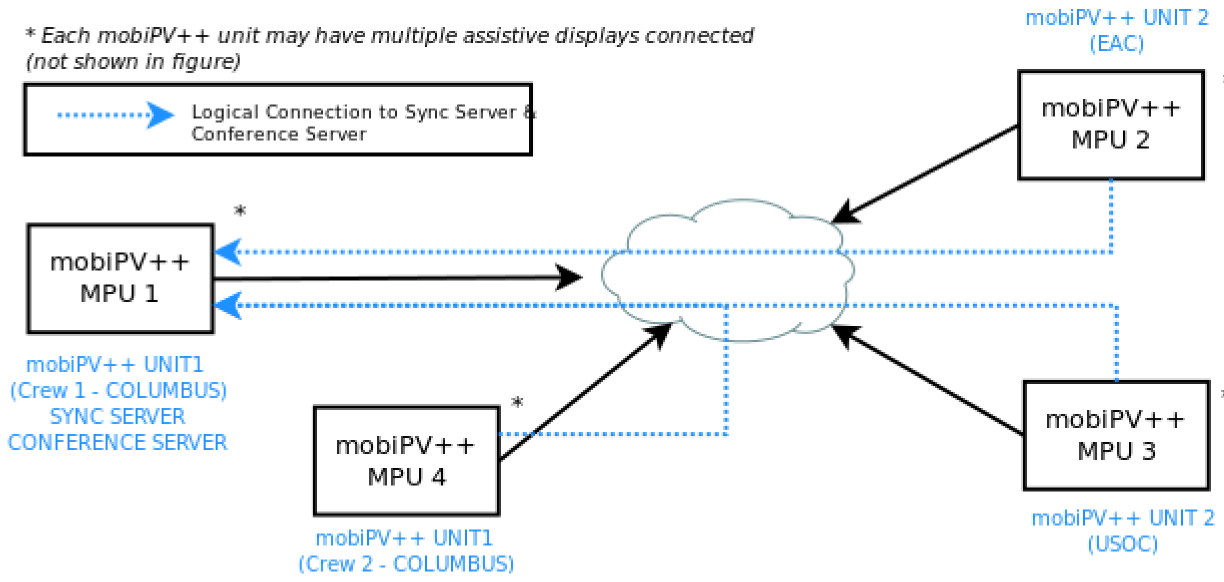

* Each mobiPV++ unit may have multiple assistive displays connected (not shown in figure)
Logical Connection to Sync Server & Conference Server
mobiPV++ UNIT 2 (EAC)
mobiPV++ MPU 2
mobiPV++ MPU 1
mobiPV++ UNIT1 (Crew 1 - COLUMBUS) SYNC SERVER CONFERENCE SERVER
mobiPV++ MPU 4
mobiPV++ UNIT1 (Crew 2 - COLUMBUS)
mobiPV++ MPU 3
mobiPV++ UNIT 2 (USOC)
*

Figure 9:Example of mobiPV++ Multiparty Configuration. From [9]

- The conference server is only used when the multiparty communication is required and can be running in any of the units. This server registers itself in the synchronization server with the name of "Conference" and all the users that start a communication with this special user are able to receive the mixed audio coming from the rest of the users and to send and mix their audio with the rest. Besides, the users are also able to stream their videos and request the video coming from any other participant.

2.3 Architecture

The mobiPV++ architecture has to be described from different points of view. In this section these views will be provided.

Physically the system is built around a set of independent executables interconnected through a network broker. These components can be grouped into:

- Interface Components which communicate with other mobiPV++ components and share an interface to third-party components.
- Internal Components which only interface with other mobiPV components.

Figure 10 shows a sketch of the physical view.

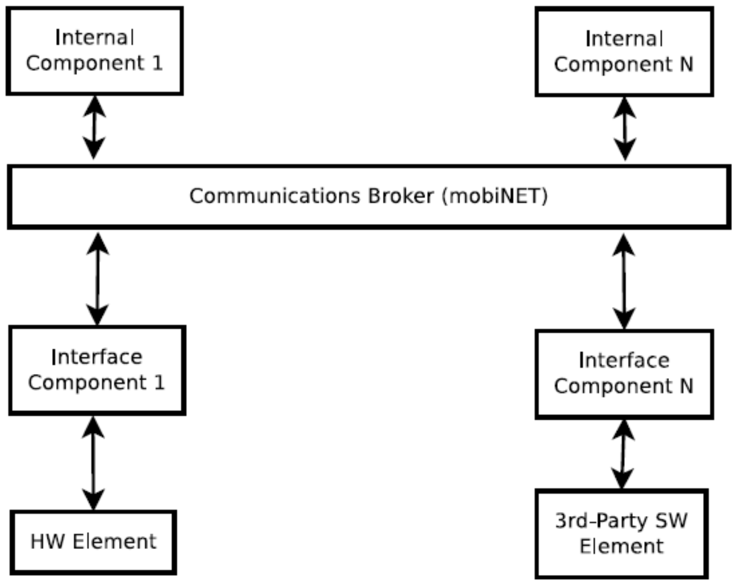


Figure 10: mobiPV++ physical architecture. From [8]

The communication broker is called mobiNet and enables the exchange of messages between the different independent executables that form mobiPV++

With a different point of view, the physical components are also grouped in logical groups that exchange information as shown in Figure 11.

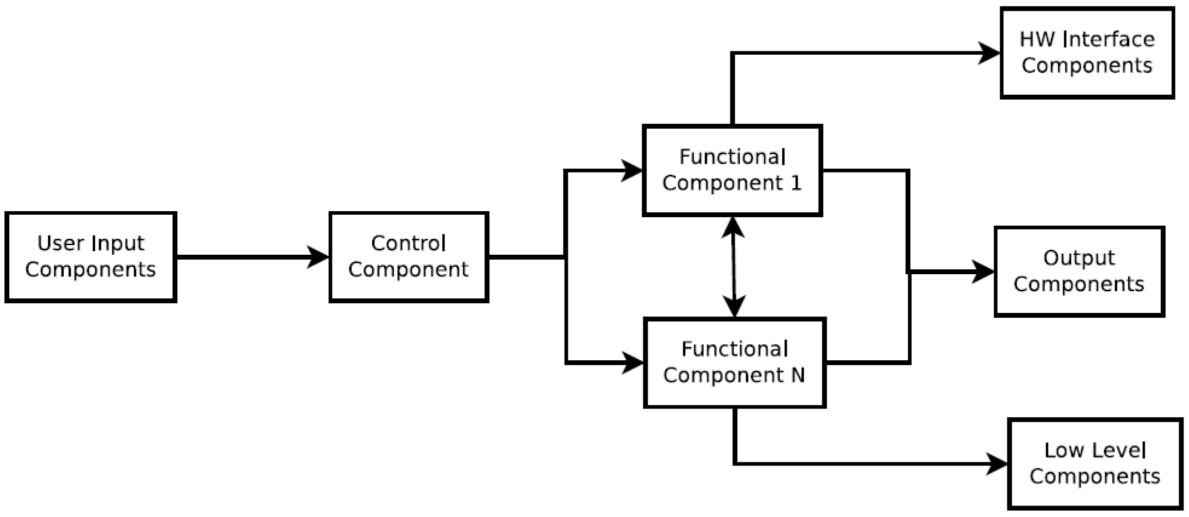


Figure 11: mobiPV++ logical architecture. From [8]

The Control Component is the central element of the architecture and drives the messages, usually coming from a user interface into the appropriate functional component. These functional components group functions from other lower elements into a full function from the user point of view, for example the communication component.

These Functional Components, as mentioned, make use of the low level components and eventually they produce some input that has to be shown to the user through the Output Components. Functionally, the HW Interface components are the same that the low level interface components but they are kept separately because of their dependency on the hardware.

Once the overview of the Functional Components is done, it is time to move into the state machine that is followed by the mobiPV++ system. See State Diagram in Figure 12.

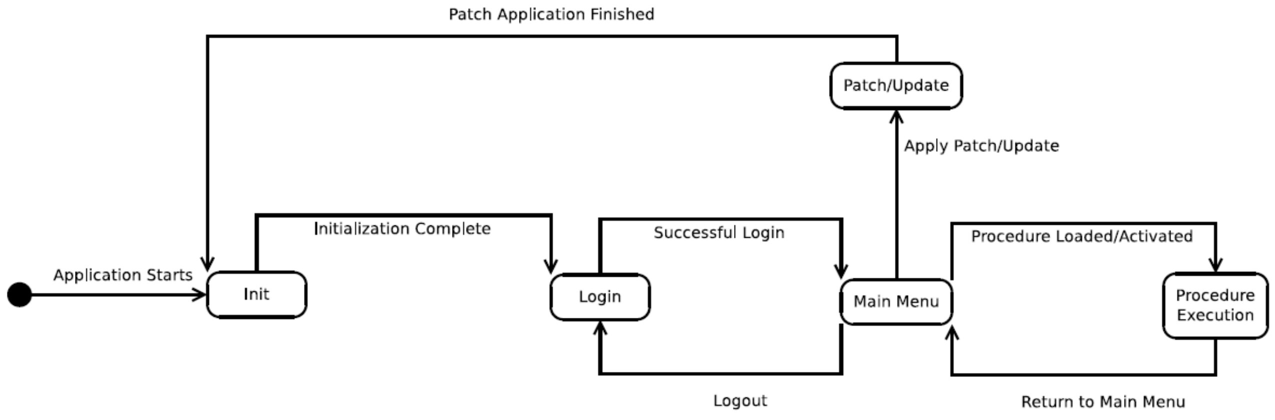


Figure 12: mobiPV++ State Diagram. From [8]

The main characteristics of each state are:

- Init; It is the first state when starting the software, the data structures are initialised and the different components are prepared. When the process is finished the system moves automatically to Login state.
- Login: In this state a log in window is shown to the user asking for a user name and (optionally) a password. When the user is validated, a new mission starts and the system moves into the Main Menu state automatically.
- Main Menu: The system now shows an outfit containing some system level actions related with the user rights. The system level actions include visualize Table of Contents (TOC) for procedure loading, search Document Repositories, view and change settings and access to the diagnosis functions. From this state the user can load a procedure and move into Procedure Execution state.
- Procedure Execution: This state shows the procedure and implements a wide range of functions as the note taking, the step marker, retrieving information about the procedure, go to step, gap filling, etc.
- Patch/Update: In this state the activity is stopped while an update is applied. Once finished the system will move into Init state.

Talking about the platform architecture, as mentioned above mobiPV++ system is designed to be deployed on as many platforms as possible. For this reason, the implementation has been chosen to be GNU/Linux based, that allows the implementation in PCs and facilitates the installation on many embedded systems and Android, as from a low level point of view is another Linux system.

When deploying in an Android system an additional app has to be built separately. This Android app enables the use of the multimedia components and implements their libraries. All other components are executed within a chroot environment.

See Figure 13 to understand better how both architectures, GNU/Linus deployment and Android Deployment are implemented.

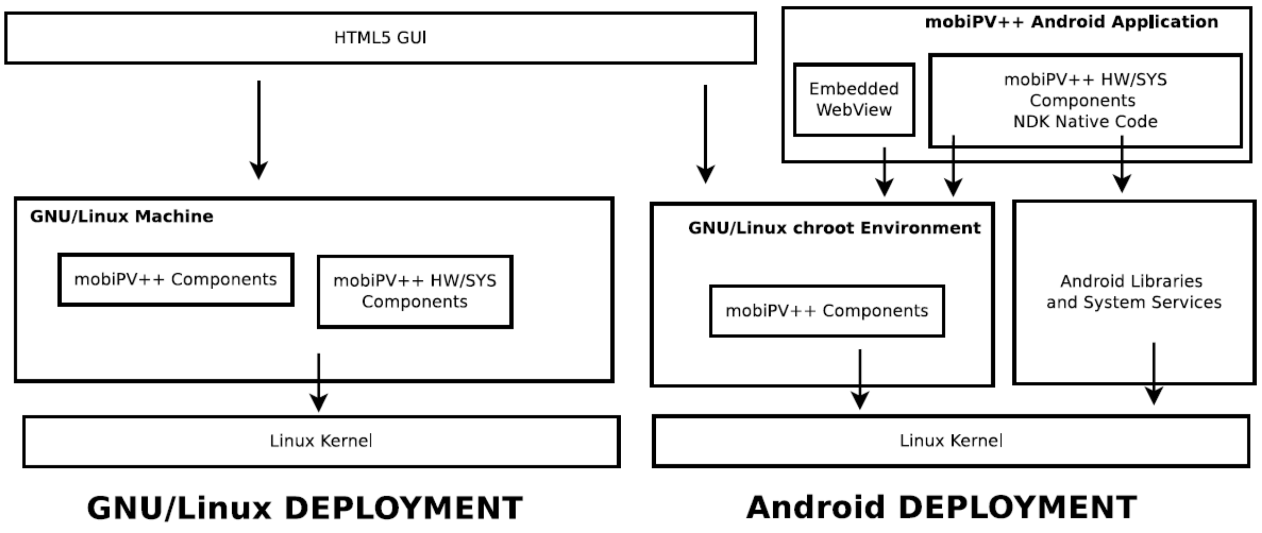

HTML5 GUI
mobiPV++ Android Application
Embedded WebView
mobiPV++ HW/SYS Components NDK Native Code
GNU/Linux Machine
mobiPV++ Components
mobiPV++ HW/SYS Components
GNU/Linux chroot Environment
mobiPV++ Components
Android Libraries and System Services
Linux Kernel
Linux Kernel
GNU/Linux DEPLOYMENT
Android DEPLOYMENT

Figure 13: mobiPV platform architecture. From [8]

On the left part of Figure 13 all the components are running as normal executable on top of a Linux kernel. On the right part the components are executed within a chroot environment running directly on top of the underlying Linux kernel.

The HTML5 Graphical User Interface (GUI) is the only possible GUI when using the GNU/Linux Deployment through a standard web browser. The web browser, through mobiWebGW component, is the responsible of the communication between the user and the mobiPV++ core. The mobiWebGW component is a WebSocket that forwards messages to the core and keeps a track on the answers and messages coming from it. Figure 14 shows the data flow.

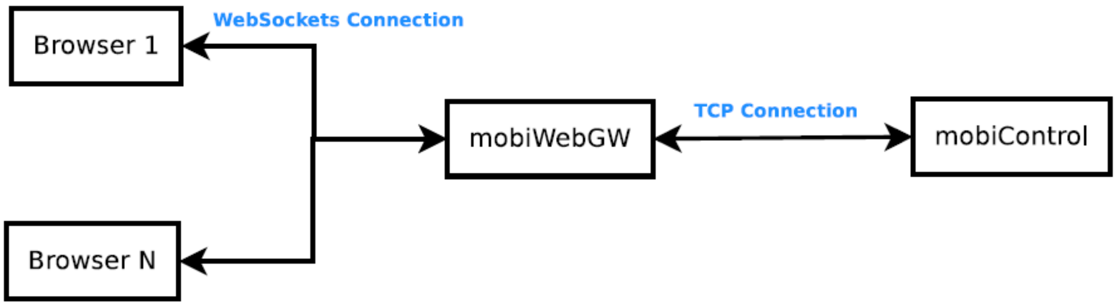


Figure 14: mobiPV++ mobiWebGW data flow into mobiPV++ System. From [8]

In case of the Android Deployment, coming back to Figure 13, the HTML5 can be used as well as the app that implements a WebView. The Android Application also make use of the mobiPV++ components either directly through chroot environment or by accessing Android Libraries and System Services.

2.4 The controller component

One last part has to be explained to better understand the final solution achieved in this project, this is the controller component (mobiControl component)

This is the central component of the mobiPV++ system and has two main functions. The first one is to link the rest of the components and to deliver the messages either coming from the GUI to the components or going from the components to the GUI. The second function is to keep track of the current state of the system. In Figure 12, mobiControl is the responsible of sending the messages in between the states to the required components in order to achieve the transition from one state to other.

In addition of the message delivery and the state transition the mobiControl component also performs these actions:

- Generates a notification when a change of state occurs to allow the other components to configure themselves if required.
- Provides a generic component routing solution to enable sending message to specific components independently of the system state.
- Provides also some commands that are not forwarded to any component such as asking for the software version
- Acts as a multiplexer for the different clients connected to the system (see Figure 15)

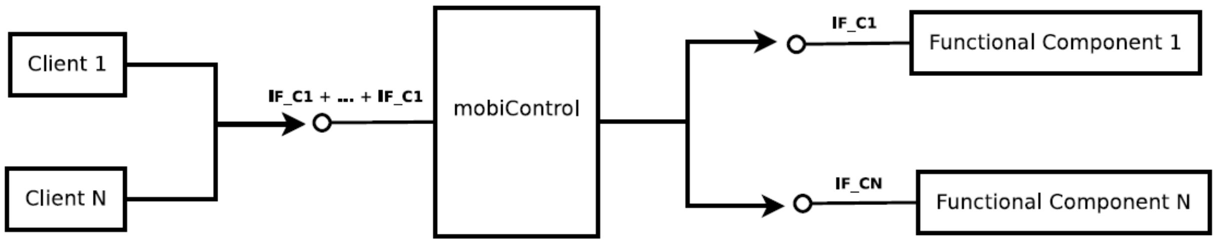


Figure 15: mobiControl as multiplexer for the clients. From [8]

2.4 Control messages

mobiPV implements a method to achieve the control of the system either from the own system for internal features, the web service used within the same software or in an external device, using an assistive display, where the interaction with the user happens or other special device as it is the case of the Android GUI developed in this project.

The purpose of this section is to give an overall guide of the main commands sent to the mobiPV system, the answer of them, the fields included in the texts of the messages, their functions and the state to which they belong, all this information is included in [8].

For example, if the message "connect mobicontrol" is sent to the IP of the device where mobiPV is running and using the port 7777, in case the connection does not fail, mobiPV answers with @RESET, that is the confirmative message and means that the connection is stablished. Table 1 shows the main control commands , the answer in an affirmative case, the function and the state to which it belongs.

Table 1: This table shows the main control messages, the answer in an affirmative case, the function and the state to witch it belongs

Command	**Affirmative answer starts with:**	**Function**	**State**
connect	@RESET	Connect to a certain service of mobiPV	All of the states
get_current	+GET_CURRENT	Asks for the current state of mobiPV	All of the states
get_user_names	+GET_USER_NAMES	Asks for the registered users in the system	Login State
login	+LOGIN	Does login in mobiPV	Login State
logout	+LOGOUT	Does logout in mobiPV	All the states but Login State
get_gui_def	/get_gui_def +get_gui_def	Asks for the list of configuration parameters	All the states but Login State

DIAG_test_resetDB	+DIAG_TEST_RESETDB	Resets the database of mobiPV system	All the states but Login State
RELOAD_TOC	+REALOAD_TOC	Reloads the procedure lists by taking them from iPV server	All the states but Login State
DIAG_clean_cache	+DIAG_CLEAN_CACHE	Cleans the cache of mobiPV system	All the states but Login State
SYS_GUI_SET	+SYS_GUI_SET	Asks core to change one configuration parameter	All the states but Login State
Start	+START	Opens a procedure and move into Procedure Viewer State	Main Menu
GET_OPEN_PROCS?	/GET_OPEN_PROCS +GET_OPEN_PROCS	Asks for the list of currently open procedures in the mobiPV	Procedure Viewer
SET_PROC	+SET_PROC	Selects as active procedure the one written after the command	Procedure Viewer
next	+NEXT	Moves the current step for the procedure to the next step	Procedure Viewer
previous	+PREVIOUS	Moves the current step for the current procedure to the previous step	Procedure Viewer

GOTO_STEP	+GOTO_STEP	Sets the current step for the indicated procedure	Procedure Viewer
GET_PROC_METADATA?	+GET_PROC_METADATA	Asks for the additional information that belongs to the procedure	Procedure Viewer
get_proc_notes?	/get_proc_notes? + get_proc_notes?	Asks for the notes of a specific step	Procedure Viewer

As a continuation of Table 1, Tables 2-11 will show the main fields of every command or its answer that will be useful for the understanding of the presented result in Chapter 3.3 Screens of mobiPV Android User Interface. In every table "ID" will be the different fields following the command and "Description" will contain information for understanding the purpose of the command.

Table 2: +GET_CURRENT affirmative answer parameters

Affirmative Answer: +GET_CURRENT	
ID	**Description**
Identifier	Name of current state

One example for Table 2 would be to receive: "+GET_CURRENT LOGIN", where LOGIN is name of the state in which mobiPV is working

Table 3: login command parameters

Command: login	
ID	**Description**
username	Valid username in the system
password	Password for user "username"

One example for Table 3 would be: “login Crew aaa”, where the user is Crew and the password for this user is aaa

Table 4: /gey_gui_def answer parameters

Answer: /get_gui_def	
ID	**Description**
Type	Type of the entry: BUTTON (3), TITLE (2), VALUE (1), STATE(0)
ID	Configuration parameter identifier
Label	Readable Description
Status	Status for entries of type STATE (OK or FAIL)
Value	Text displayed to the user for this field
Parameter	Configuration parameter value: Can be modified by the user (1), cannot be modified (0)

One example for Table 4 would be: /get_gui_def 1|SYS.BUILD_DATE|Build Date|UNKNOWN|Mar 20 2017-15:32:28|0|, where:

- 1: This parameter is a value
- SYS.BUILD_DATE: Belongs to BUILD_DATE
- Build Date: Text that will appear in the GUI, so the user can read and understand it.
- UNKNOWN: The Status is unavailable
- Mar 20 2017-15:32:28: This is the value that will appear in the GUI
- 0: This value cannot be modified by the user

Table 5: SYS_GUI_SET command parameters

Command: /SYS_GUI_SET	
ID	**Description**
Identifier	Selects the parameter that is to be changed
New value	New value that will take the parameter

One example for Table 5 would be: “SYS_GUI_SET SYS.VIDEO.CAM_RES 320x240”, this command changes the parameter “SYS.VIDEO.CAM_RES”, that corresponds to the camera resolution, to “320x240”.

Table 6: start command parameters

Command: start	
ID	**Description**
Procedure Identifier	Locator of the procedure
File	Path to find it

An example of Table 6 would be: “start nmp /neemo”, where nmp is the identifier for this procedure and /neemo is the path to find it.

Table 7: Answer /GET_OPEN_PROCS? command

Command: /GET_OPEN_PROCS?	
ID	**Description**
proc_uid	Internal unique procedure id
proc_url	URL to the HTML version of the procedure
current	Indicates if the procedure is the currently active one (value 1) or not (value 0)
proc_title	Title of the procedure

An example of Table 7 would be “`/GET_OPEN_PROCS? neemo cache////neemo.html 1 :SKIN B NOMINAL OPERATIONS`”:

- neemo: This is the ID of the procedure
- cache///neemo.html: The web based applications should check this path to find the procedure.
- 1: This is the current active procedure
- :SKIN B NOMINAL OPERATIONS: The title of the produced to be used, for example, in an upper bar in the application.

Table 8: GOTO_STEP command parameters

Command: GOTO_STEP	
ID	**Description**
proc_uid	Internal unique procedure id
step_id	Step id identifier

An example of Table 8 would be: “`goto_step neemo neemo@step_1_1`”

Table 9:#NOTE_ADDED message parameter

Command: #NOTE_ADDED	
ID	**Description**
step_id	Step id identifier

Regarding Table 8, this is not a command but a message that is sent when a procedure is open. An example would be: “`#NOTE_ADDED note_icon_neemo@parts`”

Table 10: get_proc_notes? command parameters

Command: get_proc_notes?	
ID	**Description**
proc_uid	Internal unique procedure id
step_id	Step id identifier

An example of Table 10 would be: “`get_proc_notes? neemo neemo@step_1`”

Table 11: /get_proc_notes? answer parameters

Command: /get_proc_notes?	
ID	**Description**

filename	File name that contains the note (it can be a text, a video, an audio or a photo file)
file location	Path to the file
step_number	Position of the step where this note belongs

An example of Table 11 would be: "/`get_proc_notes? photo_20170816_095253_Crew-0 | 20170816_095253_Crew/photo_ photo_20170816_095253_Crew-0 | 3",` where all the field are separated by a vertical bar

3 mobiPV Android Application development description

The tools and technologies used in this project and the stages that describe the complete process to achieve the goals of it are described in this section.

3.1 Tools and technologies

Usually when talking about technology, there are two different components on it. On one side, there is the physical part, the hardware, and on the other side there is the logical part, the software. Along this section all the hardware and the software needed to the deployment of the Android User Interface is presented.

3.1.1 Hardware

The list of hardware components contains from the USB camera to the laptop used for the development.

In the beginning of the project one laptop (Figure 16) was provided by mobiPV team to perform the development of the user interface.

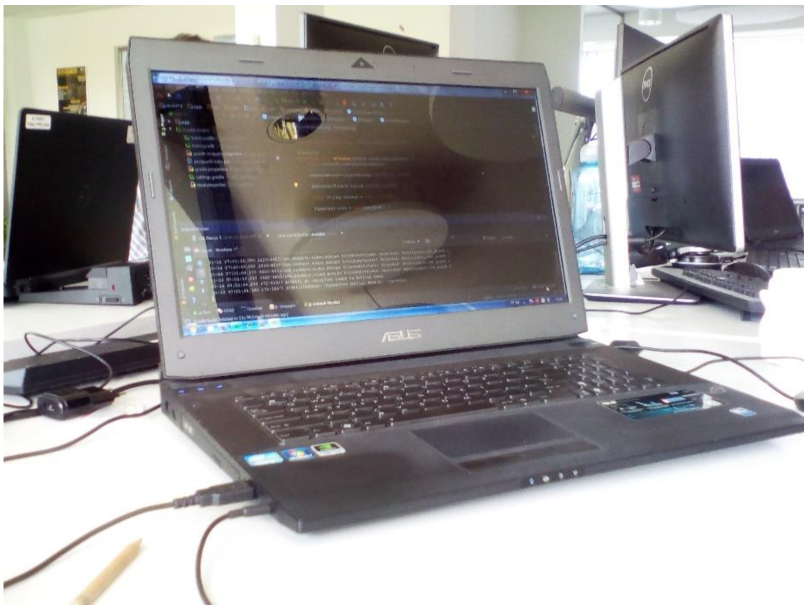


Figure 16: Asus G73S

The main characteristics of this laptop are [10]:

- Operative System: Windows 7
- CPU: Intel Core i7 2630QM
- RAM: 8GB
- HDD: 750GB

In addition to the laptop, later on the user interface was also tested in two smartphones, a Nexus 6 and a Nexus 5 (Figure 17)

Figure 17: Nexus 6 on the left and Nexus 5 on the right

The main characteristics of the Nexus 5 are [11]:

- Operative System: Android 4.4.4
- CPU: Qualcomm Snapdragon 800
- RAM: 2GB

The main characteristics of the Nexus 6 are [11]:

- Operative System: Android 7.0
- CPU: Qualcomm Snapdragon 805
- RAM: 3GB

The last hardware element is the Microsoft LifeCam Cinema (Figure 18)

Figure 18: Microsoft LifeCam Cinema. From [12]

The main characteristics are:

- Recording Quality: HD 720p
- Lens: Wide Angle
- Screen size recording: 16:9

3.1.2 Software

The software used in this project goes from the development tool as Android Studio to the mobiPV software, which was fully detailed in the previous section.

The first topics to be described in this section are the main OS that take an important role during the development of the app. These important OS are GNU/Linux (Ubuntu), as an easy-to-use desktop, and Android.

GNU/Linux is the OS where mobiPV++ runs and that is why this is one of the important for this project. The origin of GNU [13] goes back to 1984 when the Free Software Foundation started developing a free Unix-like OS. The first Linux Kernel was then created in 1991 by Linus Torvald, who is now the coordinator of a big team of programmers and this full team is the system maintainer. The main difference between GNU/Linux and other OS is that it does not have any owner, it is formed by a big community. Other characteristics are that GNU/Linux is a multitasking and multiuser system; users can take a wide range of decisions and the functionalities vary from the simple copy paste to other more professional as writing and compiling programs.

Ubuntu was born [14] (Figure 19) when in 2004 Mark Shuttleworth with a small team of developers released this easy-to-use Linux distribution.

ubuntu

Figure 19: Ubuntu OS. From [14]

Android is the target operative system where this app is going to be installed. [15] Android was bought by Google in 2005, by then it was a small company dedicated to the app development for mobile phones and this year they start the develop of the Java virtual machine optimized for mobiles (Dalvik VM). What make Android different is some special characteristics listed below:

- Open Platform. Platform based on Linux and open code

- Adaptable to any kind of hardware. Android cannot just be found in smartphones and tablets, it is also found in watches, head-mounted displays, TVs and many other different systems. This suppose an additional effort for the developer because of all the different inputs and outputs of all these systems.
- Assured portability: Due to the apps being developed in Java, they can be used in any kind of CPU.
- Internet component based architecture. For example, the user interface is made in XML what allows an app to be executed in a small screen, like the one in a smartphone, or a big one, like a TV.
- Many different built-in services. For example, localization, voice recognition, voice generation, multimedia, etc.
- Low power and low memory optimization. Android uses Dalvik virtual machine, that comes from Java, and implements a system optimization for smartphones.
- High graphical and sound quality.

For a better understanding of the next sections, it is necessary to present Android architecture. (Figure 20).

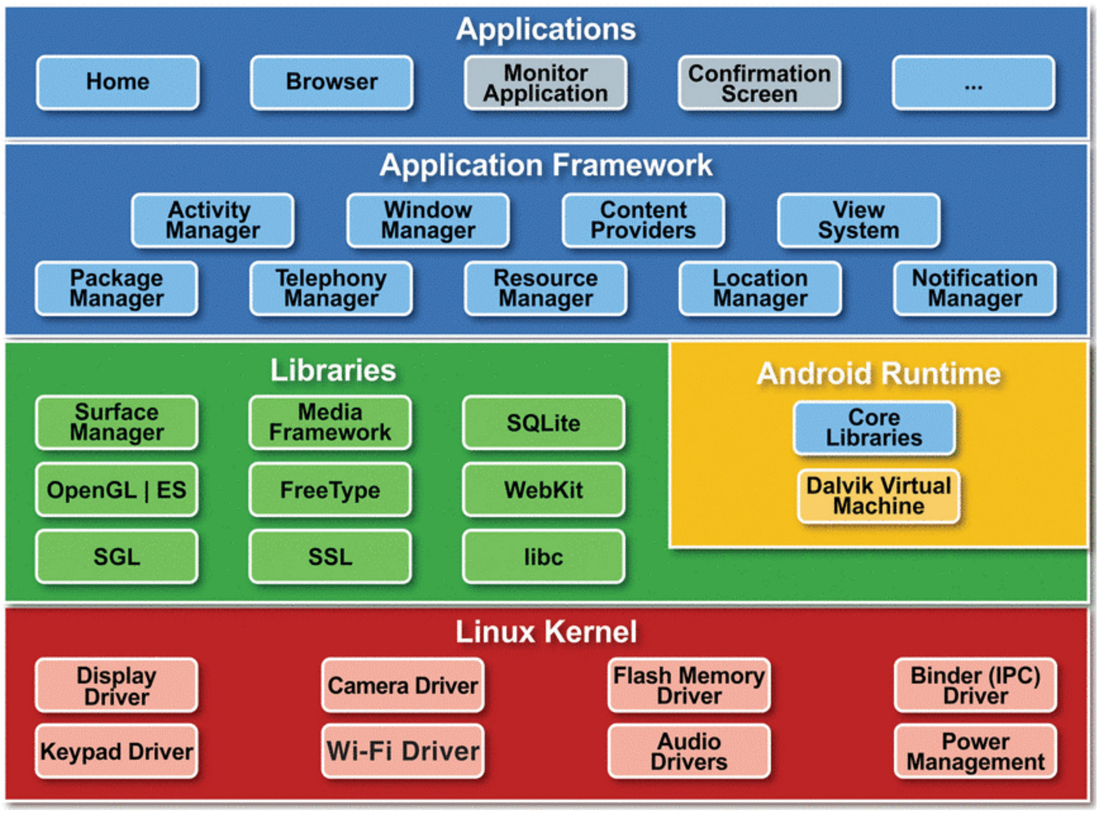


Figure 20: Android architecture

The main components of the architecture from the bottom upwards of Figure 20 are [16]:

- Linux Kernel. This layer provides services such as memory management, security or the hardware drivers. It is the link between the hardware and the rest of the layers.
- Android Runtime. It is based on the Java virtual machine. Among the characteristics for the memory optimization there is the Dalvik files execution (.dex). Every app runs its own process in Linux with an instance in Dalvik virtual machine. The threading and low memory management are delegated to the Linux Kernel.
- Libraries. It includes a set of C/C++ libraries. They are compiled in the processor native code. The main libraries are: Media Framework (implements codecs), Surface Manager (2D and 3D representation), WebKit/Chromium (web browser), SGL (2D graphics engine), 3D libraries, FreeType (vector graphics), SQLite (database) and SSL (secure connection).
- Application Framework. It provides an open development platform for apps. The most important services are: Views (visual component), Resource Manager (provides access to non-coded resources), Activity Manager (allows apps show icons in State Bar) and Content Providers (provides access to other apps data).
- Applications. This layer is formed by all the apps installed in the system, all of them have to be running in the Dalvik virtual machine.

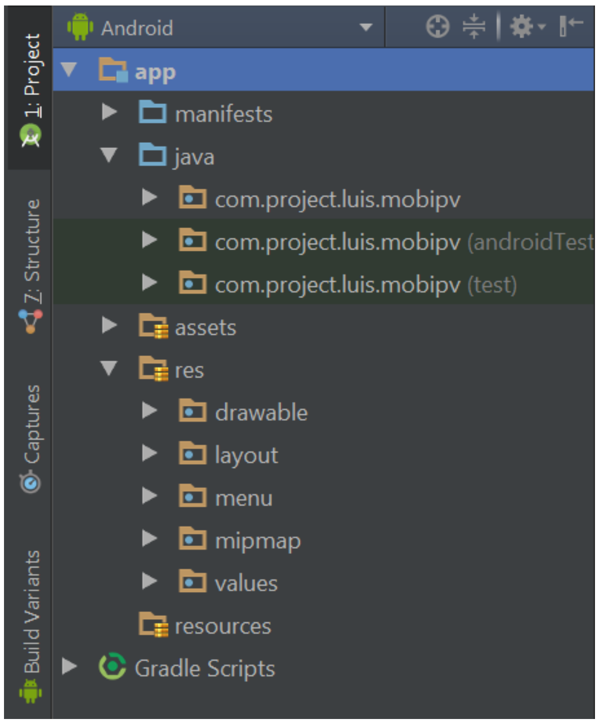
Android Studio [15] (Figure 21) is the official integrated development environment (IDE) for Android. It was presented in May 2013 in Google I/O conference, but it was not until December 2014 when the first stable version was released. Finally, Google had its own Android development platform that was the substitute of Eclipse. The main functions of Android Studio are:

- Gradle based compilation system.
- Fast emulator.
- Unified development for all platforms.
- Instant Run for applying changes while the app is executing without the need of compiling again,
- Code templates and GitHub integration for an easier compilation common app functions
- C++ and Native Development Kit (NDK) compatibility.
- Emulation of Android devices for the testing of the developed apps.

Figure 21: Android Studio. From [15]

Continuing with some fundamentals of Android programming there are two main things to be explained, on one hand the architecture of an Android project and on the other hand the component of an app as well as the main characteristics of Java, which is the programming language in which Android is based.

A project in Android is composed basically by the folders in Figure 22.



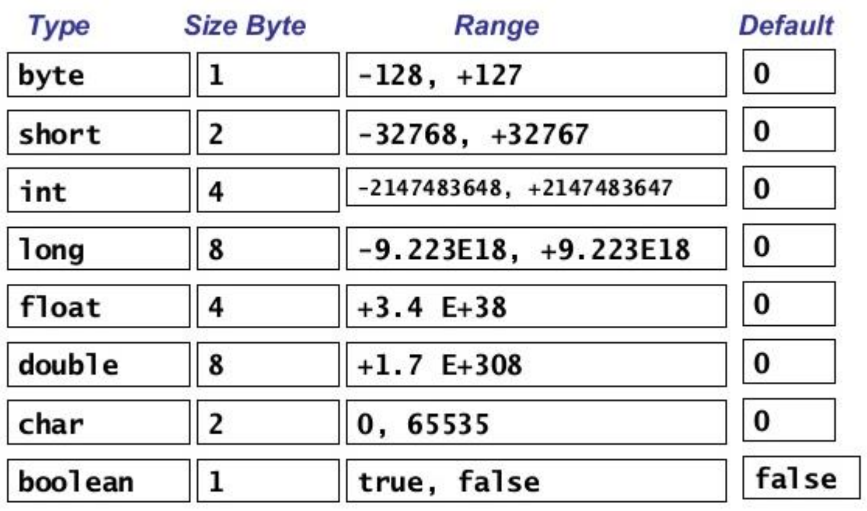
```
▼ app
    ▶ manifests
    ▼ java
        ▶ com.project.luis.mobipv
        ▶ com.project.luis.mobipv (androidTest
        ▶ com.project.luis.mobipv (test)
    ▶ assets
    ▼ res
        ▶ drawable
        ▶ layout
        ▶ menu
        ▶ mipmap
        ▶ values
      resources
▶ Gradle Scripts
```

Figure 22: Folder architecture of an Android project

The main components are:

- Manifests: This folder contains "AndroidManifest.xml" which is a file that describes the Android application. Within it, there are the main activities, the permissions of the app, the intents and the content providers.
- Java: This folder contains the source code of the app.
- Assets: This folder contains files or other folders that may be used for the app, like HTML or JavaScript files.
- Res: This folder contains the resources used by the app, organized in other folders as drawable where the images or image descriptors are located; layout where the XML files that describe the views of the app are; menu with the activities menus inside; mipmap where there are some special images as the main icon of the app and values with XML files with string, colour, style or dimensions values.
- Gradle: This is the build system that Android Studio uses to compile the applications.

The Android programming language is based in Java, so therefore it is an objects oriented language as well. An object is an entity that has some properties or attributes and behaviours or methods. A class defines the object and is able to store information of the object using attributes, initialize the object through the builder and change or check the state of an object with a method. The primitives are responsible of storing human data as numbers and alphabets, the primitives are shown in Figure 23 and more information as the size (in bytes), the max and min value and the default when they are initialized is also displayed.



Type	Size Byte	Range	Default
byte	1	-128, +127	0
short	2	-32768, +32767	0
int	4	-2147483648, +2147483647	0
long	8	-9.223E18, +9.223E18	0
float	4	+3.4 E+38	0
double	8	+1.7 E+308	0
char	2	0, 65535	0
boolean	1	true, false	false

Figure 23: Java primitives [17]

Moving into Android, there are a series of elements that are indispensable for developing apps.

- View: The views are the components that form the user interface, for example a button or a text input. All the views are objects descending from View class, therefore can be defined using Java code. Although, the standard procedure is to define them using a XML file.
- Layout: A layout is a group of views grouped in a certain way. The layouts are also objects descending from View class and as the same way as the views, they can be defined either using Java code or a XML file.
- Activity: An app in Android is going to be made by a combination of elements known as the screen of the app. Every one of this elements is an activity. Their main function is to create the user interface and all of them belong to a class descendent of Activity class.
- Thread: A thread is a Linux process that Android executes for the different apps. The standard approach is that all the components of an app are executed in the same thread but there can be also many threads running at the same time for the same application. For example, there can be one process for the interaction with the user and a secondary thread running in the background for managing the network capabilities and talking with a server or a client.
- Intent: The intent represents the intent of performance an action and it can be used for starting an activity.
- Event: In Android there are several ways of intercept events. For example, it is possible to one function listening to an event, or listener, that starts performing one task when the user presses on the screen, presses the back button, etc. On the other hand, the listener can be also waiting in a second thread for the arrival of a determined message from the network to start the programmed actions.

Summing up this part of the section, in Android, Java introduces the classes and objects with their methods and primitives. All of them together with the Android views, layouts, activities, threads, intents and events make the app which has a complex architecture in which all the folders and files make use of the rest to enable the interaction of the user with the login behind the app with a multimedia interface.

Leaving away Android, the next software is related with the possibility of using Ubuntu in Windows, since the laptop that is being used for the app development is running a Windows OS. The software is called Oracle VM VirtualBox [18] (Figure 24). VirtualBox allows the computer to run more than just one OS at the same time. To give a brief overview about the main features:

- Portability. For example, VirtualBox allows to create the image of an OS running Windows and then run the same image in a different OS. This is because there is a standard format for the images which is called Open Virtualization Format (OVF).
- It does not require the new processor features built in new technology and can be run in old hardware which does not contain these characteristics.

- Guest Additions. This capability allows to share folders between the host system and the emulated one.
- Support of hardware as a USB controller, multiscreen resolution, network controller, etc.
- Saving of snapshots of a certain state of the system so it allows the possibility of going to a past state or recovering from an error
- Groups for managing the users that can have access to certain emulated OS
- Modular design so VirtualBox can be controlled from several sources at the same time, for example it can be controlled by the interface or by a command window simultaneously.
- Remote control by implementing the Remote Desktop Protocol (RDP).

Figure 24: VirtualBox. From [18]

3.2 Stages

The project has been divided in four stages, each one of them keeps on improving the Android User Interface and brings it closer to the final solution. The aim of this section is to explain each one of the different stages.

During the whole implementation process, a web browser window in the laptop (acting as assistive display) is used to check that all the activity of the app is interacting properly with the system. To access to the web service of mobiPV++, both the computer and the system have to be connected to the same network and in the address bar of the web browser, the IP of the hardware (or virtual OS) where mobiPV++ system is running has to be written plus the port 8080.

3.2.1 Laptop as main component

This stage starts after the study of the documentation and papers related with the project. In this stage the laptop described in the previous section, running Windows 7, is the only hardware that is used and inside it there are two programs running: Android Studio, which creates an emulated Nexus 5, and VirtualBox running an Ubuntu OS in which the mobiPV software is installed. See Figure 25 for the full diagram.

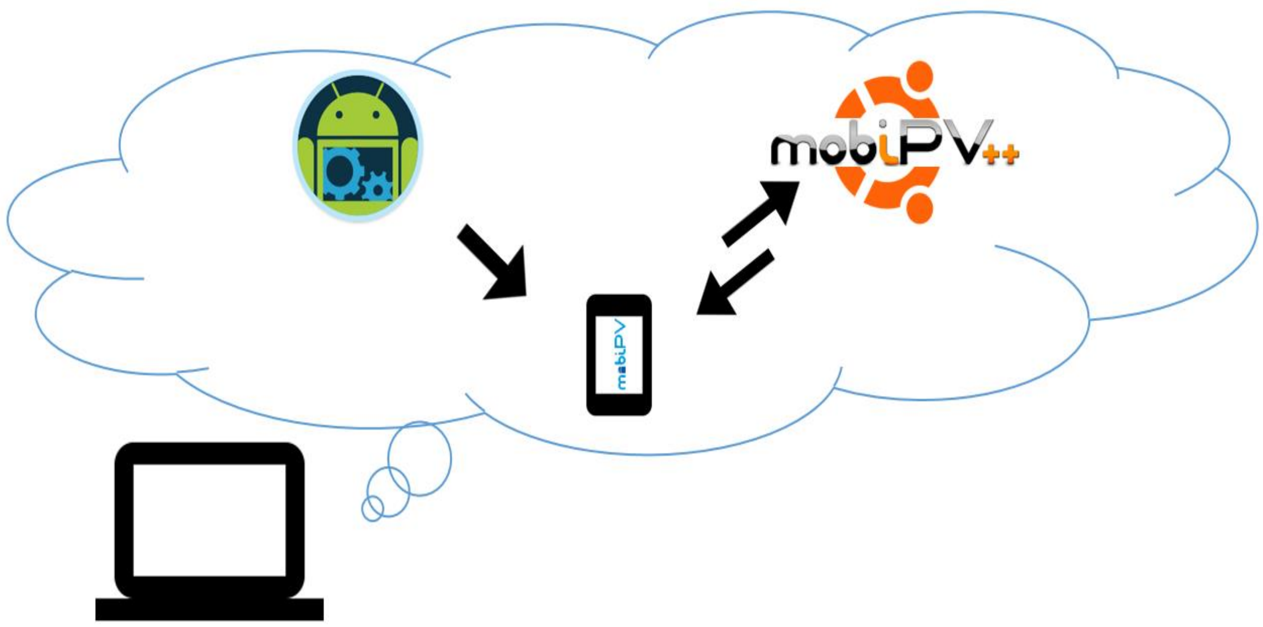
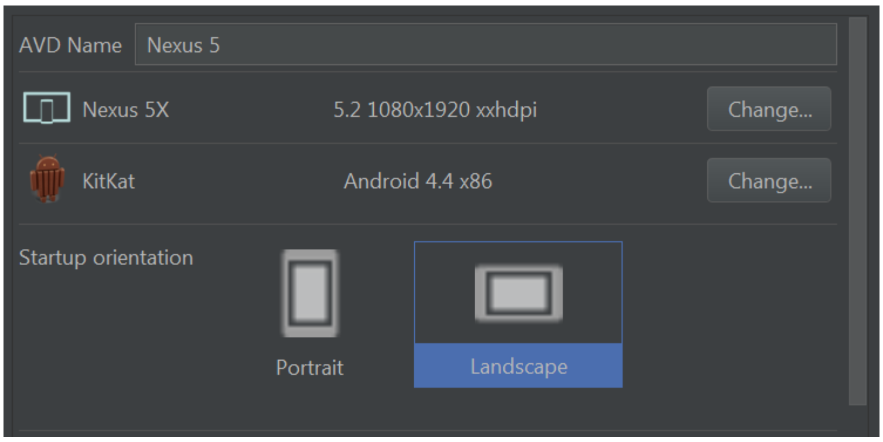


Figure 25: Diagram of the first stage configuration. With the Windows 7 laptop running Android Studio (top left icon) that generates an AVD running the Android GUI (bottom icon), and Ubuntu with mobiPV installed (top right icons)

On one hand Android Studio generates the emulation of the Nexus 5 using the Android Development Toolkit (ADT). The Android Virtual Device (AVD) allows to test mobiPV GUI while developing. The chosen configuration for the Nexus 5 can be seen in Figure 26.



AVD Name	Nexus 5		
	Nexus 5X	5.2 1080x1920 xxhdpi	Change...
	KitKat	Android 4.4 x86	Change...
Startup orientation	Portrait	[✓ Landscape]	

Figure 26: Settings Nexus 5

The chosen name is “Nexus 5”, with a resolution of 1080x1920 pixels and using the Android version 4.4 (nicknamed KitKat) that correspond with the commercial name of KitKat.

On the other hand, for connecting the AVD with Ubuntu virtual machine one virtual network is stablished using the tools offered by VirtualBox, more info about this process can be found in [18].

One static address is given to the Ubuntu OS, following the method in [19]. The IP address is 192.168.56.10 and the main reason of having a static IP is because the Android GUI is thought to be always sending messages to the core running in Linux, by getting the answer of this messages will make one decision or another.

Using the mentioned IP and mobiPV control port 7777, it is possible to stablish a socket control connection between the Android GUI and the core. Using this socket, it is possible to write control messages to mobiPV++ running in the Linux Kernel (in this stage running in the virtual machine) and receive messages from it.

This process is used by all the screens in the app and the structure shown in Figure 27 is followed too.

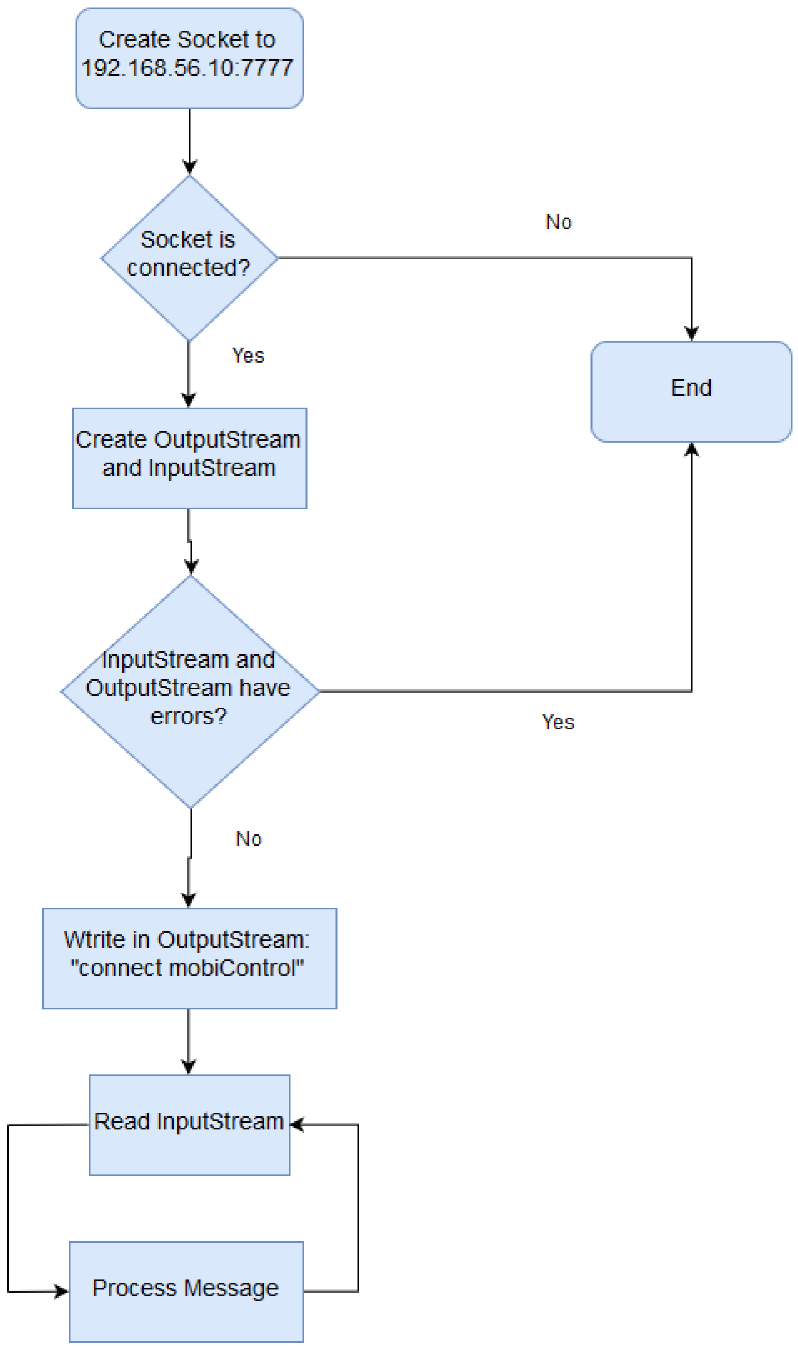
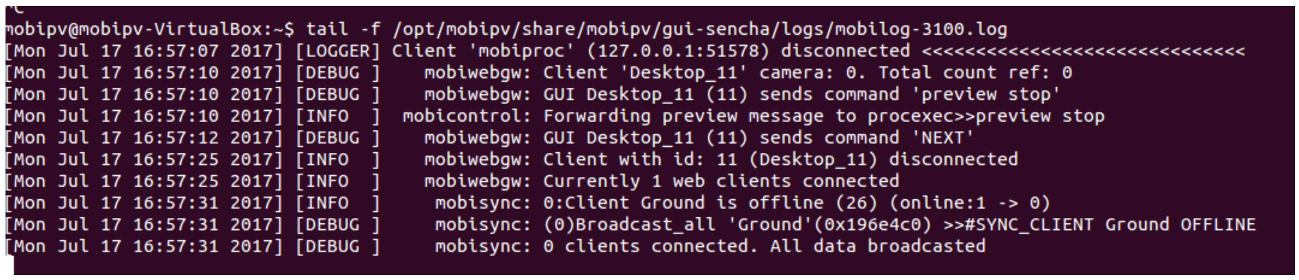

Create Socket to 192.168.56.10:7777
Socket is connected?
No
Yes
End
Create OutputStream and InputStream
InputStream and OutputStream have errors?
Yes
No
Wtrite in OutputStream: "connect mobiControl"
Read InputStream
Process Message

Figure 27: Flowchart of the connection made in the User Interface to send and receive messages from the core. The round boxes represent the beginning and ending of the Activity; the normal boxes represent the processes happening within the activity; the diamonds represent a conditional. The arrows represent the transitions. The beginning of the flowchar is the box on the top

As Figure 27 shows in first place a socket is created, then the OutputStream and the InputStream. Once the connection and both streams are checked the message "connect mobiControl" is sent to the core and the connection is stablished. The last part is the creation of a loop that will be reading all the messages that come from the core, process them and execute an action. In 3.3 Screens of mobiPV Android User Interface, this implementation will be explained more in depth with the main actions in each screen of the app.

In case either the socket or the streams contain some error, caused to for example by a broken link, one error window will pop up and the app will finish.

Inside the Virtual Machine where Ubuntu and hence mobiPV is running too, there is the possibility of accessing to mobiPV log file, where the system writes all the events that happen in the system. See Figure 28 for a visual example of the command in Ubuntu and the result.



```
^C
nobipv@mobipv-VirtualBox:~$ tail -f /opt/mobipv/share/mobipv/gui-sencha/logs/mobilog-3100.log
[Mon Jul 17 16:57:07 2017] [LOGGER] Client 'mobiproc' (127.0.0.1:51578) disconnected <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
[Mon Jul 17 16:57:10 2017] [DEBUG ]    mobiwebgw: Client 'Desktop_11' camera: 0. Total count ref: 0
[Mon Jul 17 16:57:10 2017] [DEBUG ]    mobiwebgw: GUI Desktop_11 (11) sends command 'preview stop'
[Mon Jul 17 16:57:10 2017] [INFO  ]  mobicontrol: Forwarding preview message to procexec>>preview stop
[Mon Jul 17 16:57:12 2017] [DEBUG ]    mobiwebgw: GUI Desktop_11 (11) sends command 'NEXT'
[Mon Jul 17 16:57:25 2017] [INFO  ]    mobiwebgw: Client with id: 11 (Desktop_11) disconnected
[Mon Jul 17 16:57:25 2017] [INFO  ]    mobiwebgw: Currently 1 web clients connected
[Mon Jul 17 16:57:31 2017] [INFO  ]     mobisync: 0:Client Ground is offline (26) (online:1 -> 0)
[Mon Jul 17 16:57:31 2017] [DEBUG ]     mobisync: (0)Broadcast_all 'Ground'(0x196e4c0) >>#SYNC_CLIENT Ground OFFLINE
[Mon Jul 17 16:57:31 2017] [DEBUG ]     mobisync: 0 clients connected. All data broadcasted
```

Figure 28: Use of command "tail -f /opt/mobipv/share/mobipv/gui-sencha/logs/mobilog-3100.log" for receiving information of the events happening in the system

This command is: *tail -f /opt/mobipv/share/mobipv/gui-sencha/logs/mobilog-3100.log*. Where "tail -f" gives back the last lines of one file and the rest is the path to the file where the system writes the events.

As mentioned before, mobiPV, running in the Ubuntu Virtual Machine, generates a web app, from where it is possible to interact with the system. See Figure 29 for a graphical example of how the web app looks like when accessing from a web browser running in Windows.

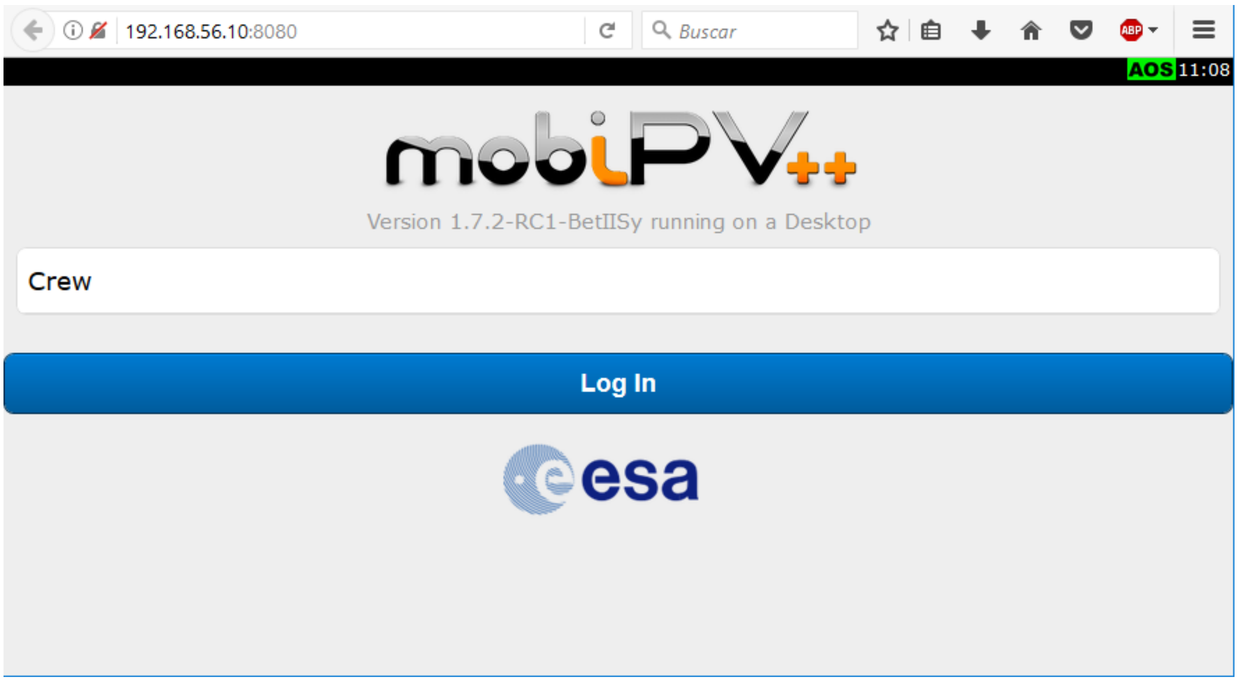


Figure 29: mobiPV web app accessed from a web browser running in Windows

In this case of Figure 29, the address in the navigation bar is 192.168.56.10:8080, where 192.168.56.10 is the static IP, of the Ubuntu OS, and the port 8080 is the address to mobiPV web app.

In this stage the views belonging to the Splash Screen, Login (see 3.3.2 Login Screen), part of Side Menu (see 3.3.3 Side Menu) and Main Menu (see 3.3.4 Main Menu Screen) were developed and tested.

3.2.2 Addition of Nexus 6 as display element

This stage is motivated with the possibility of testing the Android GUI in a real device. As a provisional step the installation of the Android GUI is made in a Nexus 6 considering that the target device of this project is a Nexus 5 due to the fact that this is the device which is currently on board of the ISS.

The fact that the Nexus 6 is running an Android 7.0 version makes also the installation of mobiPV++ within its Kernel impossible since this system is designed for an Android 4.4 version.

Considering these two factors, the final configuration for this stage is the one represented in Figure 30

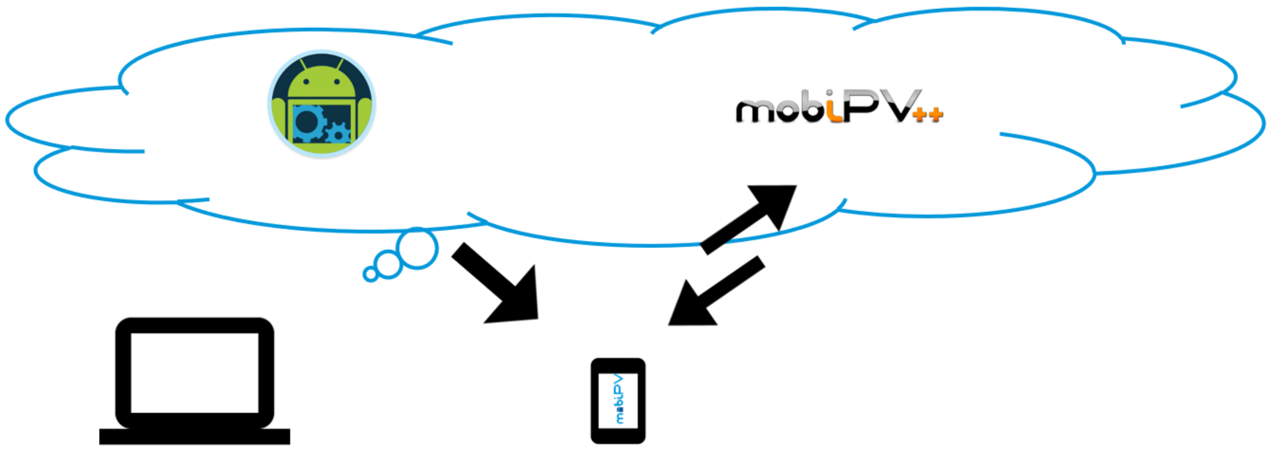


Figure 30: Configuration of the second stage. With Android studio (top left icon), mobiPV++ running in an Ubuntu virtual machine (top right icon) and a Nexus 6 running the Android GUI (bottom icon)

The configuration in this case is similar to the configuration in 3.2.1 Laptop as main component. The difference comes from the fact that the smartphone is real, so it is needed a real connection between Android Studio and the Nexus 6 on one hand and the connection between mobiPV++ and the Android GUI within the Nexus 6 on the other.

The first thing to do is to enable the USB Debugging in the Smartphone. For this configuration, the manual in [20] has been used.

Once this has been done, Android Studio will detect the smartphone as a Debugging Device and the app can be run in it as it was done before in the AVD.

Secondly, a network connection between the laptop and the smartphone is created, for this action, the steps in [21] are followed.

After checking, by using the command *ping*, that both the smartphone and the Ubuntu OS can see themselves the will work perfectly as the IP address of mobiPV++ remains the same (192.168.56.10) and the Android User Interface tries to reach this IP address using the port 7777.

The new screen implemented in this stage is the Procedure Viewer (see 3.3.5 Procedure Viewer Screen) and many of its features as process of the HTML file coming from the core; the addition of the green marker; the text and audio note taking; the automatic scrolling and the possibility of disabling and enabling it; the "jump to step" function; and the "next" and "previous" buttons to move within the procedure.

In relation with how the procedure is downloaded in the Procedure Viewer Screen, there are some aspects to be taken into account (see Figure 31).

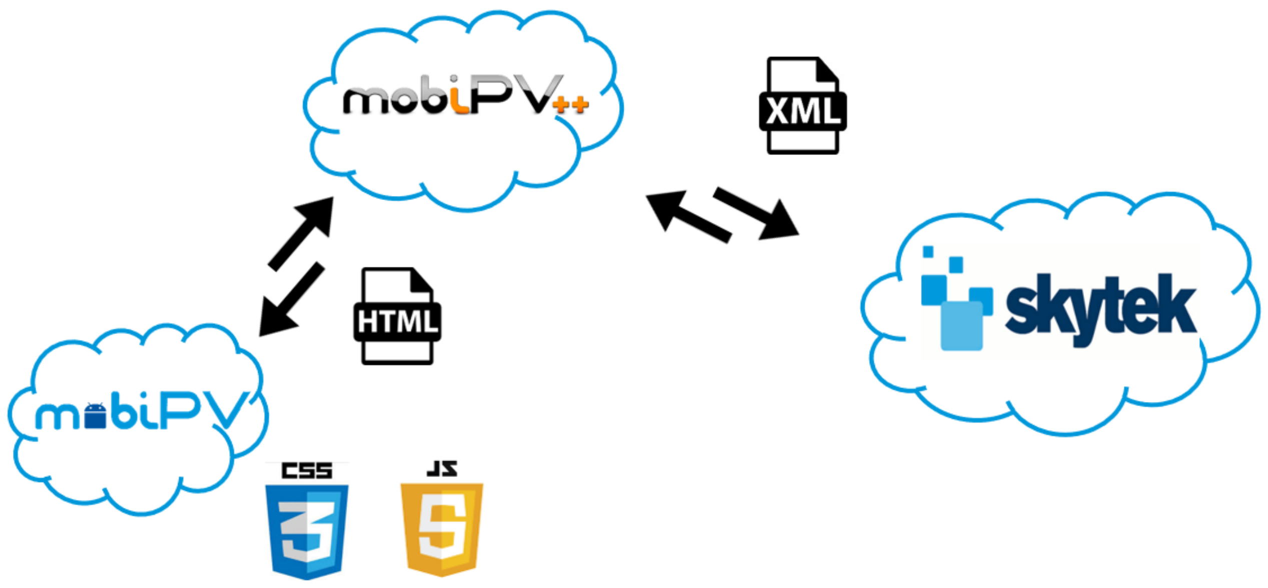


Figure 31: Procedure downloading process. From right to left, the Skytek cloud represents the server where all the procedures are stored in xml format. The cloud where mobiPV++ system represents the system running in the Linux core runs that converts the XML file into a HTML file, the last cloud represents the mobiPV Android User Interface that receives the HTML file of the procedure and adds the CSS and Javascript headers to it

In Figure 31 the whole process of acquiring the procedure is explained graphically. When mobiPV++ is executed for the first time, either using the Android User Interface or using the web browser, the user has to go to Setting Screen located in the Side Bar (See 3.3.3 Side Menu). From the Setting Screen the procedure list is downloaded from Skytek, stored in the software folders of the device and converted into a HTML file. Then, the Android GUI receives through a TCP socket message the path and name of the HTML file, reads and processes it. See more detailed information in 3.3.5 Procedure Viewer Screen.

3.2.3 Addition of Nexus 5 as main component

Once the Procedure Viewer Screen is prepared and working, it is time to move into a more realistic environment. As mentioned before the Nexus 5 is the device on the ISS and runs an Android 4.4 which is the only Android OS compatible with mobiPV.

Before trying to install mobiPV, the Nexus 5 is prepared, the first thing to do is checking the Android OS version, if the version is different from the 4.4, the device has to be upgraded or downgraded as explained in [22]. The user has to be also a root user and for that the steps in are followed.

When the Nexus 5 has the correct Android OS version and the user is a root user mobiPV can be installed. The installation is made using Ubuntu OS and following the instructions in [23].

Figure 32 shows the resulting system architecture in this stage.

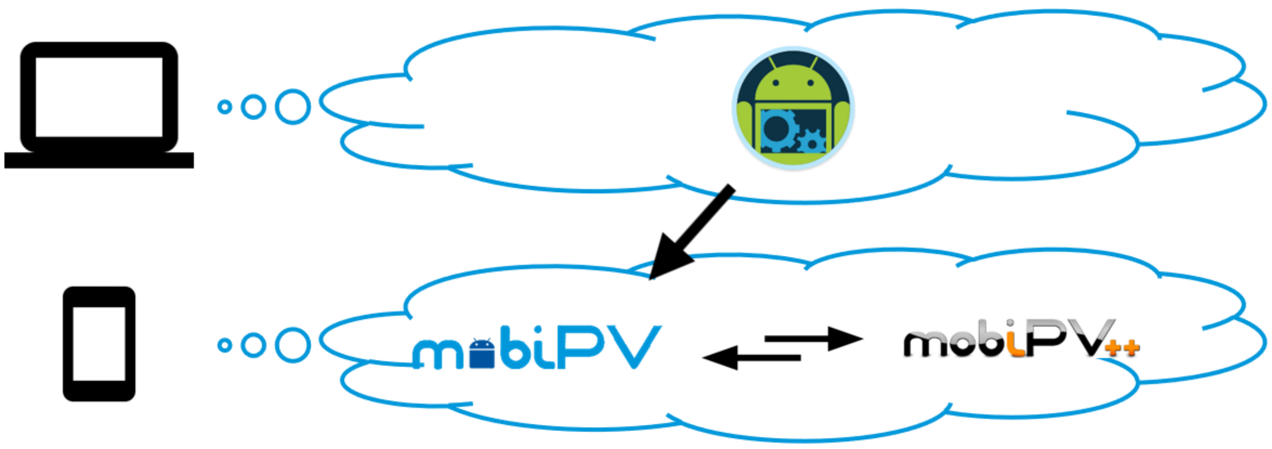


Figure 32: Architecture of the third stage of the project. On the top, there is the laptop, which runs Android Studio. On the bottom, there is the Nexus 5, which runs mobiPV and the Android GUI

There is no need of a network connection between the laptop and the Nexus 5 in this step, the connection is only made by an USB cable and is used just in case an update in the Android GUI is done. However, the connection can be useful if the user wants to monitor the behaviour of the app by having in another screen the web app created in parallel by the system. This is done by writing in the web browser the IP address of the Nexus 5 and the port 8080. The result is the same that the one showed in Figure 29.

Using the USB cable the mobiPV log can be accessed using the similar command as in 3.2.1 Laptop as main component, but with the difference that in this case, the ADB, which is an android tool, is needed for accessing to the smartphone. Once the smartphone is being managed by the Ubuntu console, the process is the same.

The communication between mobiPV and the Android GUI within the smartphone is similar as before, the difference now is that instead of using an external IP address, the Android GUI for the first connection is using “localhost:8080”, as expected the port is the same, for it is the path to the web app. The IP is now the localhost, which is the loopback address. Writing localhost corresponds to writing 127.0.0.1.

In this stage the Procedure Viewer screen is completed. The remaining features to add were the possibility of adding video and image notes to the steps of the procedures which are now available.

3.3.2 Use of Nexus 5 and laptop in a Peer to Peer configuration

This environment allows a real test (see 4 Testing and execution of mobiPV app and Appendix 1) over the app, simulating on one side a mobiPV software as the one that is located on the ISS and on the other side a mobiPV system in Linux, simulating the computer in the ground centre.

This stage allows the testing of the chat feature and the collaboration mode, which allows to put two or more mobiPV cores working together. The architecture of this stage is shown in Figure 33.

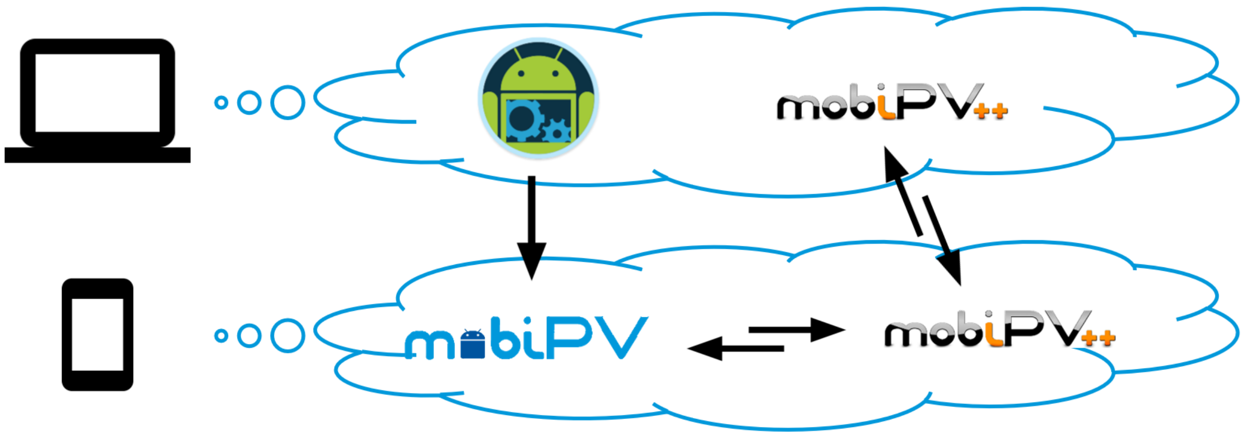


Figure 33: Architecture of the fourth stage of the project. On the top, there is the laptop, which runs Android Studio and mobiPV in a virtual machine. On the bottom, there is the Nexus 5, which runs mobiPV and the Android GUI

The figure shows the laptop and the Nexus 5. Within the laptop Android Studio is used to update the Android version of mobiPV via USB and mobiPV in Linux is connected to the mobiPV core of the smartphone via a synchronization server, that the system implements and was described in section 2.3 Architecture. Of course laptop had to be connected to the same WLAN and both of them have to be able to reach the other one.

The last connection is between the app and the mobiPV core within the Nexus 5. Same as before it is made through a TCP socket opened by the app to the localhost and port 7777, here is where mobiPV component mobicontrol publishes the control messages to be used by mobiWebGW component to interact with the Web browser through a WebSocket (see Figure 14). In this case the Android app just access to the control component directly through a TCP socket, without the need of communicating with mobiWebGW.

3.3 Screens of mobiPV Android User Interface

The purpose of this section is to show the final results of the Android GUI. During the next points both the graphical solution and the flowchart of each main class are shown.

This section is divided in each of the different screens that form the app. These screens correspond to one of the following states in Figure 34

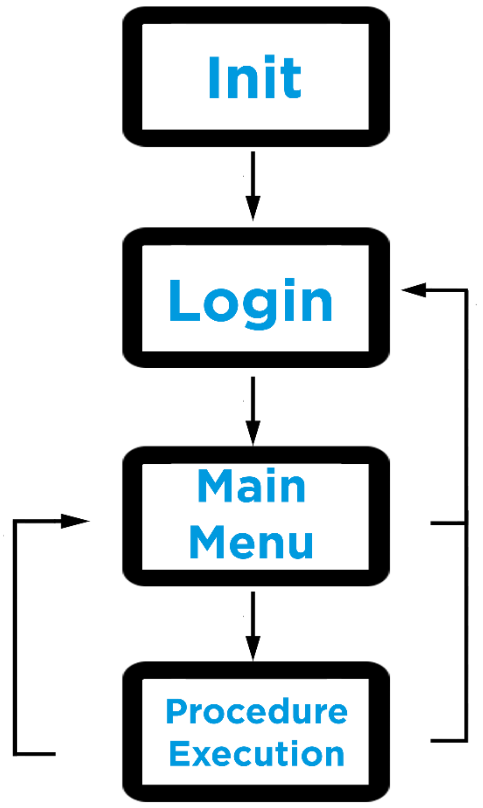


Figure 34: Different states of the mobiPV Android GUI

This diagram will be followed during the next subsections and each state and transition will be explained.

3.3.1 Splash Screen

The Splash Screen corresponds to the first screen when the app is opened. The screen belongs to the state Init from Figure 34. This state is the only one that automatically moves into the next state without the interaction of the user and starts mobiPV within the Nexus 5 core.

Figure 35 shows the flowchart of this screen.

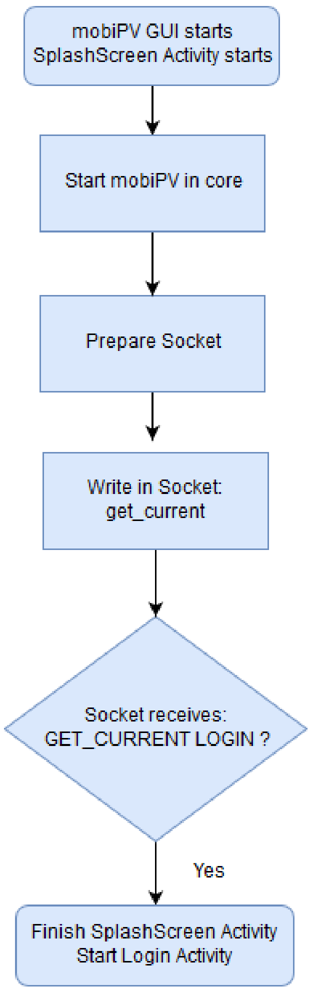


Figure 35: Flowchart of SplashScreen.

The beginning of the chart in Figure 35 corresponds with the round box on the top. Here is where the user opens the app from the desktop or the main menu of Nexus 5 Android OS. Then the first activity that is created is SplashScreen.

This activity starts mobiPV by executing two commands in the Linux core, these two commands launch the files start-mobipv1.sh and start-mobipv2.sh, which in turn execute all the mobiPV components explained in 2.3 Architecture.

What happens in "prepare Socket process" is similar to what Figure 27 shows. The only difference is that the last loop of the Figure never occurs. Instead of starting a loop, that processes all the received messages, one message containing "get_current" is sent. This command asks the system in which state is working at that moment.

If the answer is "+GET_CURRENT LOGIN", then the activity finishes and moves to Login Activity, because after the confirmation of the received command (+GET_CURRENT) comes the name of the state (LOGIN). The main reasons of adding a conditional is that maybe the answer is different after an update or the system is working in a different state.

Figure 36 shows how the SplashScreen looks like, with mobiPV logo and the ESA logo.

Figure 36: Starting of mobiPV Android GUI. This screenshot corresponds to the SplashScreen Activity in the very beginning.

Next state after Init is Login, which is automatically executed.

3.3.2 Login Screen

The login Screen belongs to the Login state in Figure 34. This is the first activity in which the user can interact with the mobiPV Android GUI. The user has to choose one user and if it is needed write one password, usually the users do not need any password and the text "aaa" is sent in the field of password, which is the standard value. See Login Screen in Figure 37.

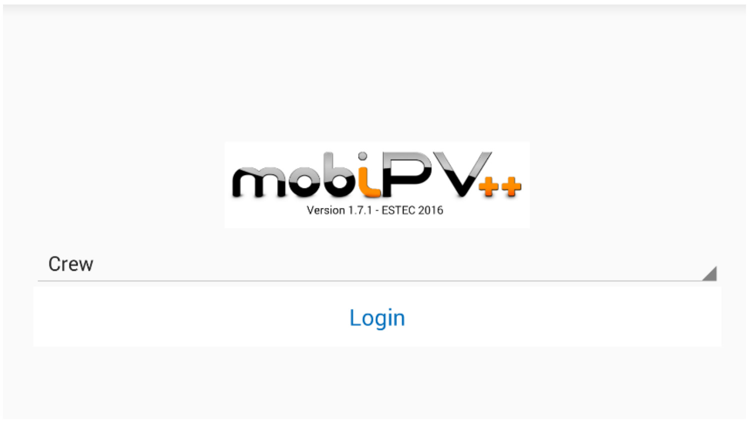


Figure 37: Login Screen

Figure 38 shows the flowchart of this activity.

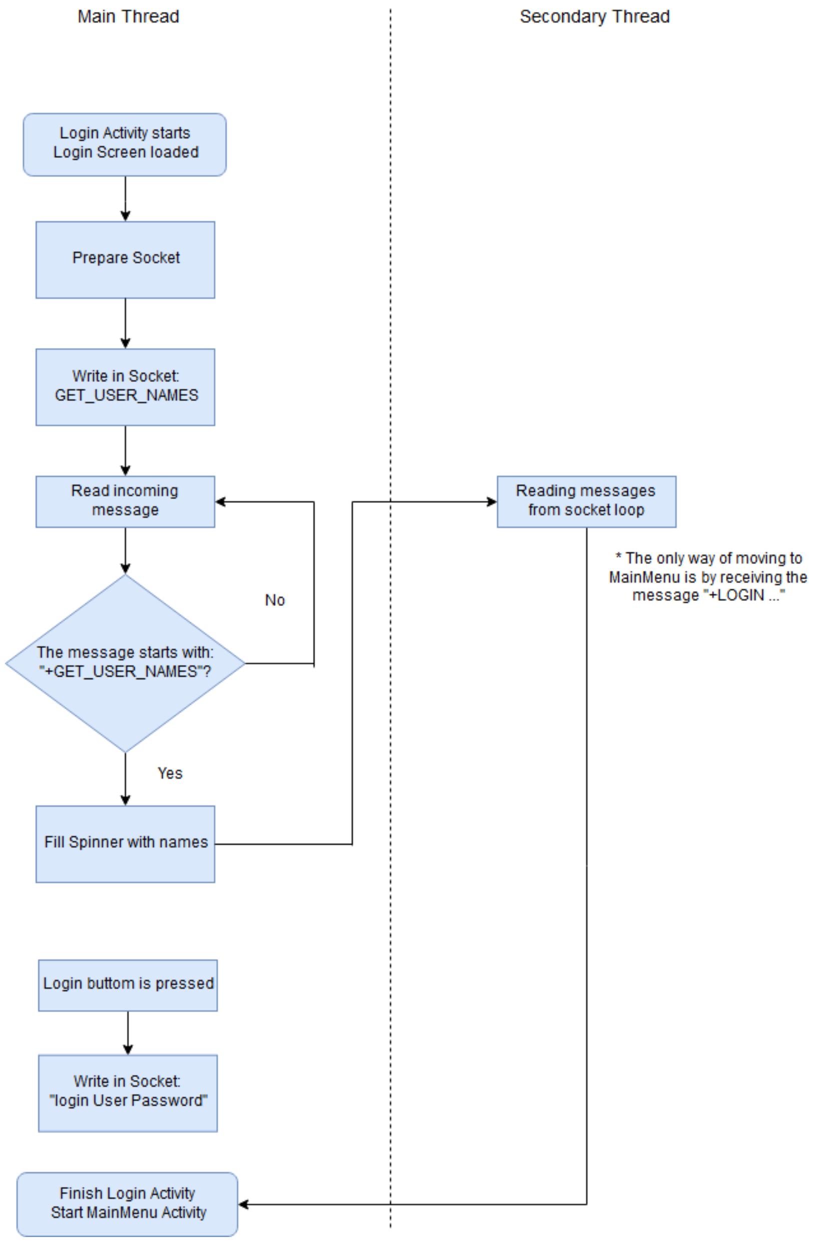


Figure 38: General flowchart of Login. There are two threads separated by the discontinuous line. The beginning of the flowchar is the box on the top left

When the mobiPV Android GUI changes from the SplashScreen to the Login Screen, the second activity starts, the first step is to prepare the socket from where the app obtains the list of users and writes the user name and the password for login. The preparation of the socket is similar to the process shown in Figure 27, with one difference. The difference is that instead of starting the loop for reading messages just after writing the message “connect mobicontrol”, the loop starts a little bit later, after some more preparations. Anyway the logic of the process is the same.

Once the socket is prepared, the system waits until the received message contains the list of registered users in the system since when mobiPV software, in the core of the Nexus 5, enters into the Login state (Figure 34) this message is automatically sent. If the received message does not start with "+GET_USER_NAMES" the app keeps on waiting until is received.

Then the message is saved within a String variable and split in the different users, each resulting word will be used to fill the spinner o the screen.

After filling the Spinner, the reading messages process coming from the loop starts in the secondary Thread. When the answer coming from the core tells the app that the login is correct the state moves to the next one and the screen also changes. For sending the user and the password to the core, the app waits until the user press login button.

In Figure 39 there is more information given about how the app does the login, because there are some other aspects to take into account. These aspects are related with a special user within the list of users, who has the opportunity of modifying a wider range of settings (see 3.3.3 Side Menu) than the other. This special user is called "sysop" and is also the only one who requires a password to login.

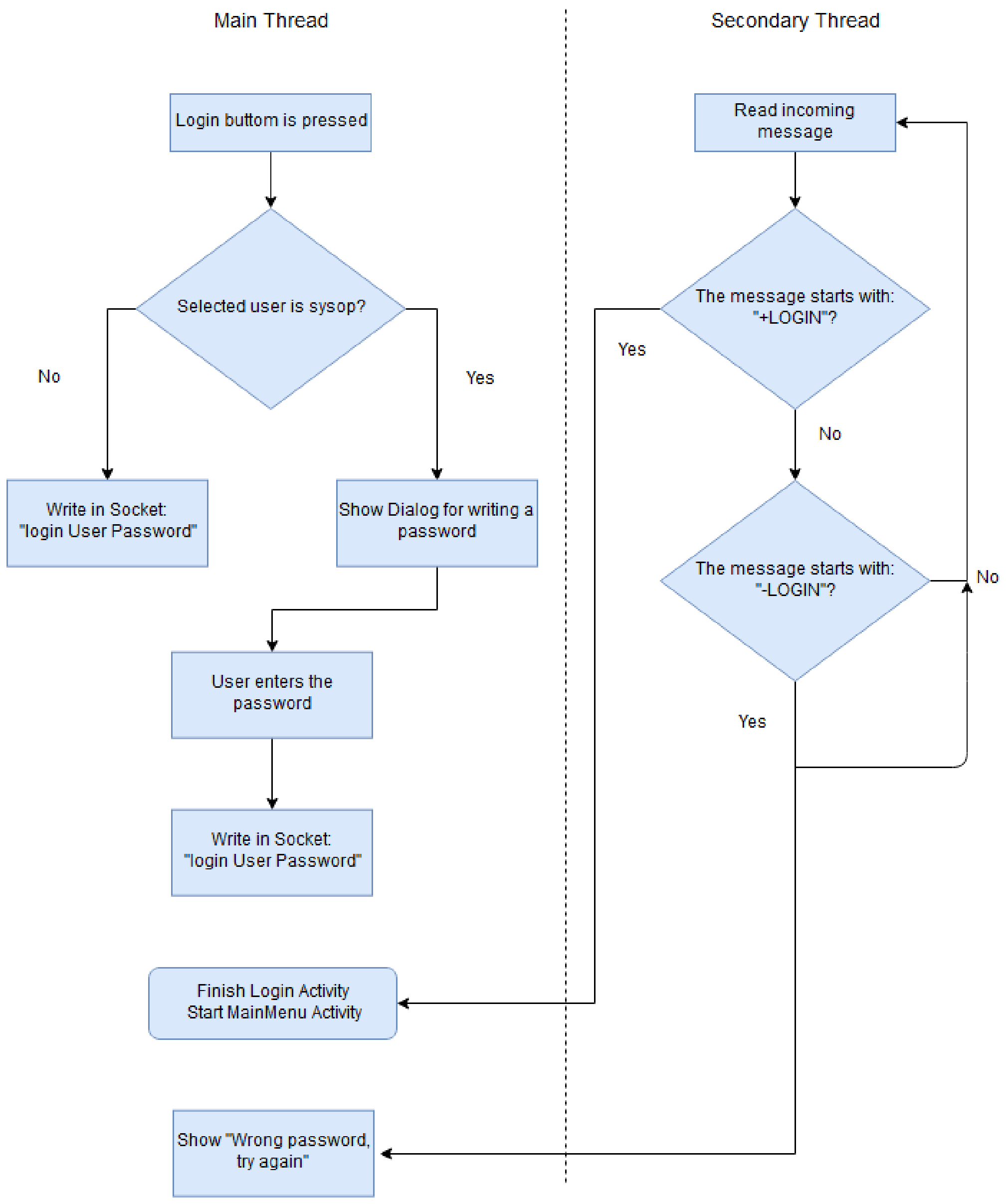

Main Thread
Secondary Thread
Login buttom is pressed
Selected user is sysop?
No
Yes
Write in Socket: "login User Password"
Show Dialog for writing a password
User enters the password
Write in Socket: "login User Password"
Finish Login Activity Start MainMenu Activity
Show "Wrong password, try again"
Read incoming message
The message starts with: "+LOGIN"?
Yes
No
The message starts with: "-LOGIN"?
No
Yes

Figure 39: Detailed flowchart of Login.

In Figure 39 the starting point is when the user enters the username for login into the app. The first verification that is done occurs in the app. The app checks if the username is the special one, “sysop”, if not, then “login [Username] aaa” is written in the socket and sent (“aaa” is the default password for all the users). If, on the contrary the user is the special one, a Dialog pops up asking for a password.

Coming back to Figure 39, when the user clicks in “CONFIRM”, this message is sent to the core: “login sysop [entered password]”.

That was the interaction of the user with the app, the rest of the processes happen in the secondary thread. For completing the login, once the message is sent the app waits until a response. There are two possible responses. On one hand if the user and password is correct (this will happen every time a login is made with the standard users and when user “sysop” password is correct), the response will start with “+LOGIN”, then the app ends the Activity and moves into the Main Menu Activity.

On the other hand, if the required password of user “sysop” is wrong, the answer from the core starts with “-LOGIN” and a message will pop up and the app will come back to a waiting state until the users presses back Login button, then the whole process starts again.

Next state is Main Menu and Figure 40 shows the next screen.

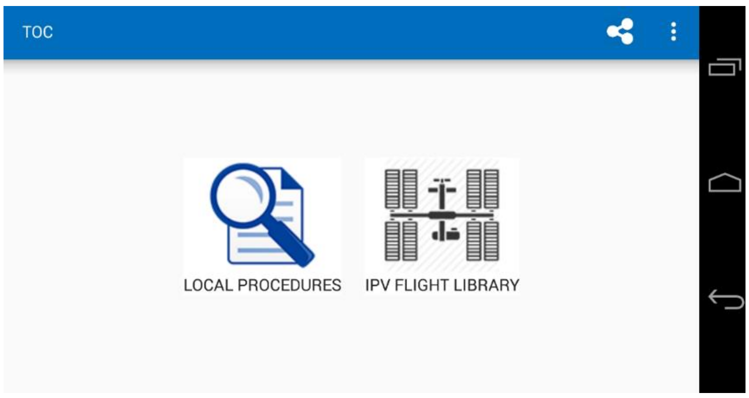


Figure 40: Main Menu screen

3.3.3 Side Menu

For a better understanding to the reader it is necessary to explain the Side Menu before the Main Menu because one of the most important aspects of the Main Menu is directly related with one of the features that are accessed from the Side Menu. On the top right corner in Figure 40, there is three vertical points. By clicking there, the side Menu is displayed, see Figure 41

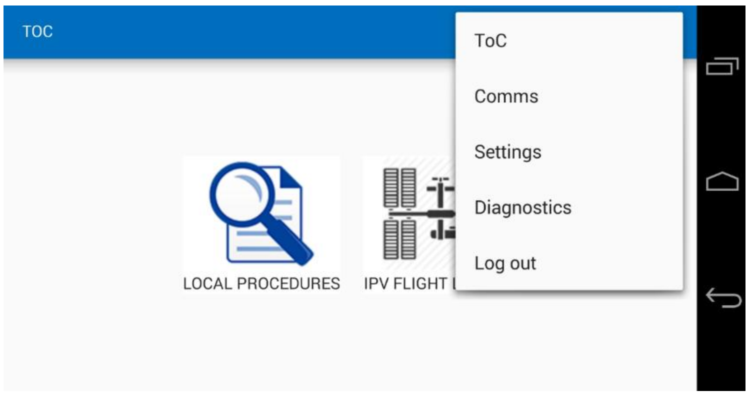


Figure 41: Side Menu Deployed

The different options are:

- TOC: By clicking on TOC, the user comes back to the Main Menu from any window or state.
- comms: One Dialog window will be opened. Through this window the user can interact in a cooperative session with other users using mobiPV by writing text messages.
- Settings: A new activity is launched and some settings of the software can be changed. These settings do not belong to the app but to the system. So the changes in the parameters affects to mobiPV running in the core and indirectly to the Android GUI
- Diagnostics: A new activity is launched from where the user can read messages coming from the core, as network or devices status. Using Diagnostics Activity, the user can identify errors occurring in the system.
- Logout: The system state and the screen come back to the Login state and screen.

When Settings or Diagnostics are pressed, it involves a change of Activity but the state in which the app keeps being the same. In the case of TOC, the app modifies its state if and only if the app is working in Procedure Viewer state (see 3.3.5 Procedure Viewer Screen). Before changing the app, the system checks if the button for moving to the next Activity belongs to the actual one, in this case a message pops ups with the following text: “You are already in the selected Menu”. On the contrary, the actual activity finishes and the app moves to the pressed one.

comms button generates a Dialog that allows the interaction between the user and the supporting team working at the same moment.

The last option is the Login button that involves a change of Activity and state, if this button is pressed from one activity, the message “logout” is sent to the core, and if it validates the message and answer back with a confirmation message then the activity and the state change. Then the Login screen is shown again.

A general flowchart showing the information given below, therefore the processes that are executed after pressing every button is shown in Figure 42.

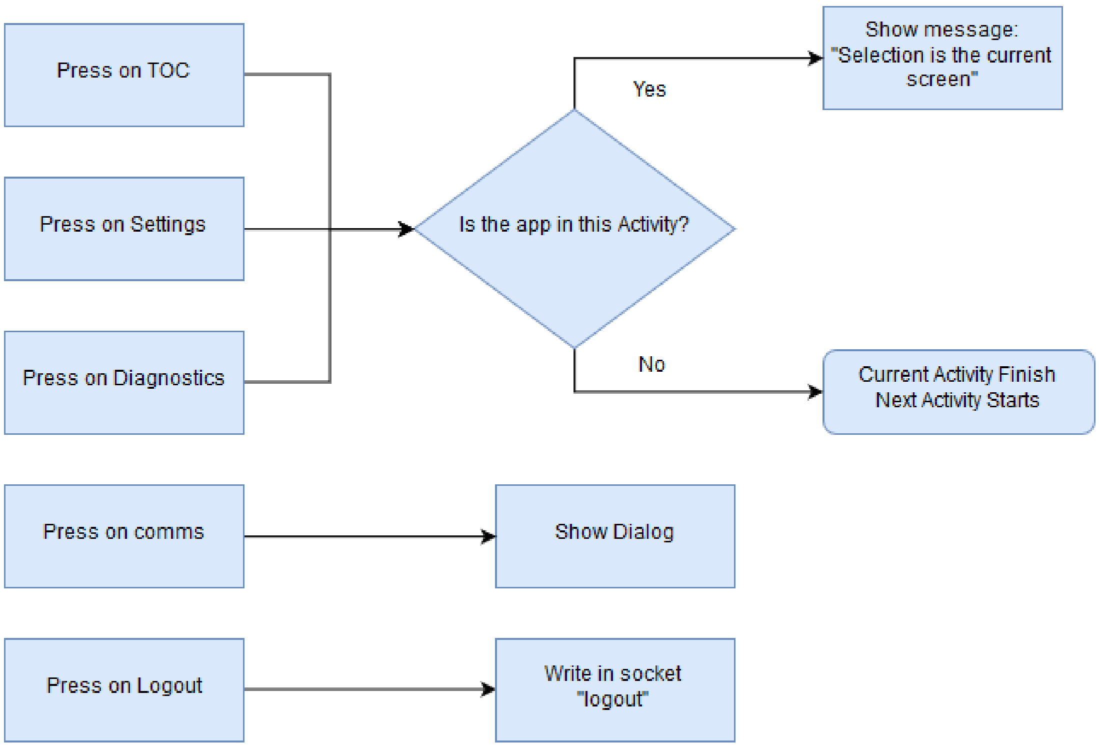


Figure 42: General flowchart of Side Menu.

Now it is time to see in depth every one of the five processes and all the logic that they implement.

TOC

Starting with the first element of the list, the user finds TOC, which comprise coming back to the main menu where everything starts. From every Activity, the app implements an Intent [24] that loads a different activity, but there is one exception. When the state is Procedure Viewer and therefore the screen is the screen explained in 3.3.5 Procedure Viewer Screen, going to Main Menu involves a change of the state. Figure 43 shows the flowchart of how the Android GUI manages this exception.

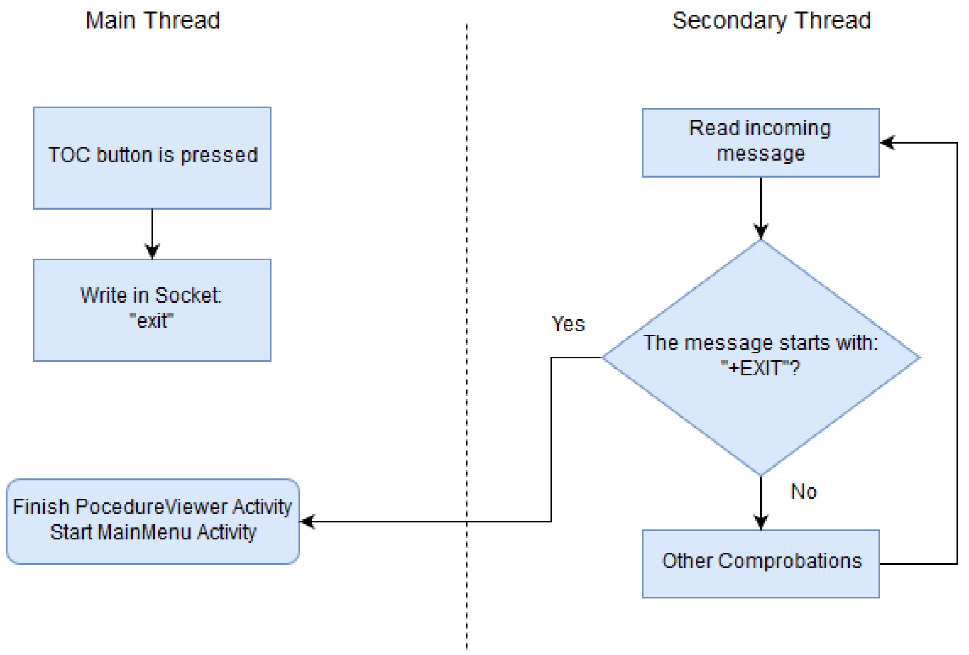


Figure 43: Detailed flowchart performance of TOC component of Side Menu.

The processes in the left part of Figure 43 involve an interaction with the user, the processes of the right part runs in a secondary thread under the control of the user. When the user presses on TOC, a message is generated and sent to the core which contains the word "exit", in this case mobiPV running in the core identifies this message and proceeds with a change of the state and answer at the same time with "`+EXIT`". When in the secondary thread, one message is found containing that word, the app also changes the screen to Main Menu. In the case that the message is not "+EXIT" the thread keeps of comparing the message with some other for further more actions that will be explained in 3.3.5 Procedure Viewer Screen.

Comms

The feature comms is accessed from the Side Menu and it implements a chat where all the users working in the same session can have a live communication. From whichever activity the button comms is pressed, one Dialog pops up that is divided in two parts. The top part consists of a white board where all the messages sent are added and the bottom part consists of an EditText box where the user can write the message and a button at its right to send the text to the core.

The logic behind this process is not complex and follows the line of Diagnostics and Settings activities. When the activity starts it opens a socket connection with the core and sends the message "`get_sms_msgs?`". The possible answers are "`+get_sms_msgs?`", that is the confirmation message and if it is received in the first place it means that there are no available messages to load;

and "`/get_sms_msgs?`" that will be followed by relevant information such as the sender, the included text, creation date or who are the receivers.

Every time that the command "`/get_sms_msgs?`" arrives with one message, this text is stored in an ArrayList and when the "`+get_sms_msgs?`" arrives which means that all the messages have been received, one method is called to reorganize the different parameters in the message and to plot them with the bubble background.

For sending a message, once the user has written the text in the box, the button has to be pressed and then the command "`COM_txt NOT ALL: + message`" is sent to the core, when the answer starting with "+`COM_txt`" arrives another message "`get_sms_msgs?`" is sent and the process to plot the new text is repeated as before. See Figure 44 for a detailed flow graph.

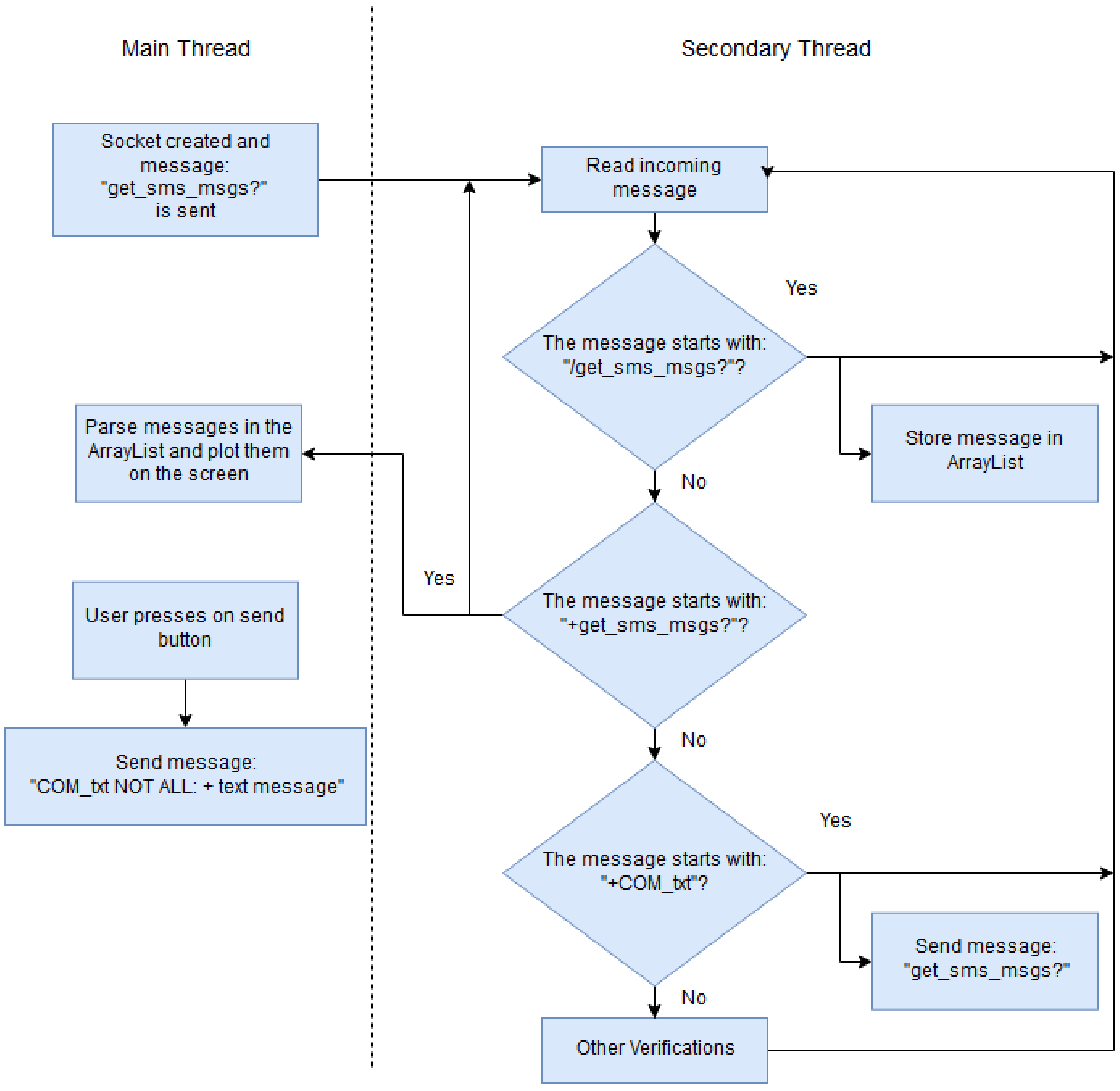

Main Thread
Secondary Thread
Socket created and message: "get_sms_msgs?" is sent
Read incoming message
Yes
The message starts with: "/get_sms_msgs?"?
Store message in ArrayList
No
Parse messages in the ArrayList and plot them on the screen
Yes
The message starts with: "+get_sms_msgs?"?
User presses on send button
No
Send message: "COM_txt NOT ALL: + text message"
Yes
The message starts with: "+COM_txt"?
Send message: "get_sms_msgs?"
No
Other Verifications

Figure 44: Detailed flowchart that describes comms activity.

Settings

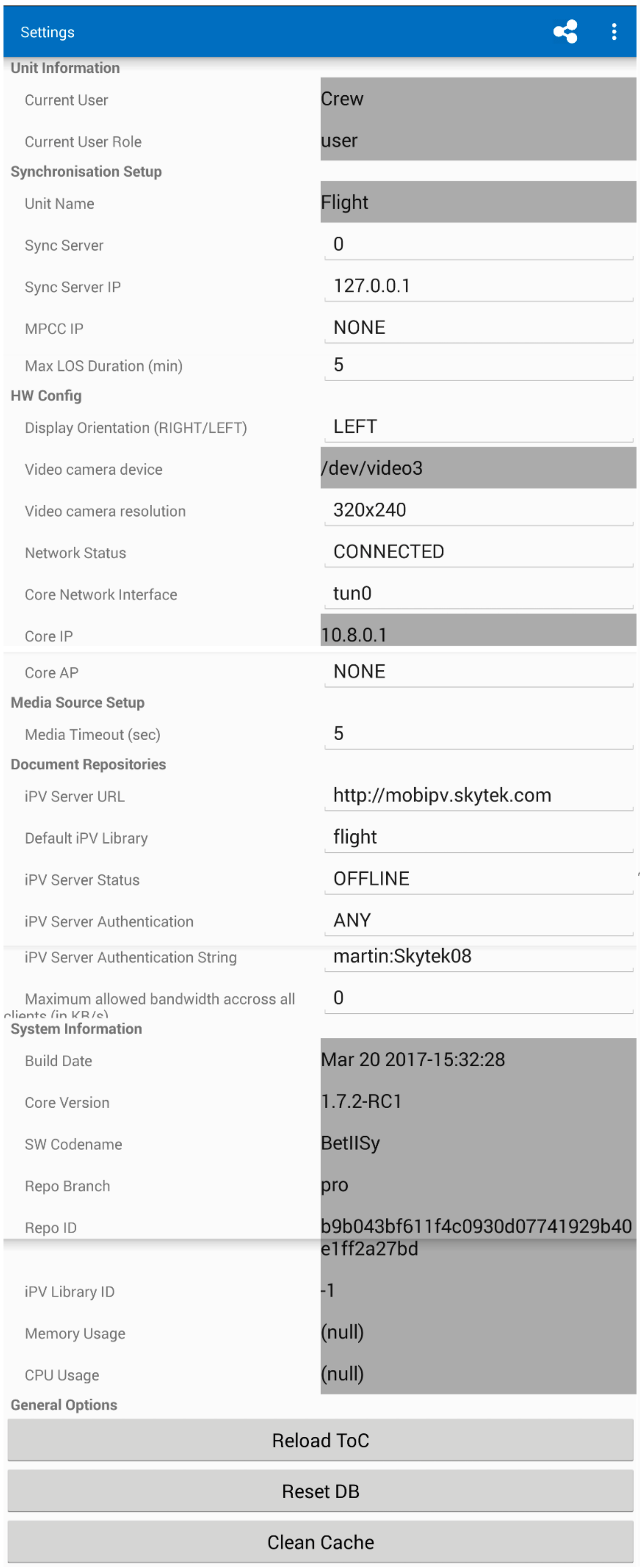
When Settings option is selected a new Activity starts and the screen changes. The new screen is shown in Figure 45. From this screen and by sending commands to the core some basics settings of mobiPV can be changed.

Trying to describe Figure 45, all the settings are divided in fields, every group of fields are predicted by a title: Unit Information, Synchronisation Setup, Hw Config, Media Source Setup, Document Repositories and System Information. When having a closer look to every field, there is also a special characteristic to be commented, not all the fields share the same colour, the fields in grey cannot be modified and are used to give information to the user, the white ones can be indeed modified following a process that will be explained later.

In relation with the fixed fields and the number of them, these two features are not the same for all the users, as mentioned before in this document, there is a special user in the system called “sysop”. When this user accesses to settings, different fields are unlocked to be modified and other new fields appear revealing new information.

The different components that can be found in the View are:

- TextView: Corresponds to the printed text of the titles of the groups of features and the values of every item
- EditText: Corresponds to the fillable fields on the right of the TextViews belonging to the items.
- Button: Corresponds to the last three boxes that implement different functions



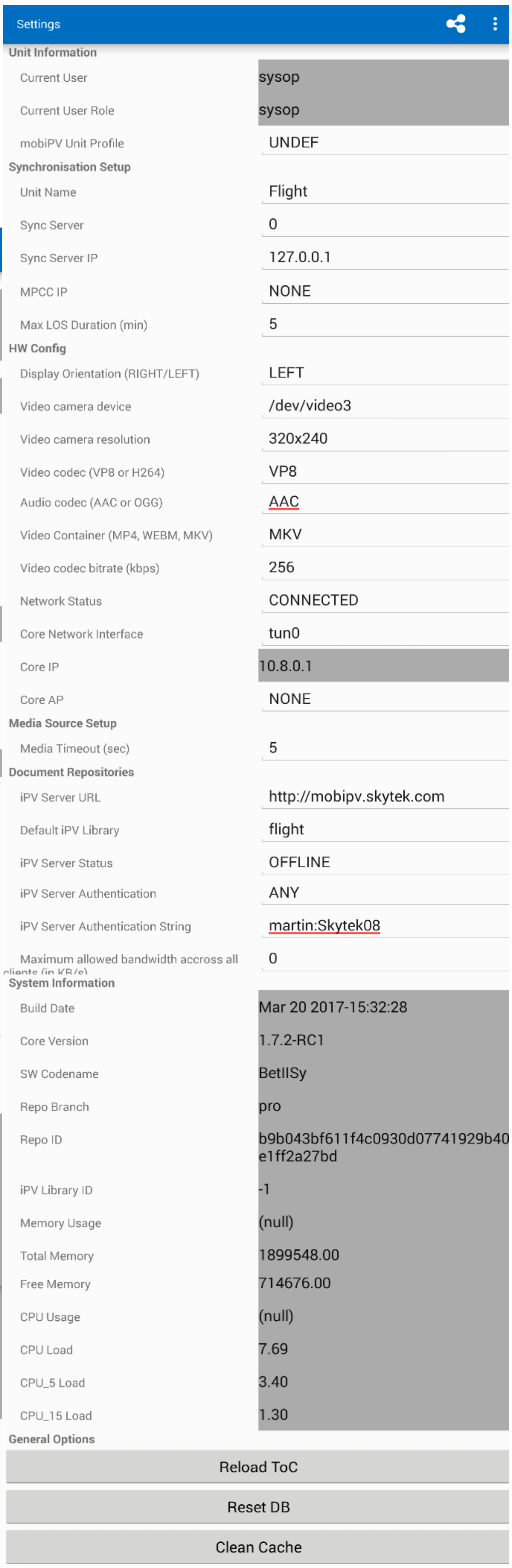
Settings

Unit Information	
Current User	Crew
Current User Role	user
Synchronisation Setup	
Unit Name	Flight
Sync Server	0
Sync Server IP	127.0.0.1
MPCC IP	NONE
Max LOS Duration (min)	5
HW Config	
Display Orientation (RIGHT/LEFT)	LEFT
Video camera device	/dev/video3
Video camera resolution	320x240
Network Status	CONNECTED
Core Network Interface	tun0
Core IP	10.8.0.1
Core AP	NONE
Media Source Setup	
Media Timeout (sec)	5
Document Repositories	
iPV Server URL	http://mobipv.skytek.com
Default iPV Library	flight
iPV Server Status	OFFLINE
iPV Server Authentication	ANY
iPV Server Authentication String	martin:Skytek08
Maximum allowed bandwidth accross all clients (in KB/s)	0
System Information	
Build Date	Mar 20 2017-15:32:28
Core Version	1.7.2-RC1
SW Codename	BetIISy
Repo Branch	pro
Repo ID	b9b043bf611f4c0930d07741929b40e1ff2a27bd
iPV Library ID	-1
Memory Usage	(null)
CPU Usage	(null)
General Options	

Reload ToC

Reset DB

Clean Cache



Settings

Unit Information	
Current User	sysop
Current User Role	sysop
mobiPV Unit Profile	UNDEF
Synchronisation Setup	
Unit Name	Flight
Sync Server	0
Sync Server IP	127.0.0.1
MPCC IP	NONE
Max LOS Duration (min)	5
HW Config	
Display Orientation (RIGHT/LEFT)	LEFT
Video camera device	/dev/video3
Video camera resolution	320x240
Video codec (VP8 or H264)	VP8
Audio codec (AAC or OGG)	AAC
Video Container (MP4, WEBM, MKV)	MKV
Video codec bitrate (kbps)	256
Network Status	CONNECTED
Core Network Interface	tun0
Core IP	10.8.0.1
Core AP	NONE
Media Source Setup	
Media Timeout (sec)	5
Document Repositories	
iPV Server URL	http://mobipv.skytek.com
Default iPV Library	flight
iPV Server Status	OFFLINE
iPV Server Authentication	ANY
iPV Server Authentication String	martin:Skytek08
Maximum allowed bandwidth accross all clients (in KB/s)	0
System Information	
Build Date	Mar 20 2017-15:32:28
Core Version	1.7.2-RC1
SW Codename	BetIISy
Repo Branch	pro
Repo ID	b9b043bf611f4c0930d07741929b40e1ff2a27bd
iPV Library ID	-1
Memory Usage	(null)
Total Memory	1899548.00
Free Memory	714676.00
CPU Usage	(null)
CPU Load	7.69
CPU_5 Load	3.40
CPU_15 Load	1.30
General Options	

Reload ToC

Reset DB

Clean Cache

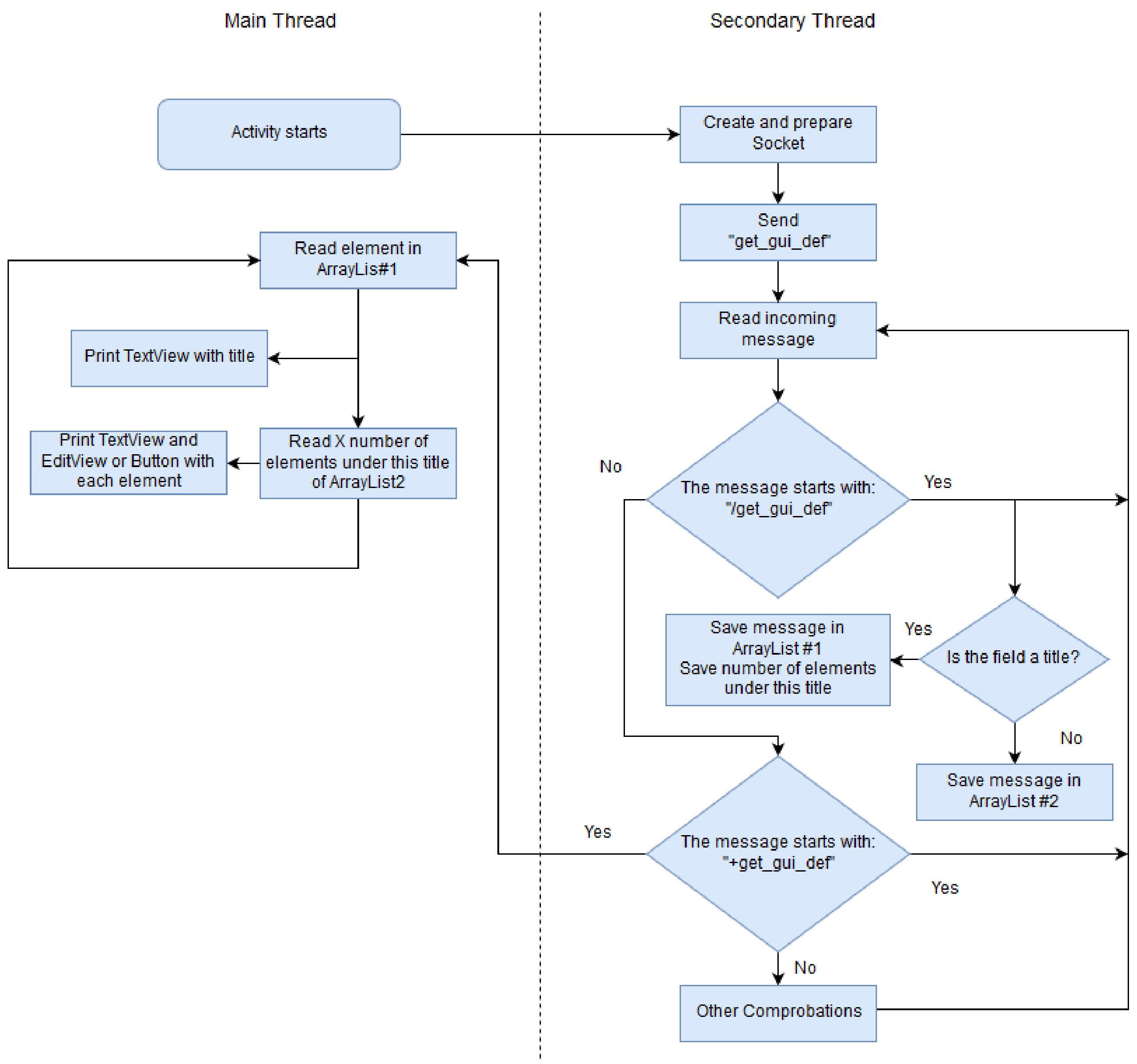
Figure 45: Different extended Setting Screen, the screen on the left corresponds to a standard user, the screen

on the right correspond to the user sysops

Some important items in Settings screen are:

- Current User: Username that is logged in this session.
- Unit Name: This field can have either “Flight” or “Ground” values and informs about for which environment this core is prepared to work (ISS or control centre).
- Sync Server IP: When starting a collaboration session, this value is the IP where the system sends the messages to start it.
- Display Orientation: This value is related with the screen and in which direction the app is oriented, left or right arm of the astronaut.
- Video Camera Device: It is the path where mobiPV core looks for the device used when recording videos or taking pictures. In this case dev/video3 corresponds to the USB camera, dev/video0 would be the external camera and dev/video1 is the internal one.
- iPV Server URL: This is the server address where the core downloads the whole list of procedures.
- iPV Server Authentication String: This field contains the username and password to login in the iPV server to have access to the procedures.

As it is the common procedure in this app, the screen is filled with the information obtained from the core after sending one message that asks for the information. The whole process is indicated in the next flowchart (Figure 46)



Main Thread
Secondary Thread
Activity starts
Create and prepare Socket
Send "get_gui_def"
Read incoming message
Read element in ArrayLis#1
Print TextView with title
Print TextView and EditView or Button with each element
Read X number of elements under this title of ArrayList2
No
The message starts with: "/get_gui_def"
Yes
Save message in ArrayList #1 Save number of elements under this title
Yes
Is the field a title?
No
Save message in ArrayList #2
Yes
The message starts with: "+get_gui_def"
Yes
No
Other Comprobations

Figure 46: Detailed flowchart of Settings.

The whole process in Figure 46 starts with the beginning of the activity. Immediately, in the secondary thread, the socket is created and connected as explained in Figure 27, once the connection is stablished, in the secondary thread, one message is sent to the core: “get_gui_def” and the system starts to listen to the core without doing anything else. There will be two possible answers for this message as it is shown in Table 1. Some messages starting with “/get_gui_def”, and another final message starting with “+get_gui_def”.

To make the printing of the different fields easier, the “/get_gui_def” messages have been divided into two groups and three ArrayLists are created for storing all the information, here is the explanation of the process with examples:

Suppose that the message arrives:

```
/get_gui_def 2|SYS.UNIT|Unit Information|UNKNOWN|(null)|0|,
```

the number 2, immediately after the title of the message tells the program that this field is a title (Table 4) in the first verification ("Is the field a title" diamond in Figure 46), then the system stores the text of the message in the ArrayList #1 and it also stores the value "0" in the auxiliary ArrayList created for knowing how many items are there in each title. Although this first "0" does not have any sense because it is the first one. Then a second message arrives:

/get_gui_def 1|SYS.UNIT.USER|Current User|UNKNOWN|Crew|0|,

in this case the verification in "Is the field a title" diamond gives a "No" as an answer because of the number "1" after the message title, so the system stores this message in the ArrayList #2 and starts a counter with the value "1". This counter is increased every time a message is stored in ArrayList #2 and is set back to "0" when a message is stored in ArrayList #1. The value of this counter is the value stored in the auxiliary ArrayList that helps the system when building the view of the screen.

After all the "/get_gui_def" have arrived, the "+get_gui_def" arrives and the secondary thread calls the main thread and a loop starts. Within this loop all the screen is built when the reading of all the titles stored in ArrayList #1 is done. When the first element in the ArrayList is read, one TextView is printed with the text in the "Label" ID parameter of the message (Table 4). Next, another loop then starts and the number of iterations is the corresponding number stored in the auxiliary ArrayList. In every iteration the following verifications are made, completing with these flowcharts the information given in Figure 46

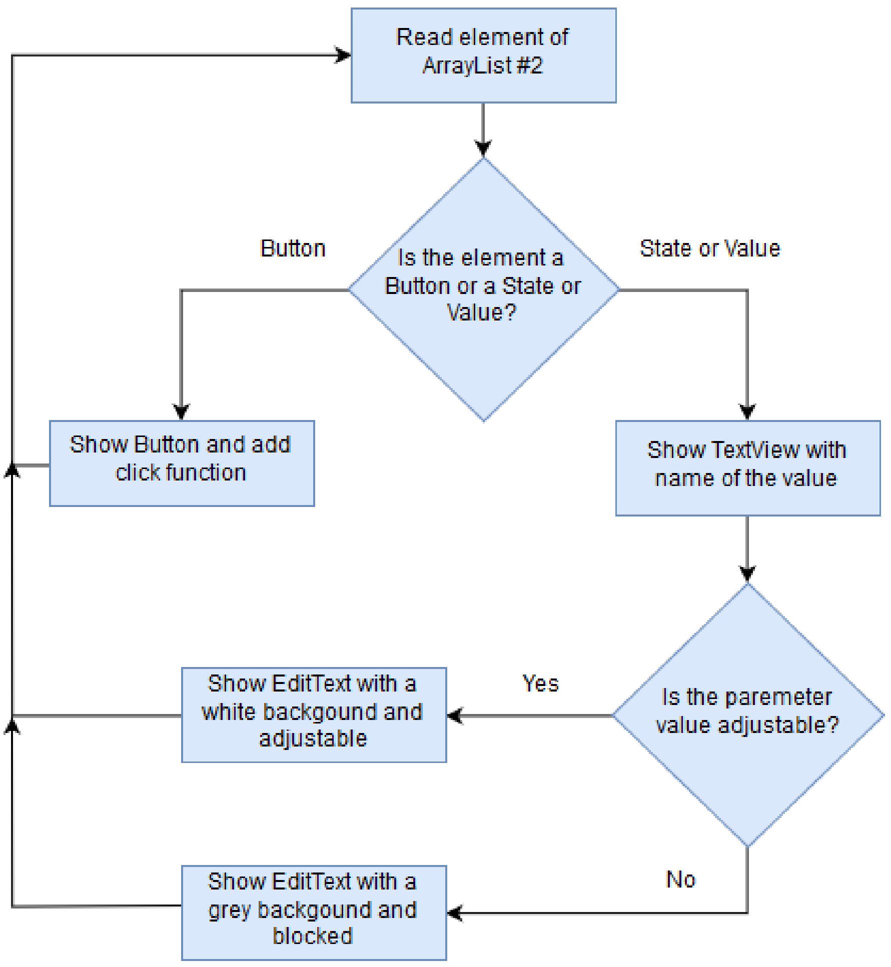

Read element of ArrayList #2
Is the element a Button or a State or Value?
Button
State or Value
Show Button and add click function
Show TextView with name of the value
Is the paremeter value adjustable?
Yes
Show EditText with a white backgound and adjustable
No
Show EditText with a grey backgound and blocked

Figure 47: Detailed flowchart of ArrayList #2 reading and view items printing in Settings activity.

In Figure 47 the process in which the Android GUI prints the elements that are not titles is explained. When the second loop starts (top box), the program looks into the message if the first field after the message title is either a "3" or a "1" or "0". These are the two possibilities:

A message containing "1" or "0" arrives:

/get_gui_def 1|SYS.VIDEO.CAM_RES|Video camera resolution|UNKNOWN|320x240|1|,

in this case the number "1", in "Type of entry" position, corresponds to a State or Value, so a TextView with the information contained in "Label" position is displayed. Afterwards, the program checks the "Configuration parameter value".

As mentioned in Table 4, number "1" corresponds to an editable value and "0" corresponds to a non-editable value. For the previous message, the "Video camera resolution" can be modified and an editable EditText is deployed with a white background.

Another message, also belonging to State or Value group is given:

/get_gui_def 1|SYS.NET.CORE_IP|Core IP|UNKNOWN|10.8.0.1|0|, as the message include a “0” in its last field, the item is not editable and a blocked EditText is displayed with a grey background.

Next flowchart shows how the EditText works when one field is modified.

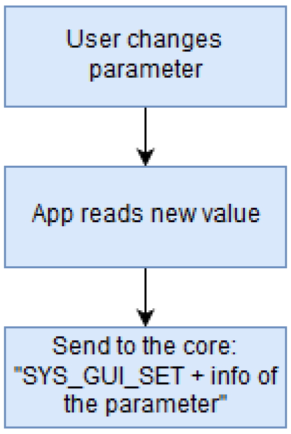


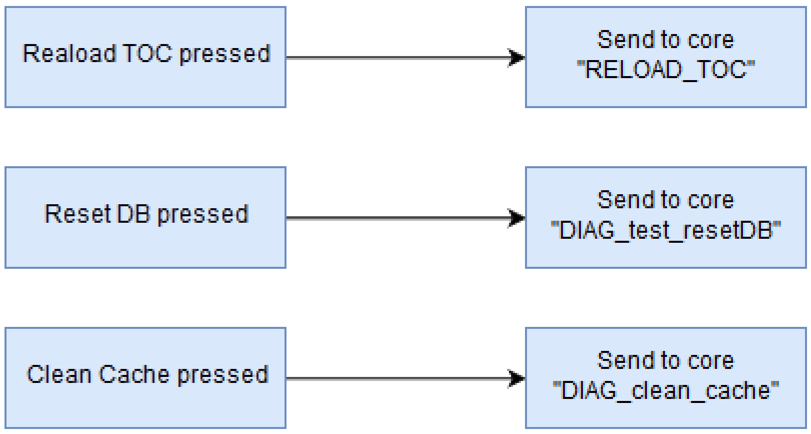
Figure 48: Detailed flowchart of changing parameter value operation in Settings activity.

There are three main steps in Figure 48, the process starts when the user decides to change one of the values in the app. The app notices the change through the listener “setOnFocusChangeListener” that is one of the operations that can be applied to any View as the EditText is [25]. When the listener is activated due to the parameter change, the app reads and stores in a variable the new value of the EditText and then builds a message with it, that is sent to the core. The content of the message follows the structure in Table 5.

With this information all the State and Value part of the first conditional in Figure 47 is explained. Moving into the rest of the flowchart, now a message containing “3” arrives:

/get_gui_def 3|SYS.RESET_DB|Reset DB|UNKNOWN|(null)|1|,

as the number is 3, this parameter corresponds to a button, then the app prints a button, with the text inside the field (“Reset DB” in this case as an example). The program also gives a “onClick” listener to the button [26]. This “onClick” listener depends on the same text in the “Label” field of the message. The next figure shows the flowchart of the three buttons and their listeners.


Reaload TOC pressed
Send to core "RELOAD_TOC"
Reset DB pressed
Send to core "DIAG_test_resetDB"
Clean Cache pressed
Send to core "DIAG_clean_cache"
Figure 49: Detailed flowchart of button listeners in Settings activity.

The three starting processes in Figure 49 are the different "onClick" listeners applied to the buttons.

- Reload TOC: Using the message "REALOAD_TOC", the app asks the system to download the list of procedures and the folders containing them. The procedures are download from the iPV Server URL that is one of the items of Settings (Figure 45). The user and password used to authenticate in this server are also items in Settings within the label iPV Server Authentication String. Once the system has been authenticated downloads the procedure tree as a JSON file.
- Reset DB: Using the message "DIAG_test_resetDB", the app asks the system to reset the database where all the activity of the different users is stored, as for example, the notes that each one takes.
- Clean Cache: Using "DIAG_clean_cache", the app asks the system to clean its cache, releasing then resources.

Diagnostics

The next screen is Diagnostics. This activity asks the core to check whether one component of the system is not working properly and prints the result of this analysis. Depending of the results of the analysis there are three kind of states for a component:

- INFO: Some information about the component for example an IP address for network component.
- OK: The component works perfectly
- FAIL: There is something wrong in this component

The flow chart of this activity is shown in Figure 50.

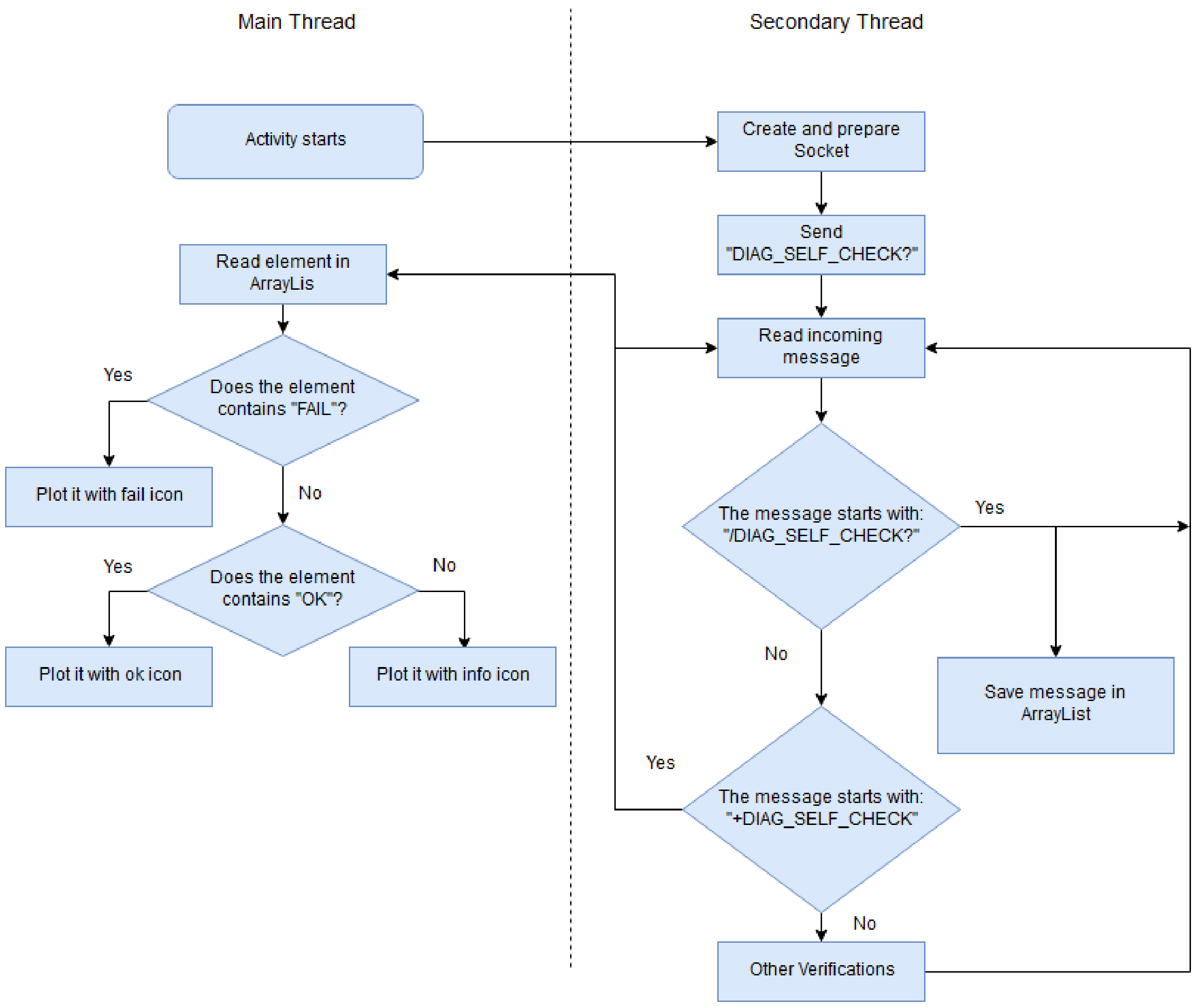


Figure 50: Detailed flow chart of Diagnostics activity

Starting reading the flowchart from the top left part, when the activity is started, a socket is created and prepared and then the app sends the command "DIAG_SELF_CHECK?" to the core, this command runs a diagnostics analysis and gets as response from the core, the different checked parameters with their status. The app starts reading the different messages from the core with the states of the components and stores them in an ArrayList, for a future use of it. When the last message related with the analysis arrives ("+DIAG_SELF_CHECK"), all the operations are sent to the main thread, which is the one that interacts with the views. So every message stored in the ArrayList is read, one by one, and either the component state is OK, FAIL or INFO, it is plotted accompanied by a green tick icon, a red cross icon or a blue information icon.

Logout

Logout tab asks the system to finish the user session. This process can be called from all of the states but SplashScreen of Login and is performed by sending to the core a "logout" message. Then, all the activities implement a listener in the secondary thread. The listener is implicit in the reading messages from the core process. If the system identifies the message "+LOGOUT", automatically jumps to Login Activity and its screen. The flowchart of this operation is in Figure 51.

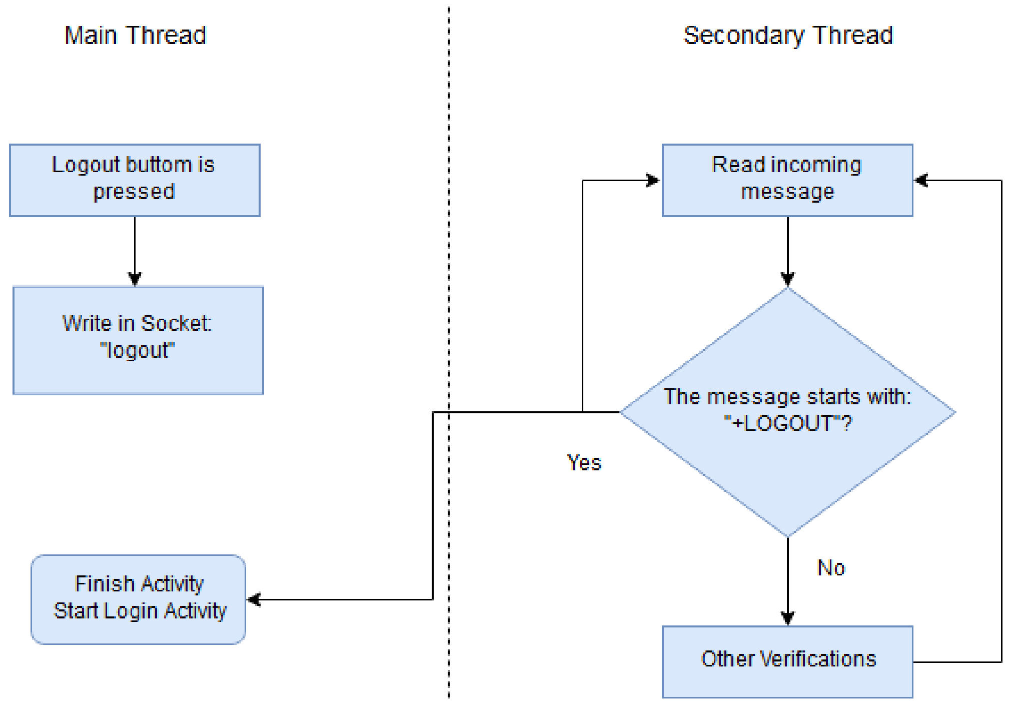

Main Thread
Secondary Thread
Logout buttom is pressed
Write in Socket: "logout"
Finish Activity Start Login Activity
Read incoming message
The message starts with: "+LOGOUT"?
Yes
No
Other Verifications

Figure 51: Flowchart of logout selection.

3.3.4 Main Menu Screen

Back to this screen (Figure 40), Main Menu Screen implements two main images and a description below each one. The image on the left shows a document and a loupe over it, with the description "local procedures" under them. On the right, the image shows an icon of the ISS, with the description "iPV flight library" under it.

When the user presses on one of the two possibilities, a new screen is open and a new activity starts, that is why maybe calling this section just "Main Menu Screen" is not really accurate, but all the screen belongs to same group and to the same architectural state.

MainMenu is probably the simplest activity in terms of the logic behind the View. The flowchart can be seen in Figure 52

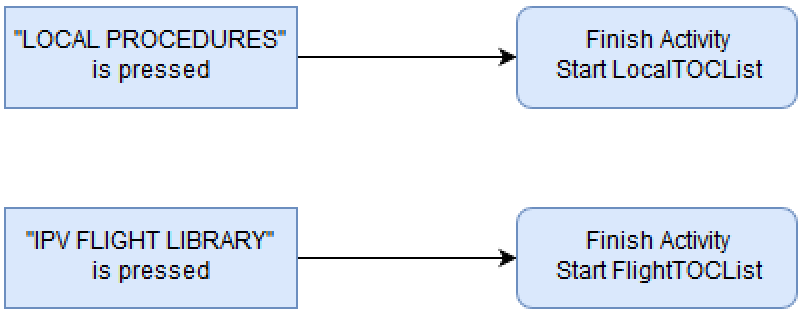

"LOCAL PROCEDURES" is pressed
Finish Activity Start LocalTOCList
"IPV FLIGHT LIBRARY" is pressed
Finish Activity Start FlightTOCList

Figure 52: Flowchart of MainMenu Activity.

As mentioned before and shown in Figure 52, from MainMenu either "LocalTOCList" or "FlightTOCList" activities can be started. Both of them implement a list of procedures retrieved from the core and the difference reside in from where the list is taken.

- LocalTOCList: The procedure list is obtained from a JSON file where all the internally procedures are included. These internal procedures are put there by the mobiPV developer and are usually implemented to test the software in certain experiments, as the two in NEEMO or in the ISS mentioned in 1.1 Current state of mobiPV.
- FlightTOCList: The procedure list is obtained from a JSON file that is created after a reading of the iPV server, a "platform developed by Skytek to assist astronauts in their daily execution of procedures on board the International Space Station." [27]. For successfully connection to this platform, the fields in "Settings" where the IP address and the user and password fields have to be the correct ones.

The logical processes in both activities are the same and are represented in Figure 53.

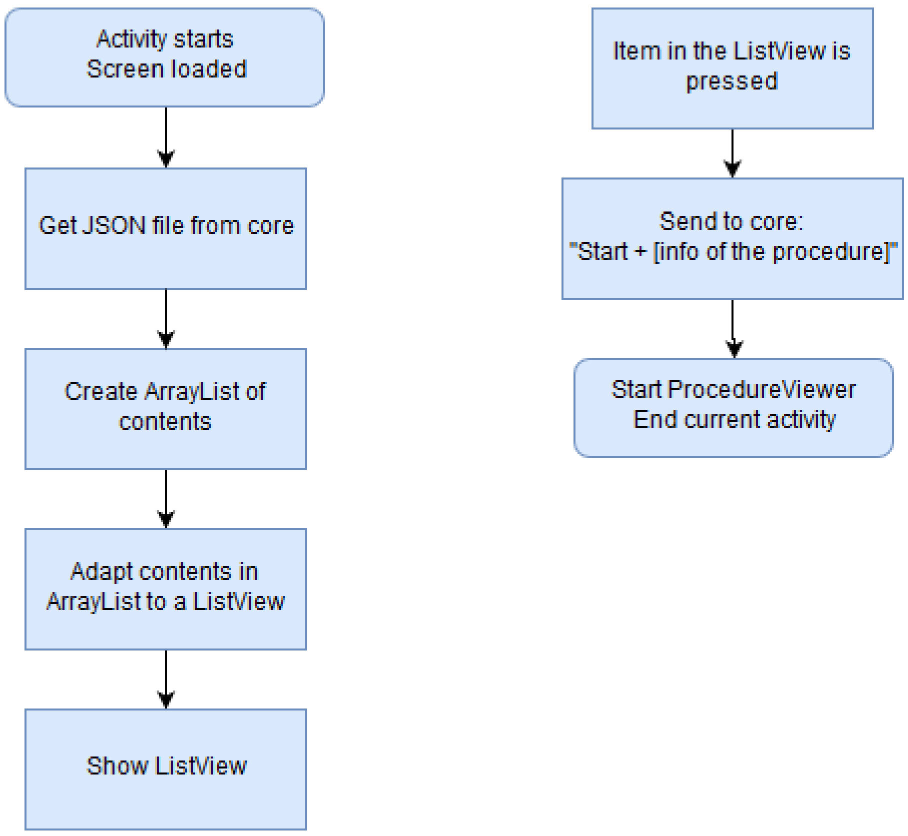

Activity starts
Screen loaded
Get JSON file from core
Create ArrayList of contents
Adapt contents in ArrayList to a ListView
Show ListView
Item in the ListView is pressed
Send to core:
"Start + [info of the procedure]"
Start ProcedureViewer
End current activity

Figure 53: Flowchart procedure list activity. The round box represents the start and end of the Activity; the normal boxes represent the processes happening within the activity.

When the activity starts and the screen is loaded, the app looks for the JSON file. The process of retrieving the JSON file is similar to the process shown in Figure 31, when the procedure goes from Skytek to mobiPV and then to the Android GUI in different formats, in this case the JSON is built once mobiPV reads on Skytek and do not change its shape or format when arrives to the app. See Figure 54 for a graphical representation.

By using the URL http://127.0.0.1:8080/cache/toc.json, the app asks for the JSON, located in the core and downloads it.

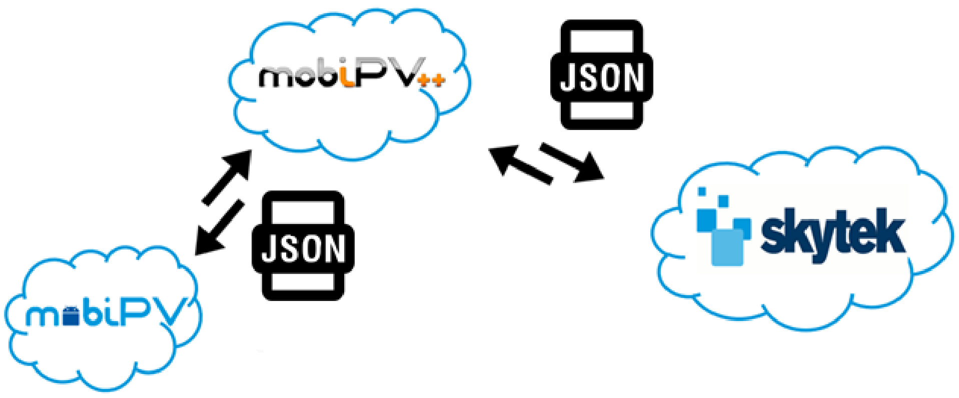


Figure 54: Procedure list downloading process. From right to left, the Skytek cloud represents the server where all the procedures are stored in xml format. The cloud where mobiPV system represents the system running in the Linux core, that reads the procedure list and generates a JSON file, the last cloud represents the mobiPV Android User Interface that receives the JSON file when it asks for it.

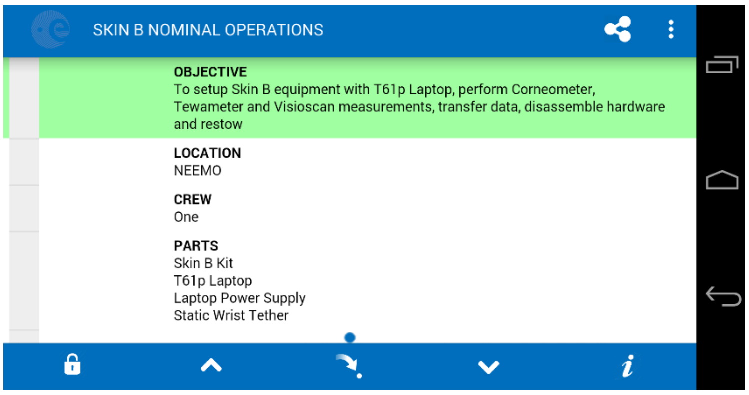
When the JSON file is loaded in the memory of the app, then all its items, one by one, are stored in objects. These objects are prepared to contain the fields of the JSON file for every single procedure. Then the objects are inserted in one special array.

When all the procedures and their description elements are stored in the array, by using a loop every component of the array is adapted to the ListView, and for everyone, a Listener is set.

The listener corresponds to the left part of Figure 54, where, if one of the elements of the ListView is pressed the listener starts pre-configured process. This process consists of sending to the core

3.3.5 Procedure Viewer Screen

The Procedure Viewer is the most important screen in the app (see screen in Figure 55) and, as it is expected, it is the most complex activity. The activity implements several technologies and the complexity resides in how all of them work together and how the communication between them happens. In Procedure Viewer activity is very common the use of listeners that are activated after one event happens as one click from the user or one message is received from the core, this feature makes the logic even more difficult to coordinate in order to not have a failure in the system. In this section the complex solution is described in an organized way: from how the architecture that mixes the technologies is formed, to the typical flowchart of the system, for this section is divided feature by feature.



SKIN B NOMINAL OPERATIONS

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Figure 55: ProcViewer screen

Beginning with the global aspect, the main architecture is compound by two main technologies: Android app and a WebView. Related with the WebView there are another three important components that have an important role: the HTML file, the stylesheet CSS file [28] and the JavaScript functions file.

The main reason for this mix of technologies is that, as a first approach to the solution of the project, the part of the Procedure Viewer that shows the procedure was chosen to still being as a web format. The main reason is that though the style sheets, the CSS, the procedure has an appearance that fits perfectly with required standards for procedures on the ISS and the stylesheets are already made.

Despite of the procedure is still shown as a web format and some listeners, that reside within the HTML which is shown in the WebView, are activated in an additional JavaScript file, the main logic components are implemented in Android native code. Still the communication and interaction between them is necessary and Figure 56 shows a scheme of the resulting architecture.

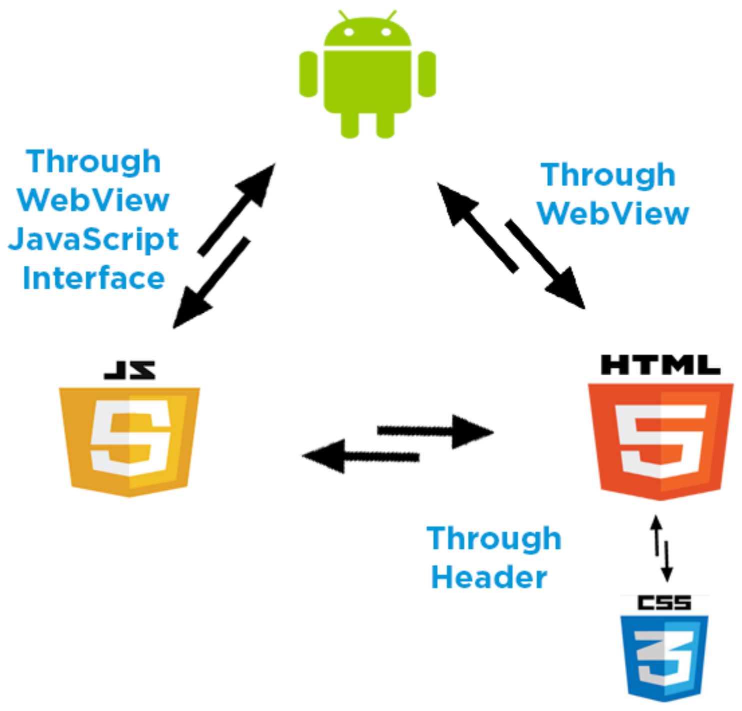


Figure 56: Architecture of Procedure Viewer activity. On the top of the figure there is the android app, on the bottom left the additional JavaScript file and on the bottom right the HTML file that accesses to a CSS style sheet.

The description of Figure 56 is the following. When the activity is loaded and has the path to the HTML file, it creates a WebView and loads the HTML file onto it. Of course this HTML file contains a header that tells the file where the necessary CSS style sheets and the JavaScript file is. By this way when the WebView loads a file that implements the required styles to fulfil the procedure standards and also the listener functions when the user clicks on one specific part of the document are implemented. In Figure 57, there is a highly detailed flowchart of the process of getting the html and process it until it is finally loaded in the WebView.

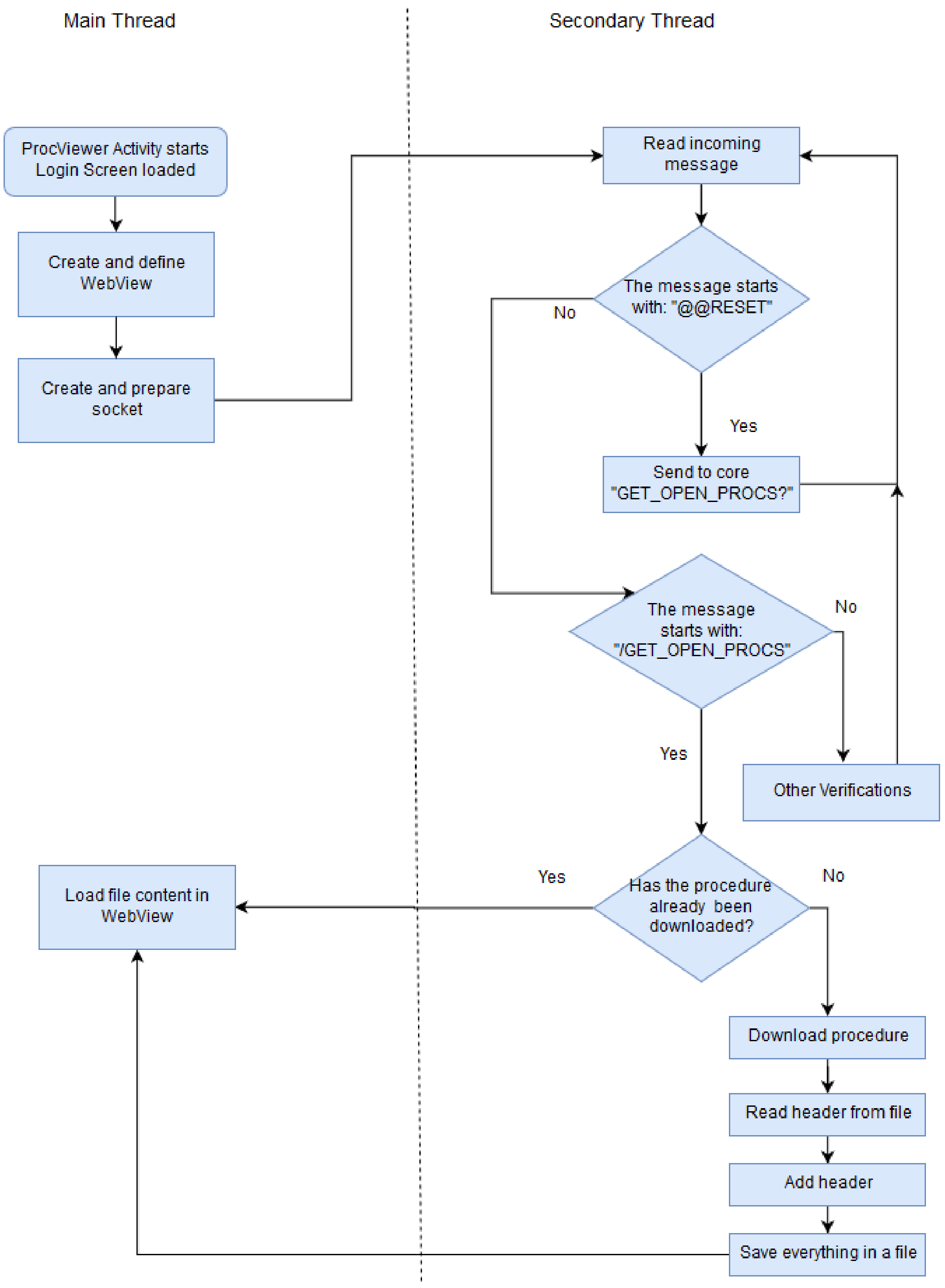

Main Thread
Secondary Thread
ProcViewer Activity starts Login Screen loaded
Create and define WebView
Create and prepare socket
Read incoming message
The message starts with: "@@RESET"
No
Yes
Send to core "GET_OPEN_PROCS?"
The message starts with: "/GET_OPEN_PROCS"
No
Yes
Other Verifications
Has the procedure already been downloaded?
Yes
No
Load file content in WebView
Download procedure
Read header from file
Add header
Save everything in a file

Figure 57: Detailed flowchart of loading WebView process.

Starting from the right top round box, when ProcViewer Activity starts and loads the screen, the process moves into creating the WebView and linking it with the View that corresponds to this activity. After this and as usual in every activity, the socket is created. When the confirmation message from the core that confirms the connection arrives (`@@RESET`) the message "`GET_OPEN_PROCS?`" is sent (more information about this command in Table 7).

By this way, the different open procedures are received and the active one is selected. In addition to the selection, the process checks if this same procedure has been already downloaded and saved in a file in the smartphone. Whether it is saved, the system loads the file in the WebView. Conversely, if it is the first time the app opens this procedure, because it is not internally stored, the app looks for it in the given path by the command "`GET_OPEN_PROCS?`" and saves it in a variable. Then the process also reads from its internal files directory the header to be added to the procedure. When both, header first and secondly procedure are together, the app saves them as a HTML file in the internal files directory and loads it in the WebView.

The remaining part is how the communication between the Android native app and the JavaScript is done. Through a WebViewJavaScriptInterface [7] this interaction is made and every time that the user presses one item in the HTML labelled with a function that activates a JavaScript function, this last one calls one method in Android that finishes the request.

There is a flow chart of the process just described in Figure 58

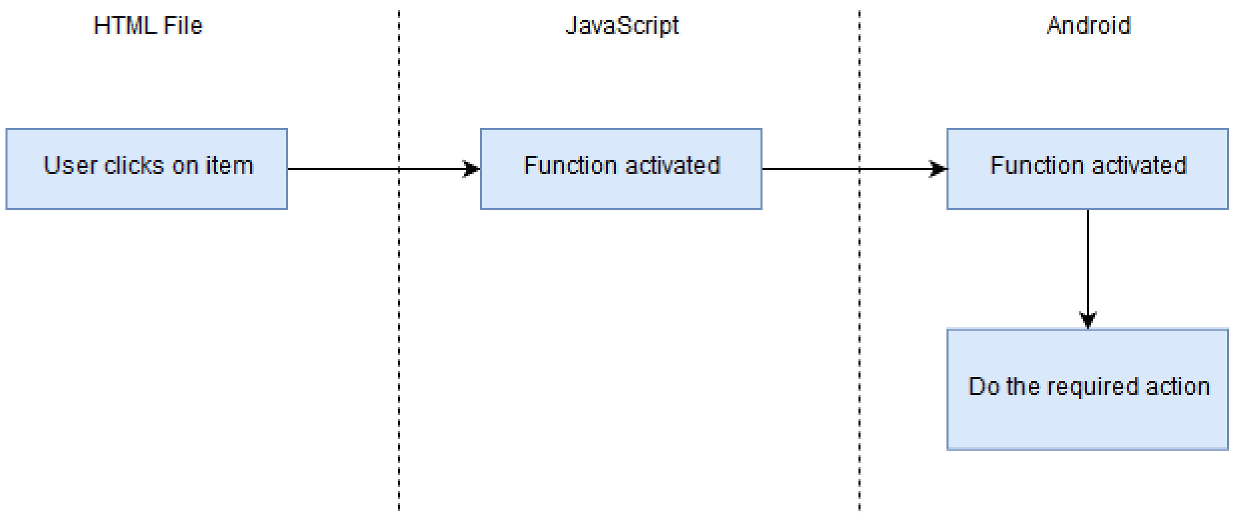


Figure 58: Flowchart of an action starting in the HTML.

Logically, this process can also happen in the opposite direction, if one listener in the Android native code is activated and is programmed to change an item in the HTML file, then is the app the one that activates a function in the JavaScript file and this last one modifies one attribute of the HTML (Figure 59).

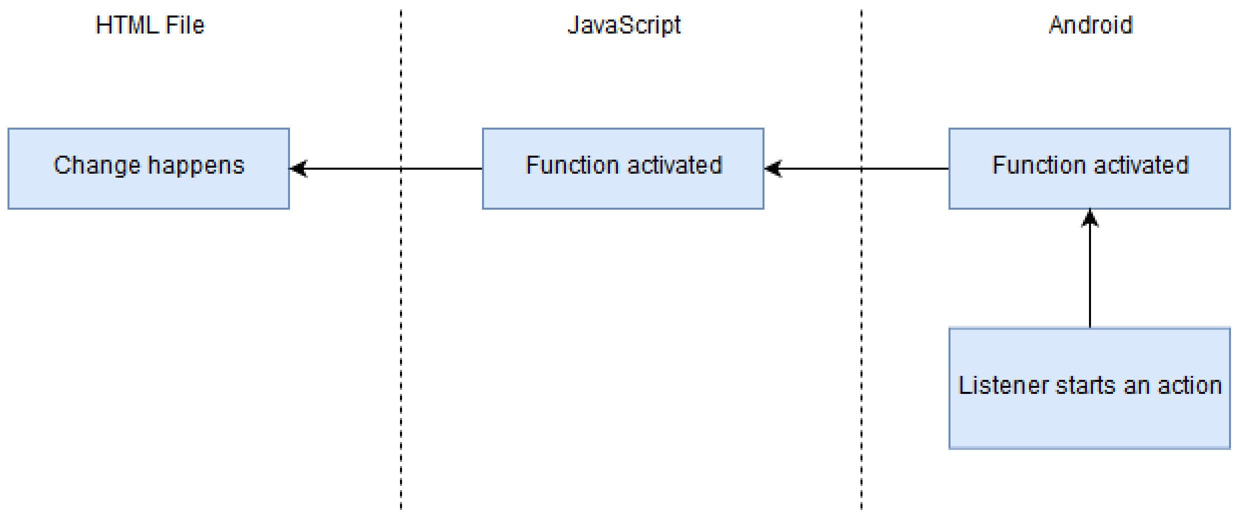


Figure 59: Flowchart of an action starting in the app.

Until here the processes of obtaining the procedure, how it is saved, loaded and shown to the user are explained. As well as from where the procedure is obtained and the logic behind this process. It is time to go deeper into the different features that have been added to the Procedure Viewer in order to make it possible interact with the environment and create a cooperative work place where the astronaut is not the only one using the system.

The first feature to explain is the green marker that can be seen in Figure 55 over one step in the procedure. This marker indicates in which step the astronaut is working or which step was the last one he was working before changing the procedure or accessing to other screen. The procedure the app follows to obtain the step where to place the green marker is based in two concepts (see also Figure 60 for a detailed graph):

- The core writes a message to the app when the procedure is open (after the app working in Main Menu Screen sends the message "`start`"). The mentioned message that indicates the step is read by the app still in Main Menu Activity, so when Procedure Viewer Activity starts has to receive in some way the information. This is done through the same Intent that is used for changing the activity and with the method "`putExtra`" in one side and the method "`getExtras`" in the other side. Once the step name is stored in a variable it is sent to a function in the JavaScript.
- The body of HTML document that is the procedure is divided in the steps, where every step corresponds to a line in the body, as well as every line is separated from the others by the label "`<div>`" and has its own id. By this way when ProcViewer Activity send the step id to the JavaScript function, this last one finds easily the step in the HTML document and adds the green background to the line.

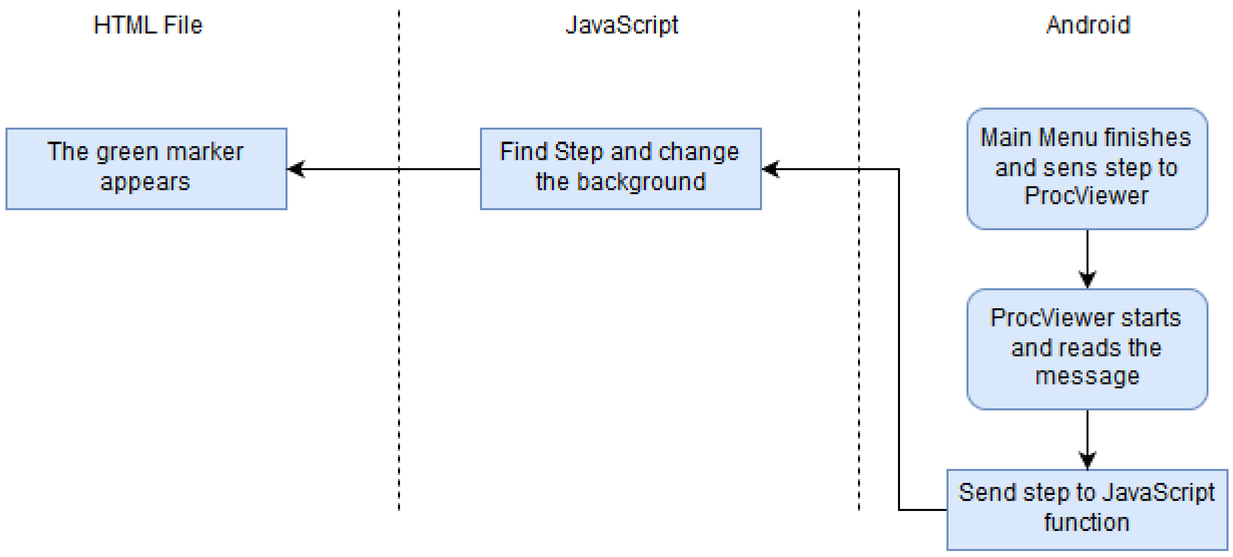

HTML File
JavaScript
Android
The green marker appears
Find Step and change the background
Main Menu finishes and sens step to ProcViewer
ProcViewer starts and reads the message
Send step to JavaScript function

Figure 60: Flowchart of the green marker positioning.

Related with the step marker positioning is the first automatic scrolling to the step. When the procedure is started for the first time, the screen remains the same due to the marked step is the first one. Although when it is not the first time this procedure is open and was not completed, the green marker is somewhere in the middle of the procedure. Then the automatic scrolling moves the screen until the central step is the one marked. This process runs in parallel with the previous process in Figure 60 with the difference that the JavaScript function to do the automatic scrolling is a different one.

ProcViewer not only shows a procedure, it is also possible to navigate through it, for the navigation step by step there are two main buttons at the bottom of the screen in a blue bar denoted with an arrow (See Figure 55). These navigation step by step buttons implement the change from one step to the next one or previous one by clicking on them. The right arrow in the figure represents move forward to the end and the left arrow represents going back to the beginning of the procedure.

The navigation is achieved in two stages. In the first one, after the user presses one of the buttons a message is sent to the core. "`next`" with the arrow on the right and "`previous`" with the arrow on the left (See Table 7). In the second stage of the process the core sends a confirmation answer "`+NEXT`" or "`+PREVIOUS`" and a listener in the app starts and calls the JavaScript method that moves the green market to the following or prior step. Similarly to the previous part when the screen was scrolling automatically, in this case when the positive answer from the core is received the JavaScript method to scroll is called. The main reason why the system waits for the answer is because by this way both systems, core and app, are coordinated and at the same step at the same time. See Figure 61 for a detailed graph of the process.

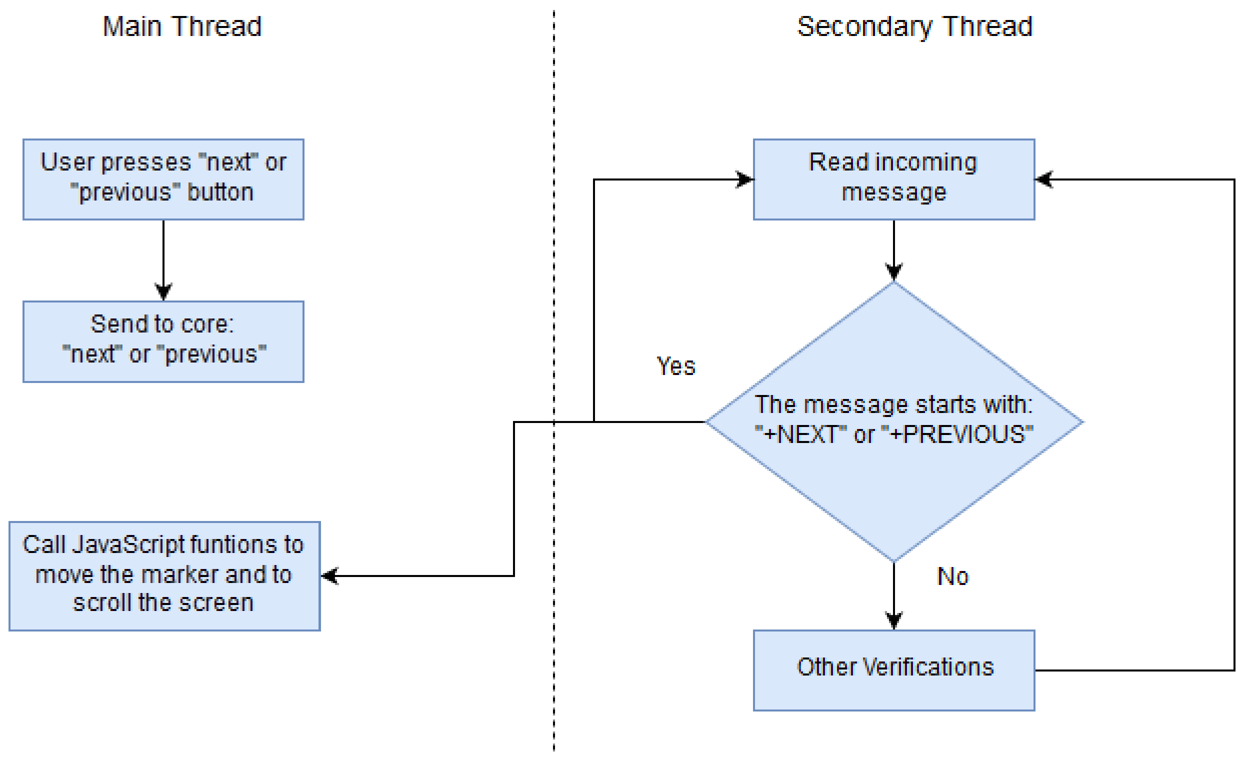

Main Thread
Secondary Thread
User presses "next" or "previous" button
Send to core: "next" or "previous"
Call JavaScript funtions to move the marker and to scroll the screen
Read incoming message
The message starts with: "+NEXT" or "+PREVIOUS"
Yes
No
Other Verifications

Figure 61: Detailed flowchart of changing the step of the procedure step by step.

Using "next" and "previous" buttons is not the only way of navigating through the procedure. The two buttons implement a way of moving step by step but there are also two ways of moving from one step to another that are not together.

The first way of doing it takes more complexity to implement and is accessed by pressing one of the buttons in the lower bar of the app (the icon with the arrow and the circle that means "go to"), then one dialog pops up and shows the list of steps within the procedure.

The logic behind this solution is not trivial since it starts from the moment that the procedure in HTML format is downloaded from the core. At this moment, after checking that the processed procedure by the app is not stored in the internal memory of the app one process starts. This process consists of a method that reads every line of the HTML file looking for the label of a specific HTML class. This label is "`nmp_num`" and delimit a step which is numbered, in contrast to many other steps that are compound for example by an empty gap or information as objectives or location where the procedure has to be run (the steps shown in Figure 55, for example do not have the label "nmp_num"). Coming back to the process, when a line with this label is found the app looks for the step ID, the number of the step and the description of it are extracted and stored in an ArrayList. Here is an example of one line of the HTML containing this label and the needed information.

`<div class = "odf2_minorStep" id = " minor_step_`**`neemo@step_1`**` " > < div` [more HTML code] `< div class = "nmp_S" > < div class = " nmp_location_inner " > Sleeves </div> <div class = "nmp_num" >` **`1`** `</div> <div class = "nmp_step_title" >` **`PREPARING FOR SKIN B`** `</div>          </div> </div> </div>` [more HTML code]

In the previous HTML code the step ID, the step number and the step information are in bold and this is the content that the app looks for and extracts by implementing some String methods as "`split`", "`substring`" or "`contains`" [29].

Additionally to store this information in an ArrayList, it is also stored in two different files in the internal memory of the app, so the next time the procedure is loaded in the first conditional where the app checks if the procedure is within the app directories, the step list is directly read from there. The file corresponding to the step number and the step description is written with the following format:

```
1 This is an example of step/2 This is the second example of a step/3 …
```

On the other hand, the file corresponding to the step IDs is written like that:

```
neemo@step_1|neemo@step_2|neemo@step_3
```

For an easier reading of the file the next time the procedure is loaded the step number and the step description are separated from the next one by a slash. For the steps, one is isolated by the next one by a vertical bar.

By the time that all the numbered steps are stored there are no more processes until the user presses the mentioned icon before. At this moment, the app reads the ArrayList and prints it in the Dialog, giving at the same time a function to every step. This function is to send the message "`GOTO_STEP` + procedure name + stepID". See Table 1 and Table 8. Finally, as before with "previous" and "next" options, when the confirmation message from the core is received, the step is changed and the screen is scrolled by calling the JavaScript method. See Figure 62 for a detailed graph of the process.

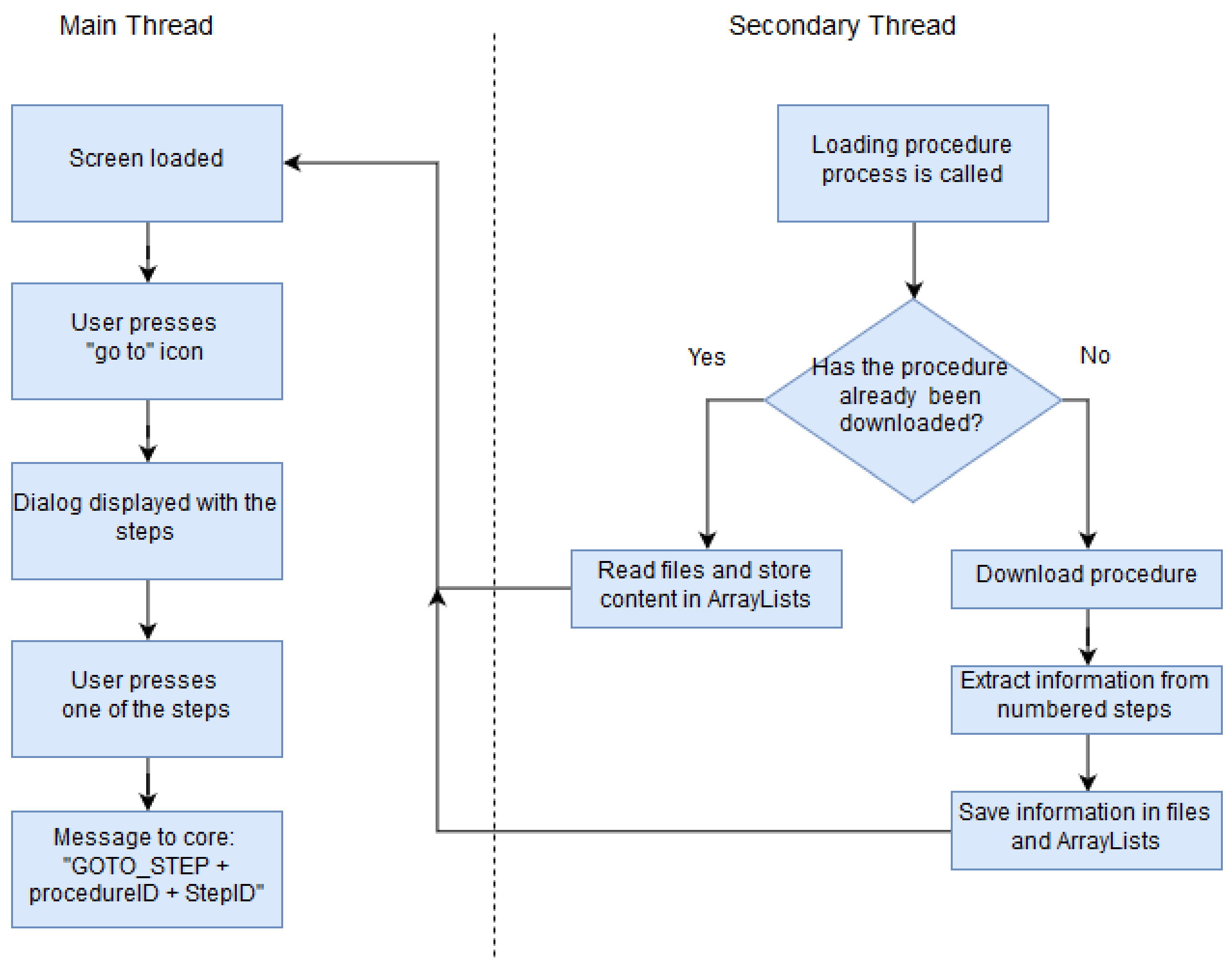

Main Thread
Secondary Thread
Screen loaded
User presses "go to" icon
Dialog displayed with the steps
User presses one of the steps
Message to core: "GOTO_STEP + procedureID + StepID"
Loading procedure process is called
Has the procedure already been downloaded?
Yes
No
Read files and store content in ArrayLists
Download procedure
Extract information from numbered steps
Save information in files and ArrayLists

Figure 62: Detailed flowchart of changing the step of the procedure using "go to" button.

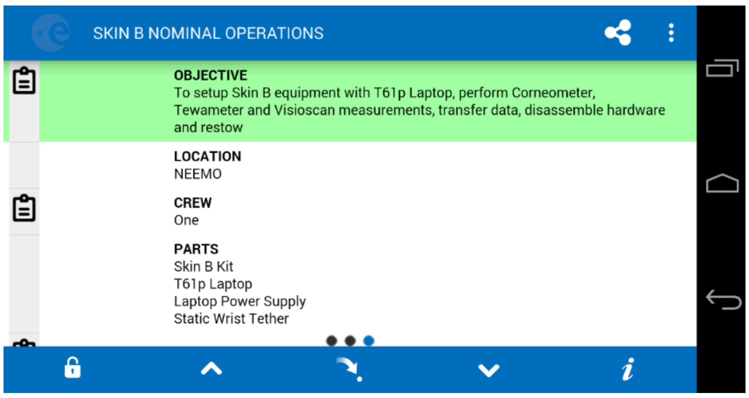
The last navigation tool is implemented within the HTML file and the listener is activated in the JavaScript function after the event of pressing some specific text that is in blue and underlined along the procedure. This text imitates hypertext in a standard web and makes the green marker jump to a different step.

The logic is similar to the general example in Figure 58, where the action starts in the HTML, so one JavaScript function is activated and then, it calls a method in the Android activity which sends the same message "`GOTO_STEP` + procedure name + stepID" to the core. The difference now from the previous navigation method is that the procedure name and the stepID are obtained from the HTML file by the JavaScript file since it is easier because there are many functions implemented in JavaScript to interact with HTML.

In the bottom bar, at the left, there is an icon represented by a lock. This icon implements a function that disables/enables the automatic screen scrolling. As it was said before, every time the green marker is moved, there is a method written just after the marker is moved, that scrolls the screen. Also, before the method and by using a conditional and a Boolean variable, there is a verification and if the automatic scrolling has been disabled, this method will not be called. When the user presses the button one message appears informing about the actual state of the automatic scrolling. The only action that is done after the button is pressed is changing the value of the Boolean variable from “true” to “false”.

Going now to the last icon of the bottom bar, the user can get some information from the procedure. This additional information is called metadata and it may contain information about the creator of the procedure; the origin of it; the creation date; the permissions, whether if it is public or private; a description; etc. This metadata is sent after the command “GET_PROC_METADATA?” which the app sends when the procedure is loaded at the beginning (see Table 1)

Just above of the button bar of Figure 55, there is a blue dot, the dots indicate the number of procedures that are open at the same time and the blue colour the position of the actual one in which the astronaut is working. This process is useful in case the user wants to move from one to another without leaving the activity ProcViewer just by swiping with the finger to the right or to the left. In Figure 63 there are three procedures open at the same time and the current one occupies the third position, if one left swipe is done, the screen changes and the current procedure is the one occupying the first position in the list.


SKIN B NOMINAL OPERATIONS

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Figure 63: ProcViewer screen with three procedures open at the same time, each one denoted with one dot at the bottom

This process of storing all the open procedures and allowing the user to move to them starts when the first “/GET_OPEN_PROCS” (see Table 1) is received at the beginning of the activity. For every open procedure received, one element is added to an ArrayList created for this process. Prior storing the open procedure in the ArrayList, the app checks if the procedure has been already added to it. If the procedure is not in the list, it is added and the system saves the position of the active procedure in a variable, that will be necessary later for placing the dots on the bottom of the screen and to do the swiping from one procedure to other. This all process happens in the Secondary Thread. See Figure 64 where all this information is displayed in a graph.

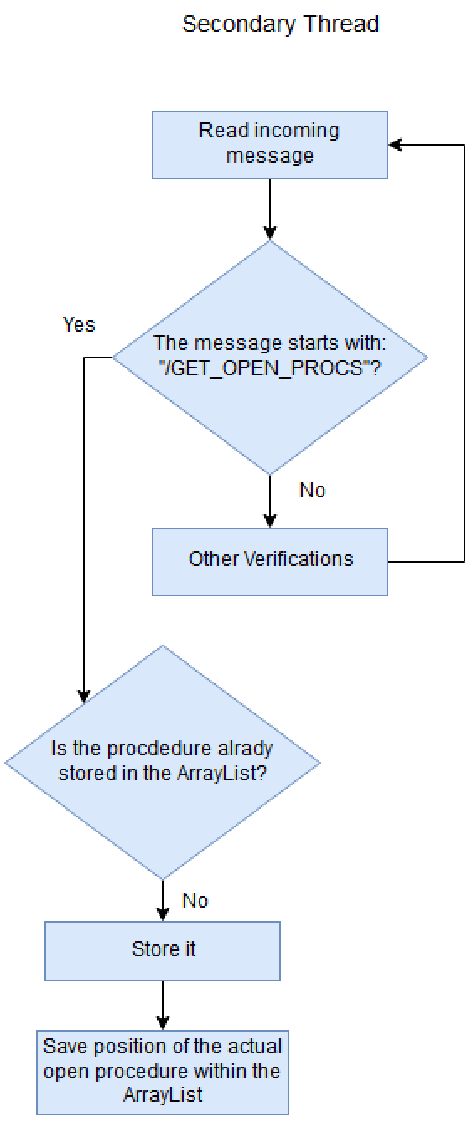

Secondary Thread
Read incoming message
The message starts with: "/GET_OPEN_PROCS"?
Yes
No
Other Verifications
Is the procdedure alrady stored in the ArrayList?
No
Store it
Save position of the actual open procedure within the ArrayList

Figure 64: Detailed flowchart of how the open procedures are stored for a subsequent use.

Once the last "`/GET_OPEN_PROCS`" is received, the system uses the command "`+GET_OPEN_PROCS`" to signal that all the open procedures have been sent and it is the final of this chain of messages. The app implements one listener that starts with this last message. This listener places one point for every open procedure and places the blue dot in the position where the open procedure is in the ArrayList. The placement of the dots happens in the main thread while the reading of the message and the activation of the listener happens in the secondary thread. See Figure 65 where all this information is displayed in a graph.

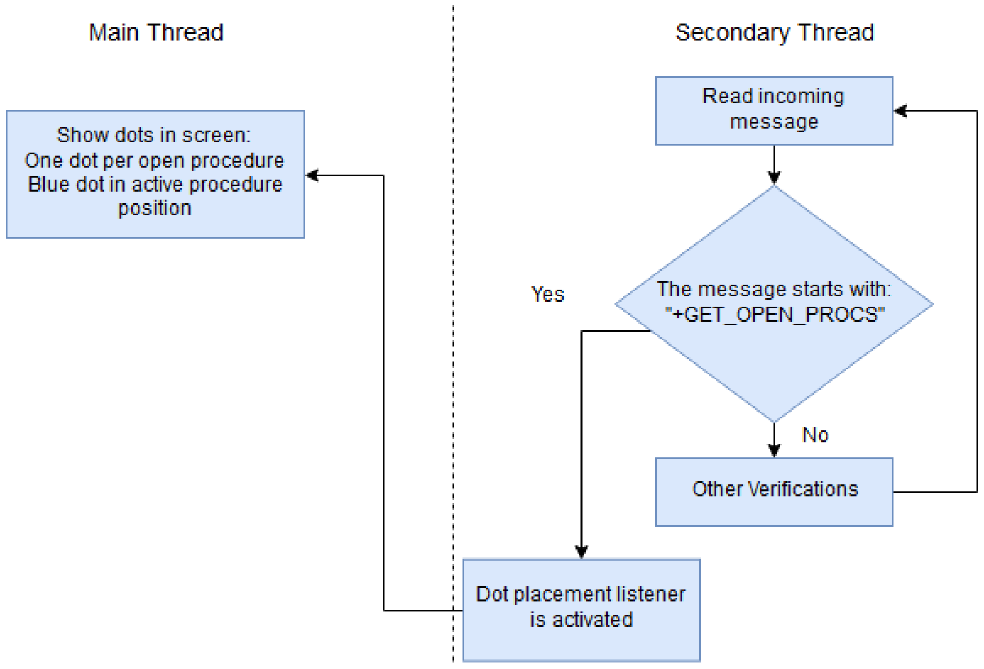

Main Thread
Secondary Thread
Read incoming message
Show dots in screen: One dot per open procedure Blue dot in active procedure position
Yes
The message starts with: "+GET_OPEN_PROCS"
No
Other Verifications
Dot placement listener is activated

Figure 65: Detailed flowchart that describes the placement of the dots in the screen.

The following activity related with moving by swiping from one procedure to another involves the user and is not activated until this person swipes right or left. If the movement is to the right it means that the procedure to be open is the next one in the previous one in the list of open procedures, if the active procedure occupies the first position in the ArrayList, then the next active procedure is the last one in the lest, making by this way a closed loop. If the user swipes left, the next active procedure is the next one in the list and if the active procedure is the last one, then the next active procedure is the first one in the list.

When the app knows which procedure is the next one to be active a "`SET_PROC`" command is sent followed by the procedure ID, for example, if the next one to be active is the "NEEMO SKIN B" the command would be: "`SET_PROC _neemo`". When the success answer for this command is received, the new procedure is loaded and the command "`GET_OPEN_PROCS`" is sent and then the process described in Figure 64 and Figure 65 for getting the active procedure position and placing the dots is repeated. See Figure 66 where all this information is displayed in a graph.

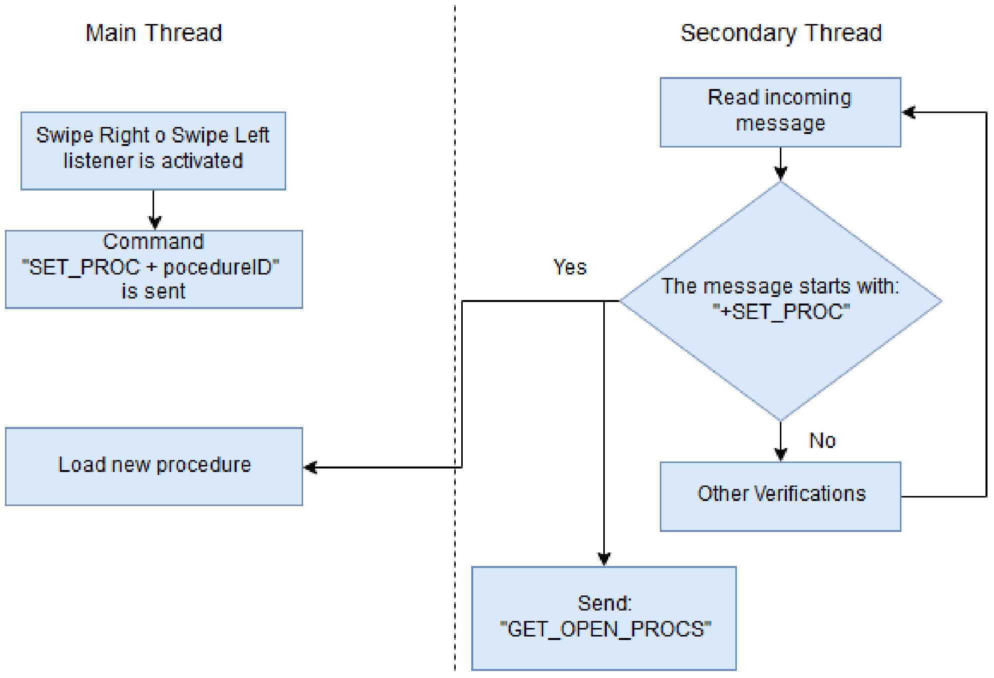
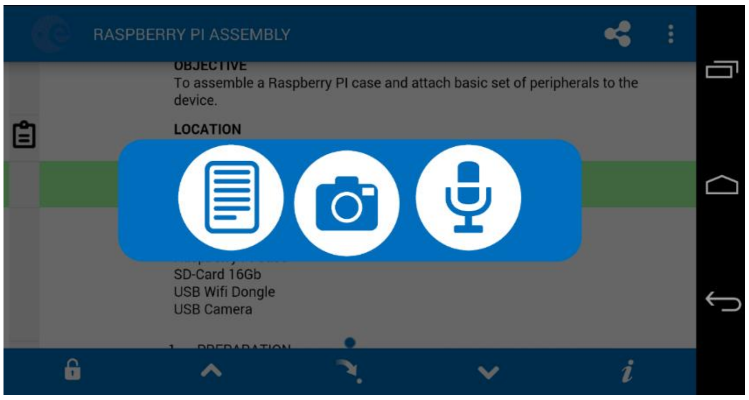

Main Thread
Secondary Thread
Swipe Right o Swipe Left listener is activated
Command "SET_PROC + pocedureID" is sent
Load new procedure
Read incoming message
The message starts with: "+SET_PROC"
Yes
No
Other Verifications
Send: "GET_OPEN_PROCS"

Figure 66: Detailed flowchart that describes the swiping listener.

The last feature to be explained for ProcViewer activity is the note taking. In every step of the procedure on the left of the text there is a small box in a darker colour. If there is a stored note for this step the note icon will appear (see Figure 63). If the user presses on this box, the Dialog in Figure 67 appears.



RASPBERRY PI ASSEMBLY

OBJECTIVE

To assemble a Raspberry PI case and attach basic set of peripherals to the device.

LOCATION

SD-Card 16Gb
USB Wifi Dongle
USB Camera

Figure 67: Note Taking Dialog

There are three kind of notes, everyone is represented by a ImageButton text notes (left ImageButton in Figure 67), image or video notes (centre ImageButton in Figure 67) and audio recording notes (right ImageButton in Figure 67). One by one these three possibilities are explained in the following lines.

- Text notes: These kind of notes consist of plain written text. The app implements an EditText box so that the user can write in it and a “Save” button to store the note. If there is already a saved note in the step, the app implements a TextView where the user can read it and a ImageButton with the shape of a trash can to remove it.

- Image and video notes: These kind of notes require the USB camera to work so after pressing the ImageButton in the middle a CameraView starts with two ImageButton on the right side. The one on the top is for recording videos and the one on the bottom is for taking photographs After pressing the video ImageButton it changes into a Stop ImageButton, the system is recording and will stop the video arter pressing again in the same position.
  When the user presses on a step note icon and there is a video recorded for this step the Dialog shows the name of the file in a TextView and below it a Button to play the video and an ImageButton with the shape of a trash can to remove it.
  If, on the other hand, the user presses on the photo ImageButton, a photograph is taken and the Dialog closes. When the user presses on a step note icon and there is a photograph note within it, the same Dialog is shown to the user. Below the three main ImageButtons a TextView with the title of the file appears, and below it a Button to open the Image and a ImageButton with the shape of a trash can to delete the note. After pressing the note to open the photo, one ImageView is shown with the taken picture.

- Audio notes: These notes require the microphone of the smartphone and after pressing the ImageButton the system starts to record environmental audio and a text dialog appears informing that the record has started. The ImageButton changes too into a stop button. After the user presses the stop button the dialog disappears and another text dialog pops up informing that the audio recording has been stopped.
  When the user presses on a step note icon and there is an audio recorded for this step, the Dialog shows the name of the file in a TextView and below it a Button to play the audio, a time line for the playing, a timer to follow the audio playing progession and an ImageButton with the shape of a trash to remove it.

Until here that was everything in relation with the views and the different features of the note taking. As a continuation of this part, it is time to study deeper the logic behind this process.

Starting with the opening of the procedure, once the view is loaded, the first related process with the note taking that starts is the note icon placement in those steps where there is a note recorded from a previous session. This action is performed through first reading the received command `“#NOTE_ADDED + step_ID”` (see Table 9). Immediately after the step_ID is read, a message is sent to the core asking for the corresponding note of this step using the command "`get_proc_notes? + procedure name + step_ID`" (see Table 10). By reading the answer (see Table 11), where information such as the step position in the procedure or the note name, type and path is given, the app adds the note to a specific ArrayList made of objects "notes". These objects are specifically created for this purpose. Of course a prior check is done before adding the note just in case the note has been previously added.

With the note added to the ArrayList the remaining part of this first stage is to show the icon in the WebView. To do so the app calls a method in the JavaScript file which modifies the HTML file to show the image by changing the style of this step. This process follows the method shown in Figure 59.

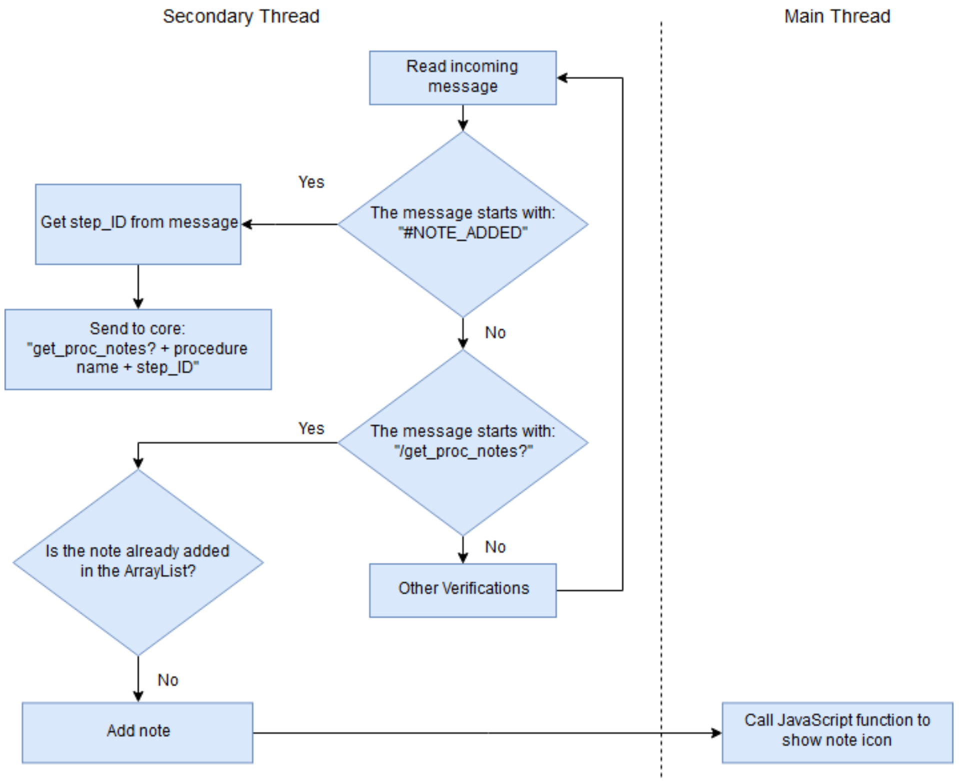

Secondary Thread
Main Thread
Read incoming message
The message starts with: "#NOTE_ADDED"
Yes
Get step_ID from message
Send to core: "get_proc_notes? + procedure name + step_ID"
No
The message starts with: "/get_proc_notes?"
Yes
Is the note already added in the ArrayList?
No
Other Verifications
No
Add note
Call JavaScript function to show note icon

Figure 68: Detailed flowchart that describes the first process that involves note taking feature.

That was all in relation of what the user cannot control about this process, the remaining information about the note taking feature involves the interaction of the user with the app and it includes the note creation; the note checking and the note erasing.

Regarding note creation, from the Dialog shown in Figure 67, the user can choose either text, image or video and audio note:

- Text note: After pressing this ImageButton, the app displays below the row with the three main image buttons a EditText box for writing and a save Button to send the text to the core. When the save Button is pressed the command “take_note TXT step_ID + text” is sent to the core and then the message “#NOTE_ADDED…” is received, so the process to add the note to the ArrayList and to show the icon is the same to the one in Figure 68. See Figure 69 for more details of this process.

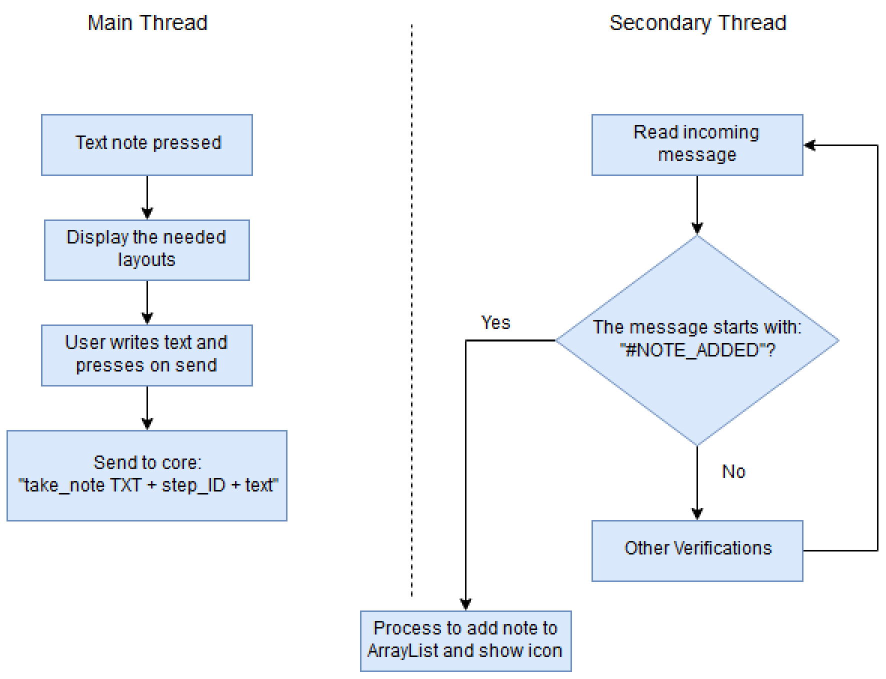

Main Thread
Secondary Thread
Text note pressed
Display the needed layouts
User writes text and presses on send
Send to core: "take_note TXT + step_ID + text"
Read incoming message
The message starts with: "#NOTE_ADDED"?
Yes
No
Other Verifications
Process to add note to ArrayList and show icon

Figure 69: Detailed flowchart that describes how a text note is taken.

- Video and photo notes: After pressing on the camera ImageButton, the app sends the command “preview start” to the core. As an on success message the core answers with “+PREVIEW START” and a listener starts in the app that starts an Activity called Camera. This activity does not implement a typical camera view, what this activity implements is a WebView pointing to the URL where the core is streaming the images gotten from the camera, since the multimedia part is managed by mobiPV software. Next to the WebView there are two buttons for starting the video recording and the photo taking.

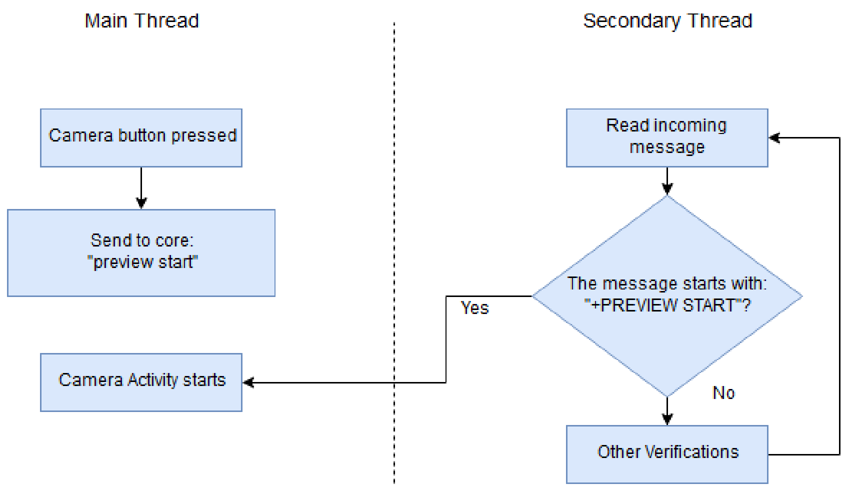


Figure 70: Detailed flowchart that describes how to start the video recording and photo taking.

If the user presses the video recording Button, the app sends a message to the core, asking it to start a video, the command is: "start_note video + step_ID". Then the image in the button changes into a stop button and if the user presses on it, the message "stop_note video + step_ID" is sent. The dialog is closed and the core sends the message "#NOTE_ADDED…" so the process in Figure 68 takes place again to add the note to the ArrayList and to show the note icon.

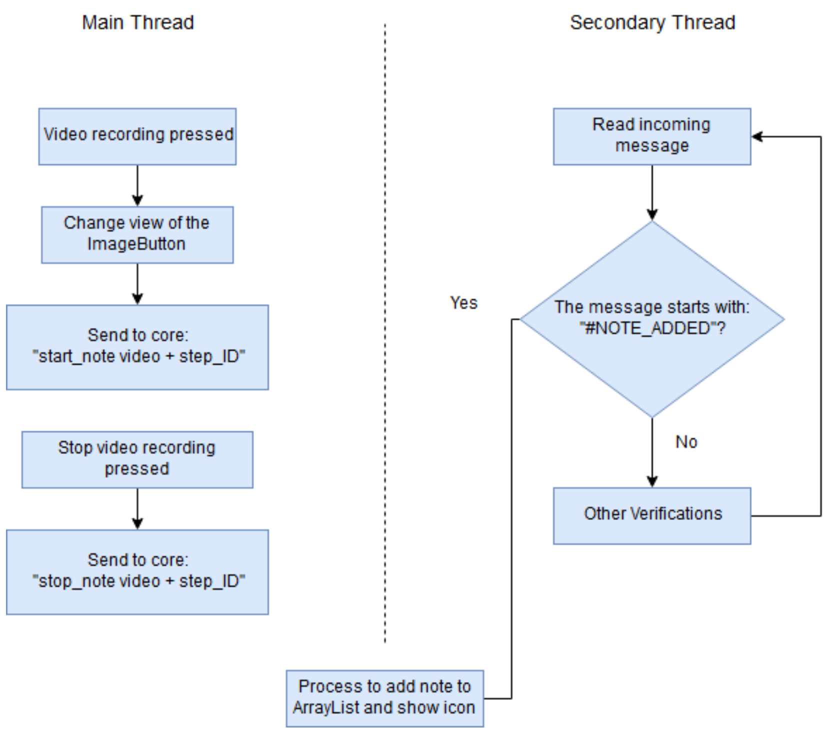

Main Thread
Secondary Thread
Video recording pressed
Change view of the ImageButton
Send to core: "start_note video + step_ID"
Stop video recording pressed
Send to core: "stop_note video + step_ID"
Read incoming message
The message starts with: "#NOTE_ADDED"?
Yes
No
Other Verifications
Process to add note to ArrayList and show icon

Figure 71: Detailed flowchart that describes how to start recording a video.

If, on the other hand, the user presses over the photo button, the core sends the message: "take_note photo + step_ID". Then the dialog is closed and the core answers with the message "#NOTE_ADDED" and the system adds the note into the ArrayList and shows the icon.

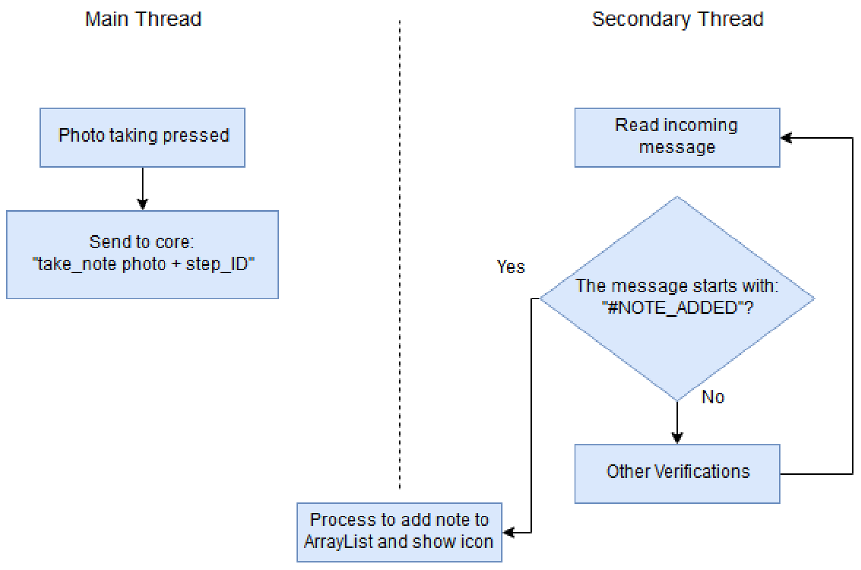

Main Thread
Secondary Thread
Photo taking pressed
Send to core: "take_note photo + step_ID"
Read incoming message
The message starts with: "#NOTE_ADDED"?
Yes
No
Other Verifications
Process to add note to ArrayList and show icon

Figure 72: Detailed flowchart that describes how to take a photo.

- Audio Note: After the user presses the last ImageButton the app sends the command: “start_note audio + step_ID” and similarly to the video note, the button changes the image into a stop one. When this last ImageButton is pressed the command: “stop_note audio + step_ID” is sent and the system stops recording. The message “#NOTE_ADDED” is received and the same process again.

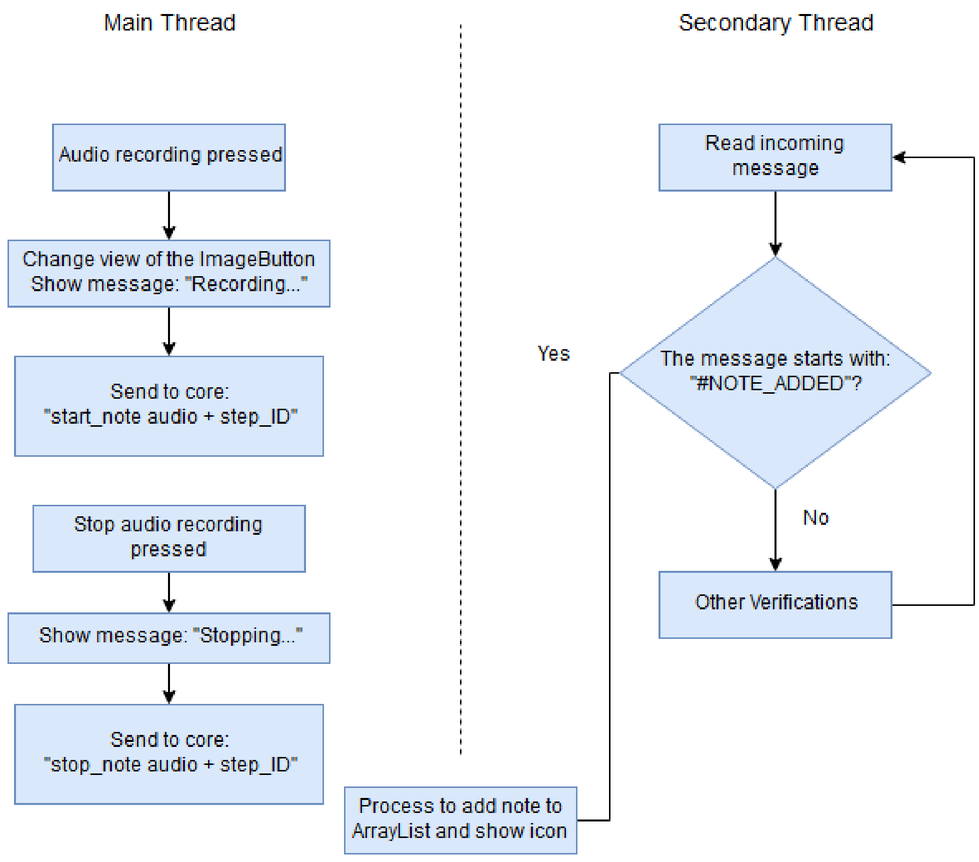

Main Thread
Secondary Thread
Audio recording pressed
Change view of the ImageButton Show message: "Recording..."
Send to core: "start_note audio + step_ID"
Stop audio recording pressed
Show message: "Stopping..."
Send to core: "stop_note audio + step_ID"
Read incoming message
The message starts with: "#NOTE_ADDED"?
Yes
No
Other Verifications
Process to add note to ArrayList and show icon

Figure 73: Detailed flowchart that describes how to record audio.

Changing into the note checking, this process happens every time the user presses a displayed note icon, when the usual dialog pops up, it includes more information according with the type of note that is included in that step.

- Text Note: The app shows a Text View, showing the text, and a trash can ImageButton.
- Video Note: The app shows the tittle of the note, a button to open it and the trash can
- Photo Note: The app shows the tittle of the note, a button to open it and the trash can. After pressing the button, the dialog displays the photo.
- Audio Note: The app shows the tittle of the note, a play button, a timeline, a timer and the trash can.

To finish, in the Dialog displayed to the user for checking every note there is one trash can that removes the note from the system by sending the command “del_note CURRENT + note file name”. As an on success answer the system answers with “+del_note + note file name” and the app removes the icon and the note from the notes ArrayList.

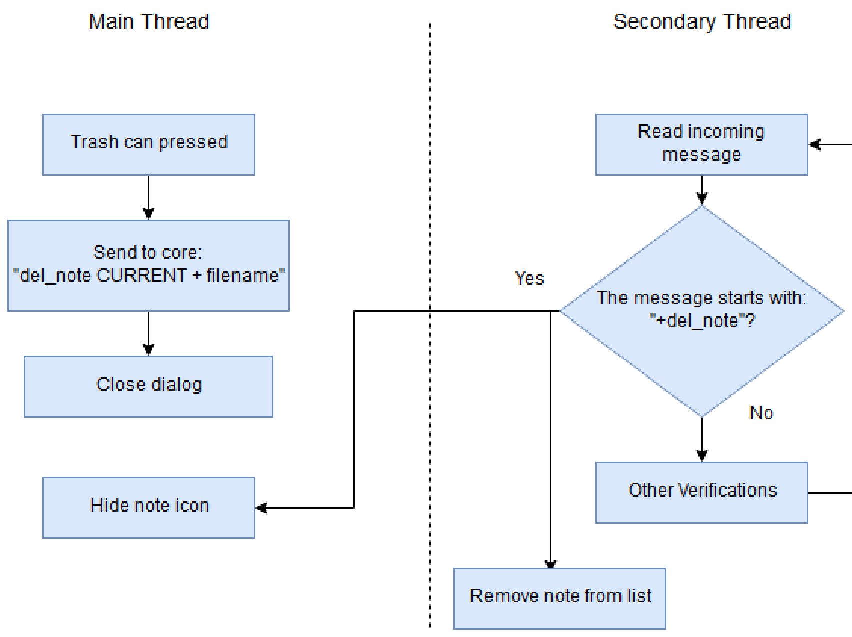


Figure 74: Detailed flowchart that describes how a note is removed.

Finally, the last feature to describe is the Collaboration Session, which is thought to be used with another mobiPV core. The app can work wither as a slave or as a master. When it is working as a slave the green marker will be shifted to the position of the green marker where the system designated as master is working, and the free navigation is disabled for the app, this means that the app is not anymore under the control of the position of the green marker. In this environment is when the use of the automatic scrolling disabling or enabling is more interesting because maybe the crew member wants to read for longer one step while ground team move forward along the procedure. When the system is designated as master, it has total control of the green marker and the slaves will be following it.

As a master, the app can establish the other core working as a slave with master role and if the app is working as slave can ask the master to change roles. All this interactions in the collaboration sessions are accessed with the only remaining button of Figure 55 that has not been yet discussed. It is located on the top bar of the screen just to the left of the three dots option menu.

Pressing the mentioned button, involves asking the core about the synchronization session state, by using the command "SYNC_SESSION_INFO?". When the core answers with the message that starts with "+SYNC_SESSION_INFO? …", the app launches the SyncSession Activity that implements a dialog. See Figure 75 for a detailed flowchart of this process.

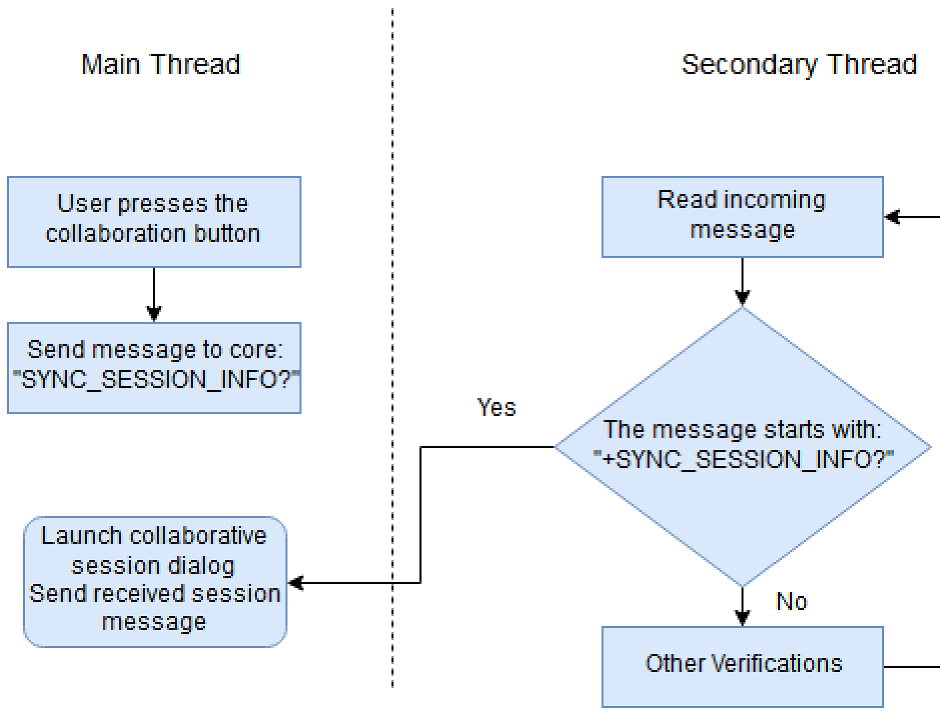

Main Thread
Secondary Thread
User presses the collaboration button
Send message to core: "SYNC_SESSION_INFO?"
Read incoming message
The message starts with: "+SYNC_SESSION_INFO?"
Yes
No
Launch collaborative session dialog Send received session message
Other Verifications

Figure 75: Detailed flowchart of the logic when pressing the collaborative session button

The dialog pops up and processes the message that was transferred by the other activity, this message contains information about the state of the collaboration session and according to it, the dialog is built. So, if there is not any stablished session the dialog will show a start session button which sends the command "START_SYNC_SESSION OK DEFAULT" command to the core. If the message contains information about a previous stablished session by other core, the dialog displays a join session button. This button sends the command "JOIN_SYNC_SESSION" to the core.

The button that opens the dialog can be also pressed while the app is involved in a collaboration session, then there are other possibilities to build the dialog after reading the message:

- The app is the master of the session, but nobody joined yet. Then a leaving session is added, so the session is finished by pressing it and sending the command "LEAVE_SYNC_SESSION OK" and the dialog is closed. As nobody is on the session the message "Nobody joined yet" is printed.
- The app is the master and there is someone that joined. Then the same leaving session button from before is added that finishes the session. And as there is one or more other users, the option of transferring the master role to them is also added with a button, the command that it implements is "set_master DEFAULT + username"
- The app is not the master. Then the leaving button is added to leave the session as before. On the other hand, one button for asking for the master role to the current master is added.

In ProcViewer screen, depending of the current state of the session the collaboration button is modified:

- The button is white if there is no session.
- The button is green with “MASTER” text added, when the app is set as master on the session.
- The button is green with “SLAVE” text added, when the app is set a slave on a session.

Until now the main components of the app and how they internally work have been explained, it is time to move into the execution of the application a test and to have a look at the results of it.

4 Testing and execution of mobiPV app

Along this section the results of performing a test on the mobiPV app are found in the next lines in the shape of screenshots from the app screen and from the debug tool of Android Studio to follow the message exchange between the app and the core.

The full text of the test can be found on Appendix 1, since in this section it is divided point by point with the solutions below each one of the parts of the test.

The material used for the test has been:

- Laptop with Linux virtual machine running one of the two cores of mobiPV.
- Nexus 5 running the other core of mobiPV and the app developed in this project.
- A WLAN generated by the laptop where both the virtual machine and the smartphone have to be connected and reachable.
- A camera connected to the Nexus 5.

Once everything is prepared the test can start:

1. **Open mobiPV app on the Desktop denoted by the icon**

The icon as shown in Figure 76 can be located on the Desktop or in the Main Menu.

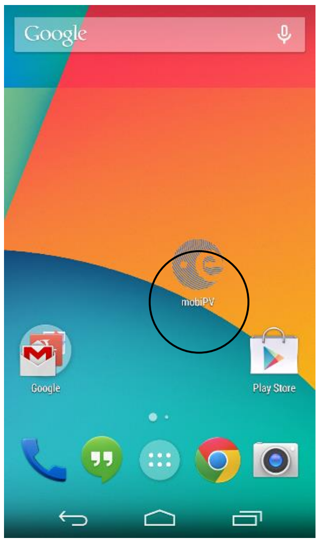
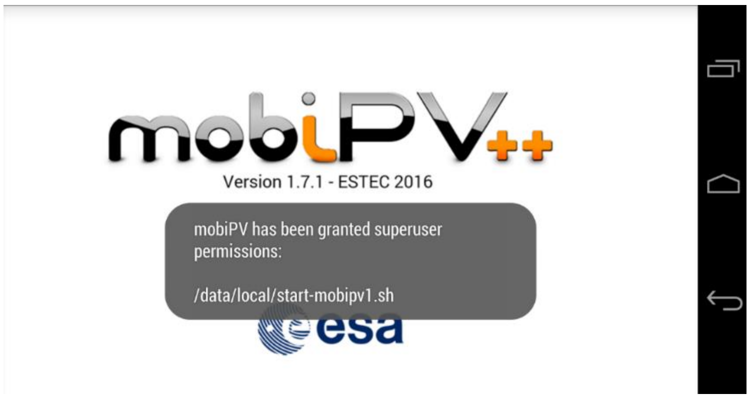


Figure 76: mobiPV location on Desktop

2. **SplashScreen is shown for some seconds. Check that some Super User permissions are given to the app. Then the screen changes into Login Screen.**

Figure 77 shows the screen while a Toast [30] (the pop up message) informs that start-mobipv1.sh has been granted with superuser permissions, that is what happens when the first of the commands in "Start mobiPV in core" process from Figure 35 is executed. Figure 77 also shows both mobiPV logo and ESA logo


mobiPV++
Version 1.7.1 - ESTEC 2016

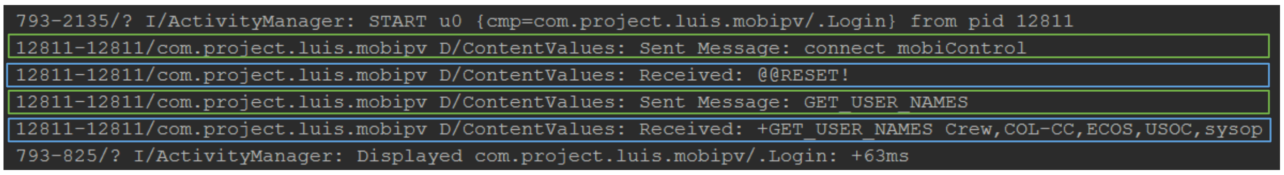
mobiPV has been granted superuser permissions:

/data/local/start-mobipv1.sh

esa

Figure 77: Starting of mobiPV Android GUI. This screenshot corresponds to the SplashScreen Activity when the first command (start-mobiPV1.sh) is executed.

Once Login screen is loaded the first exchange of messages between the app and the core starts. As all the messages exchange, the command "connect mobicontrol" is sent to a socket pointing to localhost and the mobicontrol port 7777. The confirmation "@@RESET!" messages arrives later and then the app asks for the users registered in the system with the command “GET_USER_NAMES”. Finally, the last command from the core comes and the spinner in the app is filled with this information. See Figure 79, where the app messages are highlighted in green and the core messages are highlighted in blue.



```
793-2135/? I/ActivityManager: START u0 {cmp=com.project.luis.mobipv/.Login} from pid 12811
12811-12811/com.project.luis.mobipv D/ContentValues: Sent Message: connect mobiControl
12811-12811/com.project.luis.mobipv D/ContentValues: Received: @@RESET!
12811-12811/com.project.luis.mobipv D/ContentValues: Sent Message: GET_USER_NAMES
12811-12811/com.project.luis.mobipv D/ContentValues: Received: +GET_USER_NAMES Crew,COL-CC,ECOS,USOC,sysop
793-825/? I/ActivityManager: Displayed com.project.luis.mobipv/.Login: +63ms
```

Figure 78: Message exchange in the app when the Login screen is loaded

3. **Press on Login with Crew user selected, the screen will change into MainMenu Screen**

The default user is Crew, so it is the selected one when the screen is loaded as shown in Figure 79

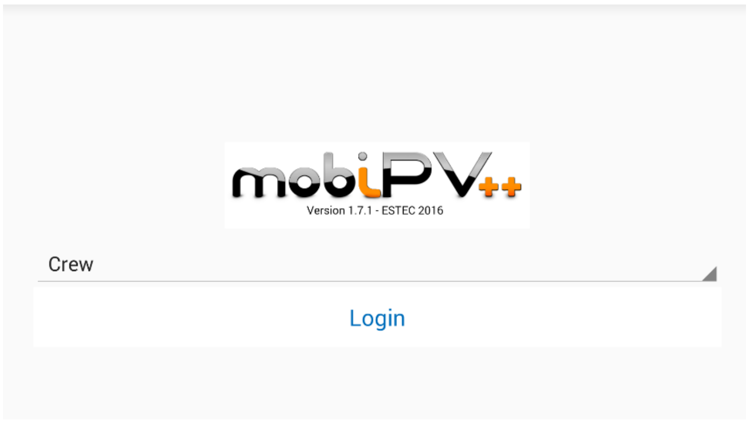


Figure 79: Login Screen

4. **Press on the three dots on the top right corner of the Action Menu denoted by ⋮ on white to open the Side Menu.**

The displayed Side Menu is shown in Figure 80 and the available options are TOC, Comms, Settings, Diagnostics and Log out.

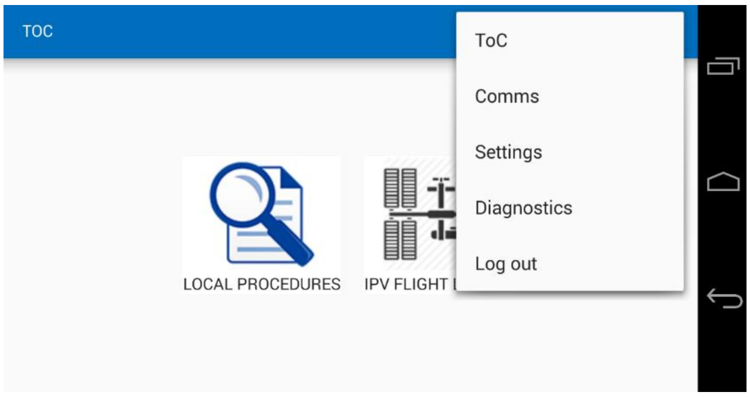
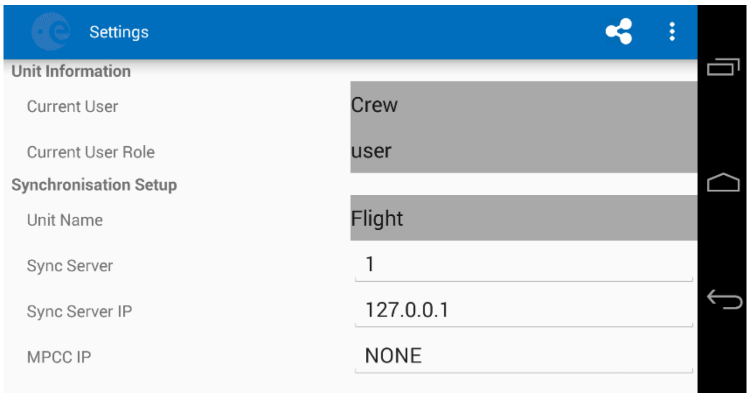


Figure 80: Displayed Side Menu

5. **Press on Settings option, Setting screen will be opened.**

Figure 81 shows the Settings window of the selected user, Crew. It can be verified by reading the grey box to the right of the field "Current User"


Settings

Unit Information

Current User	Crew
Current User Role	user

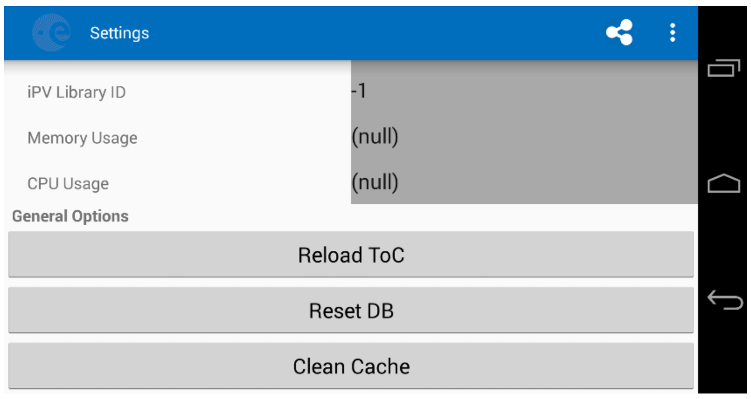
Synchronisation Setup

Unit Name	Flight
Sync Server	1
Sync Server IP	127.0.0.1
MPCC IP	NONE

Figure 81: Settings window of user Crew

6. **Check that all the options and the three button on the bottom are added. Check as well that, for example, the option "Video camera device" is blocked and cannot be changed.**

Inspecting now the screen and scrolling to the bottom of if it is easy to check that all the fields of the left part of Figure 45 are there, as well as the three buttons shown again in Figure 82

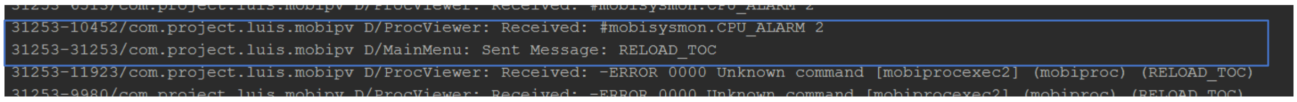


```
-1
(null)
(null)
```

Figure 82: Three buttons on Settings view for Crew user

To check whether the buttons work or not, the debugger of Android Studio is used:

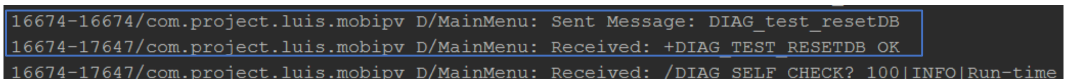
- "Reload TOC" is pressed: The message "RELOAD_TOC" is sent and as an answer the message "-ERROR …" is received due to the Nexus 5 is connected to the WLAN of the laptop and not to the Skytek site where the procedures are stored (see Figure 83).



```
31253-6513/com.project.luis.mobipv D/ProcViewer: Received: #mobisysmon.CPU_ALARM 2
31253-10452/com.project.luis.mobipv D/ProcViewer: Received: #mobisysmon.CPU_ALARM 2
31253-31253/com.project.luis.mobipv D/MainMenu: Sent Message: RELOAD_TOC
31253-11923/com.project.luis.mobipv D/ProcViewer: Received: -ERROR 0000 Unknown command [mobiprocexec2] (mobiproc) (RELOAD_TOC)
31253-9980/com.project.luis.mobipv D/ProcViewer: Received: -ERROR 0000 Unknown command [mobiprocexec2] (mobiproc) (RELOAD_TOC)
```

Figure 83: Message exchange between app and core when "Reload TOC" button is pressed.

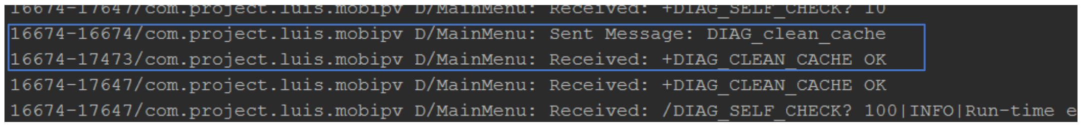
- "Reset DB" is pressed: The message "DIAG_test_resetDB" is sent and the confirmation message "+DIAG_TEST_RESETDB OK" is received (see Figure 84).



```
16674-16674/com.project.luis.mobipv D/MainMenu: Sent Message: DIAG_test_resetDB
16674-17647/com.project.luis.mobipv D/MainMenu: Received: +DIAG_TEST_RESETDB OK
16674-17647/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 100|INFO|Run-time
```

Figure 84: Message exchange between app and core when "Reset DB" button is pressed.

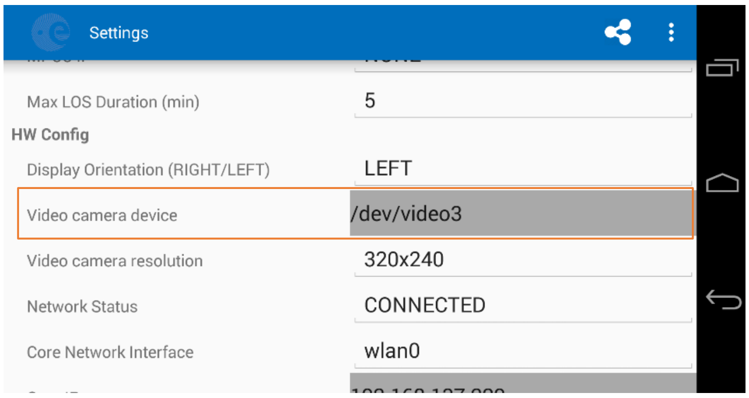
- "Clean Cache" is pressed: The message "DIAG_clean_cache" is sent and the confirmation message "+DIAG_CLEAN_CACHE OK" is received (see Figure 85).



```
16674-17647/com.project.luis.mobipv D/MainMenu: Received: +DIAG_SELF_CHECK? 10
16674-16674/com.project.luis.mobipv D/MainMenu: Sent Message: DIAG_clean_cache
16674-17473/com.project.luis.mobipv D/MainMenu: Received: +DIAG_CLEAN_CACHE OK
16674-17647/com.project.luis.mobipv D/MainMenu: Received: +DIAG_CLEAN_CACHE OK
16674-17647/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 100|INFO|Run-time e
```

Figure 85: Message exchange between app and core when "Clean Cache" button is pressed.

In relation with the blocked value. As the current user is “Crew” the whole values to configure are not available, for example, as the title mentions, the option “Video camera device” is not available as can be checked in Figure 86.



Max LOS Duration (min) 5

HW Config

Display Orientation (RIGHT/LEFT) LEFT

Video camera device /dev/video3

Video camera resolution 320x240

Network Status CONNECTED

Core Network Interface wlan0

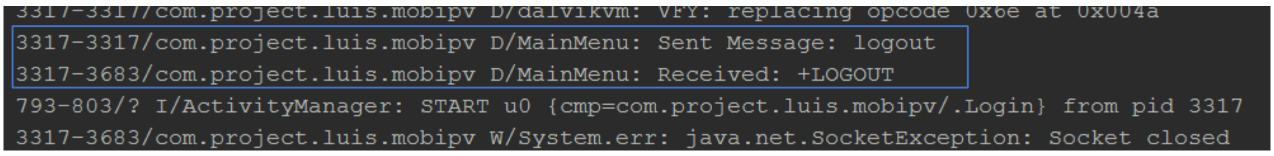
Figure 86: Setting screen for user Crew, where "Video camera device" is highlighted and blocked

7. **Press Return Android button to come back to the Main Menu Screen. Denoted by ⮌ .**

In Figure 86, the button on the black bar on the right bottom is pressed and again the Main Menu is shown.

8. **Press again on the Side Menu button and press now on Logout. The Login Screen will be loaded.**

The last option of the Side Menu, see Figure 80, is pressed and the app goes back to Login screen. The message “logout” is sent to the core and as soon as the confirmation answer arrives, the app moves back to the Login screen.



```
3317-3317/com.project.luis.mobipv D/dalvikvm: VFY: replacing opcode 0x6e at 0x004a
3317-3317/com.project.luis.mobipv D/MainMenu: Sent Message: logout
3317-3683/com.project.luis.mobipv D/MainMenu: Received: +LOGOUT
793-803/? I/ActivityManager: START u0 {cmp=com.project.luis.mobipv/.Login} from pid 3317
3317-3683/com.project.luis.mobipv W/System.err: java.net.SocketException: Socket closed
```

Figure 87: Message exchange when the app does logout

9. In the user selector, choose special user "sysop", press on "Login", check that a Dialog appears asking for the password.

To change from the default user "Crew" to a different one, it is necessary to press on the user selector and scroll it until user "sysop" appears. All these users coincide with the content of the message shown in Figure 78. See Figure 88 where the list of users is displayed.

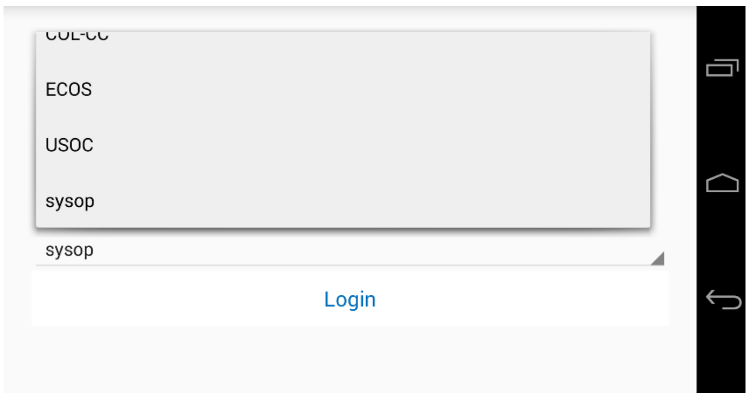


Figure 88: User "sysop" selection

When the button Login is pressed, the app detects that the special user has been selected and displays the message shown in Figure 89

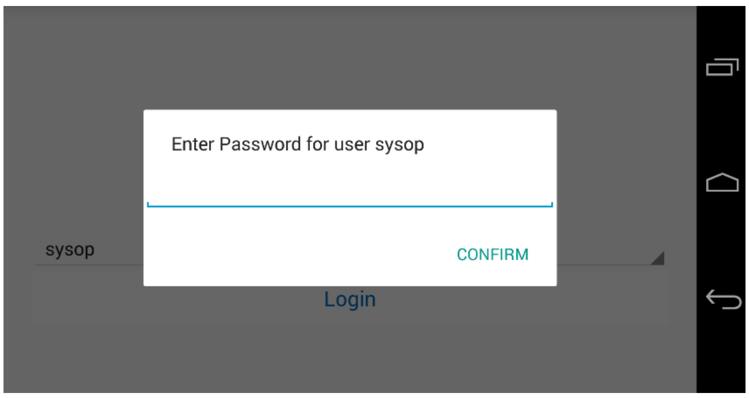

Enter Password for user sysop

Figure 89: Login Screen. If the selected user is sysop, the app is going to ask for a password.

10. Write "superuser" and press on "Confirm". Check that a message appears with the text "Wrong password, try again"

See Figure 90 where the message has popped up.

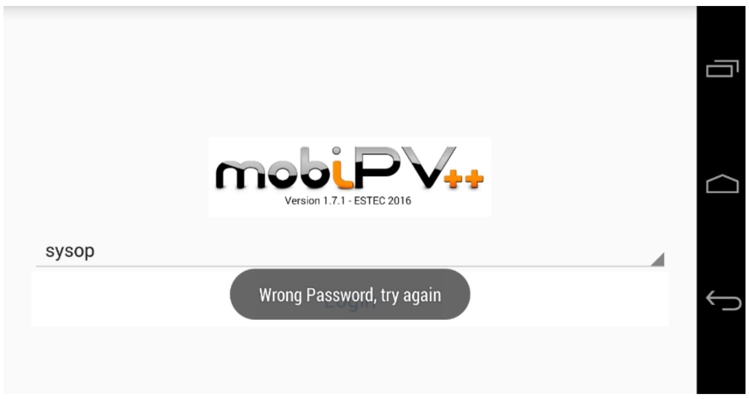
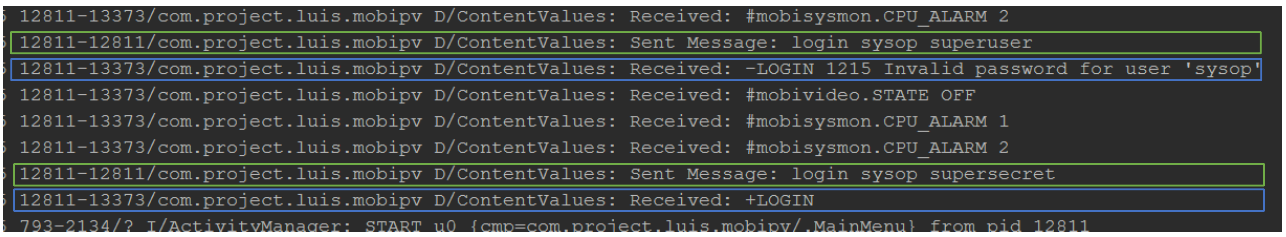


Figure 90: A message saying "Wrong Password, try again" pops up if the answer from the core starts with "-LOGIN"

11. **Press again on Login and write "supersecret", press on "Confirm" and check that the MainMenu screen is loaded.**

This time the password "supersecret" is the correct one and the system will do login. In Figure 91 a screenshot of the message exchange is shown. This message exchange belongs to the last two point where first the user tries to submit a wrong password (superuser) and then a correct one (supersecret). In the messages of the figure, the first sent message is "login sysop superuser", where the second word is the selected user and the last word is the password and the answer is "-LOGIN …" which is not the on success answer due to the password is wrong. The second message is "login sysop supersecret" and since it is the correct one the core answer with "+LOGIN".

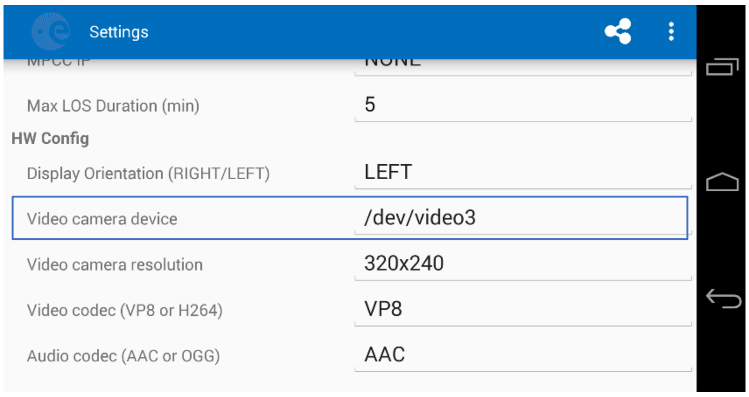


```
12811-13373/com.project.luis.mobipv D/ContentValues: Received: #mobisysmon.CPU_ALARM 2
12811-12811/com.project.luis.mobipv D/ContentValues: Sent Message: login sysop superuser
12811-13373/com.project.luis.mobipv D/ContentValues: Received: -LOGIN 1215 Invalid password for user 'sysop'
12811-13373/com.project.luis.mobipv D/ContentValues: Received: #mobivideo.STATE OFF
12811-13373/com.project.luis.mobipv D/ContentValues: Received: #mobisysmon.CPU_ALARM 1
12811-13373/com.project.luis.mobipv D/ContentValues: Received: #mobisysmon.CPU_ALARM 2
12811-12811/com.project.luis.mobipv D/ContentValues: Sent Message: login sysop supersecret
12811-13373/com.project.luis.mobipv D/ContentValues: Received: +LOGIN
793-2134/? I/ActivityManager: START u0 {cmp=com.project.luis.mobipv/.MainMenu} from pid 12811
```

Figure 91: Message exchange between the app and the core when the login for user "sysop". The green highlighted messages correspond to the app and the blue highlighted ones to the core.

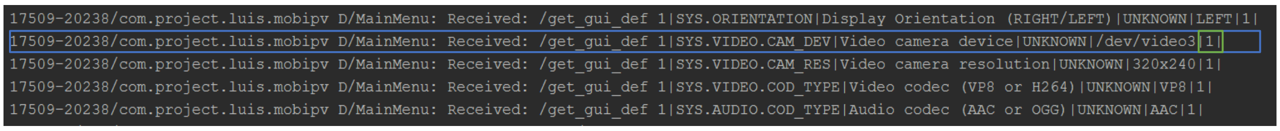
12. **Open the Side Menu and select again Settings, check now that the option "Video camera device" is available to be edited.**

Figure 92 shows how the field is now edditable. When checking the messages coming from the core that help the app to build this screen it can be also checked that the “/get_gui_def …” that corresponds to the “Video camera device” field, in its last parameter has a “1”. This means that the field is editable (See Figure 93).



MPCC IP	NONE
Max LOS Duration (min)	5
HW Config	
Display Orientation (RIGHT/LEFT)	LEFT
Video camera device	/dev/video3
Video camera resolution	320x240
Video codec (VP8 or H264)	VP8
Audio codec (AAC or OGG)	AAC

Figure 92: Setting screen for user "sysop"

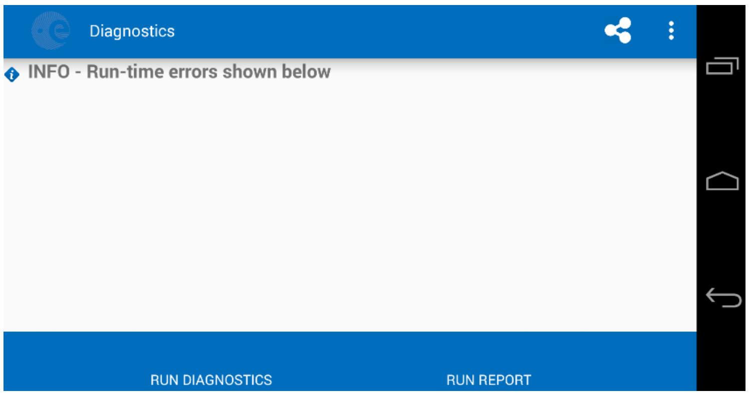


```
17509-20238/com.project.luis.mobipv D/MainMenu: Received: /get_gui_def 1|SYS.ORIENTATION|Display Orientation (RIGHT/LEFT)|UNKNOWN|LEFT|1|
17509-20238/com.project.luis.mobipv D/MainMenu: Received: /get_gui_def 1|SYS.VIDEO.CAM_DEV|Video camera device|UNKNOWN|/dev/video3|1|
17509-20238/com.project.luis.mobipv D/MainMenu: Received: /get_gui_def 1|SYS.VIDEO.CAM_RES|Video camera resolution|UNKNOWN|320x240|1|
17509-20238/com.project.luis.mobipv D/MainMenu: Received: /get_gui_def 1|SYS.VIDEO.COD_TYPE|Video codec (VP8 or H264)|UNKNOWN|VP8|1|
17509-20238/com.project.luis.mobipv D/MainMenu: Received: /get_gui_def 1|SYS.AUDIO.COD_TYPE|Audio codec (AAC or OGG)|UNKNOWN|AAC|1|
```

Figure 93: Part of "/get_gui_def" battery of messages. The highlighted one corresponds to the "Video camera device" field

13. Go back to Main Menu and open the Side Menu, press on “Diagnostics”. Diagnostics Screen will be opened.

At the very beginning the screen is empty of content, the two button on the bottom bar fill it with content after pressing them (see Figure 94).

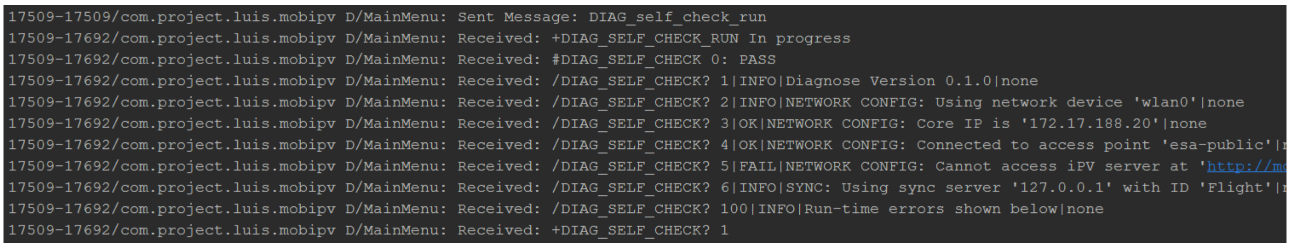


INFO - Run-time errors shown below

Figure 94: Diagnostics screen

14. Run a Diagnostics by pressing the button on the left bottom and check that some results appear and there are three kind of messages: INFO, denoted by , OK, denoted by , and FAIL, denoted by

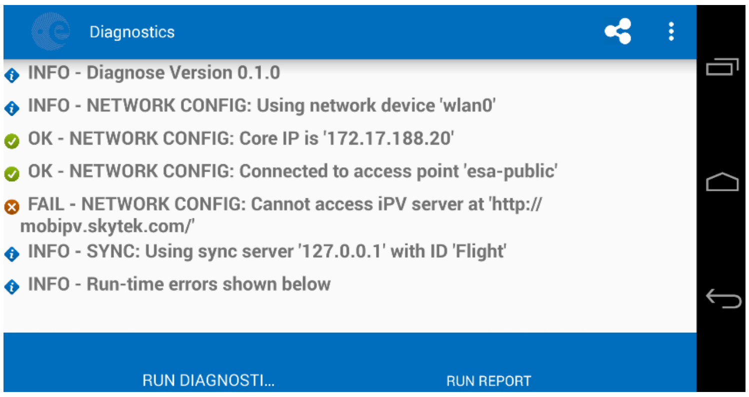
After pressing the bottom button on the left, the message "DIAG_self_check_run" asking for running a diagnostic is sent to the core and the core answers with several messages informing that the checking has started, informing whether the test was passed or not and informing about the different checked features and their states. Check Figure 95 to see the whole message exchange.



```
17509-17509/com.project.luis.mobipv D/MainMenu: Sent Message: DIAG_self_check_run
17509-17692/com.project.luis.mobipv D/MainMenu: Received: +DIAG_SELF_CHECK_RUN In progress
17509-17692/com.project.luis.mobipv D/MainMenu: Received: #DIAG_SELF_CHECK 0: PASS
17509-17692/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 1|INFO|Diagnose Version 0.1.0|none
17509-17692/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 2|INFO|NETWORK CONFIG: Using network device 'wlan0'|none
17509-17692/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 3|OK|NETWORK CONFIG: Core IP is '172.17.188.20'|none
17509-17692/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 4|OK|NETWORK CONFIG: Connected to access point 'esa-public'|
17509-17692/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 5|FAIL|NETWORK CONFIG: Cannot access iPV server at 'http://
17509-17692/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 6|INFO|SYNC: Using sync server '127.0.0.1' with ID 'Flight'|
17509-17692/com.project.luis.mobipv D/MainMenu: Received: /DIAG_SELF_CHECK? 100|INFO|Run-time errors shown below|none
17509-17692/com.project.luis.mobipv D/MainMenu: Received: +DIAG_SELF_CHECK? 1
```

Figure 95: Message exchange between core and app when a diagnostic is run

After the system informs that the test has been passed in Figure 95, there are three kind of "/DIAG_SELF_CHECK", the ones with the state "INFO", with state "OK" and the state "FAIL". Every one of them is represented on the screen with one icon as can be checked in Figure 96



```
INFO - Diagnose Version 0.1.0
INFO - NETWORK CONFIG: Using network device 'wlan0'
OK - NETWORK CONFIG: Core IP is '172.17.188.20'
OK - NETWORK CONFIG: Connected to access point 'esa-public'
FAIL - NETWORK CONFIG: Cannot access iPV server at 'http://
mobipv.skytek.com/'
INFO - SYNC: Using sync server '127.0.0.1' with ID 'Flight'
INFO - Run-time errors shown below
```

Figure 96: Diagnostics screen filled with the diagnostic results.

15. **Go back to Main Menu Screen and press on "Local Procedures", the screen LocalTOCList will be loaded. Check that there are three procedures and press on "SKIN B NOMINAL OPERATIONS". The screen ProcViewer will be opened.**

On LocalTOCList screen, there are three procedures (see Figure 97):

- RASPBERRY PI ASSEMBLY
- SKIN B NOMINAL OPERATIONS
- MobiPV++ Checkout

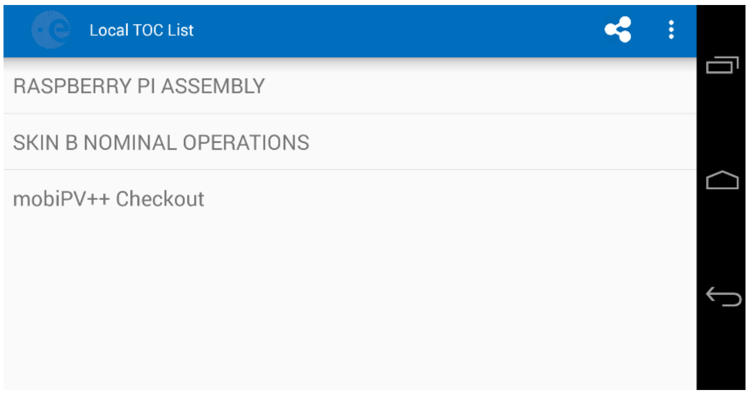
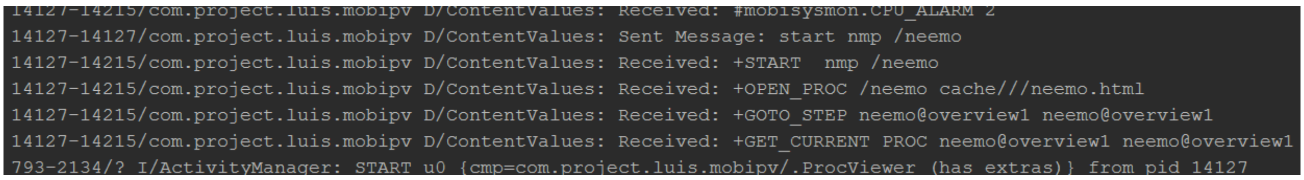


Figure 97: LocalTOCList screen

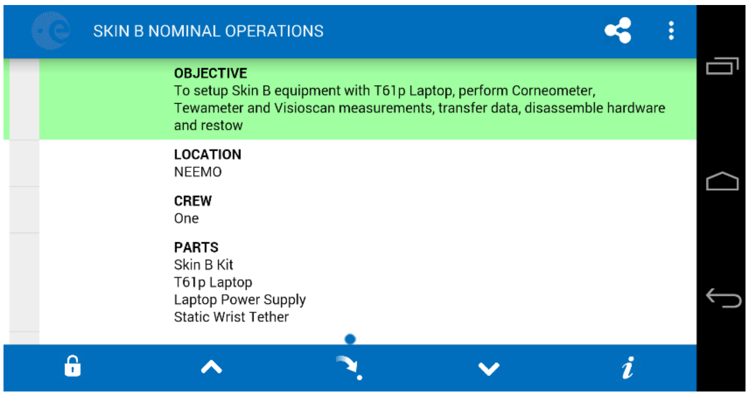
When the procedure is opened by pressing on it, there is an exchange of messages between the app and the core that stars after the app sends "start nmp /neemo". The core then sends the confirmation and some information about the procedure as the actual working step (See Figure 98)



```
14127-14215/com.project.luis.mobipv D/ContentValues: Received: #mobisysmon.CPU_ALARM 2
14127-14127/com.project.luis.mobipv D/ContentValues: Sent Message: start nmp /neemo
14127-14215/com.project.luis.mobipv D/ContentValues: Received: +START  nmp /neemo
14127-14215/com.project.luis.mobipv D/ContentValues: Received: +OPEN_PROC /neemo cache///neemo.html
14127-14215/com.project.luis.mobipv D/ContentValues: Received: +GOTO_STEP neemo@overview1 neemo@overview1
14127-14215/com.project.luis.mobipv D/ContentValues: Received: +GET_CURRENT PROC neemo@overview1 neemo@overview1
793-2134/? I/ActivityManager: START u0 {cmp=com.project.luis.mobipv/.ProcViewer (has extras)} from pid 14127
```

Figure 98: Message exchange between the app and the core when a procedure is opened

The new screen shows the procedure and the green marker located at the beginning of it (see Figure 99)



SKIN B NOMINAL OPERATIONS

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

LOCATION
NEEMO

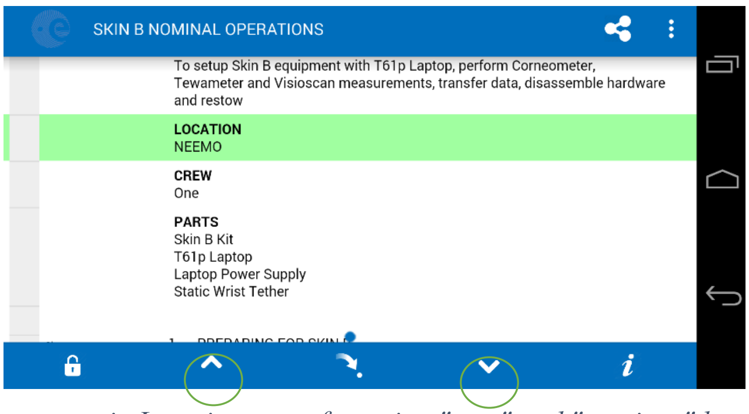
CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Figure 99: ProcViewer screen

16. Check that the green marker is located at the beginning of the procedure, press on next button, denoted by , on the bottom menu twice and check that the green marker moves forward two positions. Press now on the previous button, denoted by , located in the same menu once and check that the green marker goes backwards one position. Check that at the end of this step the green marker is on the second position.

Figure 100 shows the new location of the step marker, over the step “Location”, after pressing on the two highlighted buttons “next” (on the right) and “previous” (on the left).


SKIN B NOMINAL OPERATIONS

To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Figure 100: ProcViewer screen in Location step after using "next" and "previous" buttons, highlighted with a green circle

The exchange of message between the core and the app in this case is very simple. On one hand the first time the user presses over the next button, the command "next" is sent and the cores answers with "+NEXT neemo@location neemo@overview1". The command is compound by the confirmation of the command ("+NEXT") and the step where the marker is moving ("neemo@location") and the step where the marker is at the beginning of the operation ("neemo@overview1"). When the button next is pressed a second time the marker goes to "Crew" step.

On the other hand, pressing the previous button means that the app sends the command "previous" and the core answers "+PREVIOUS neemo@crew neemo@location". This command has a meaning similar to the one above.

Figure 101 shows the message exchange for this navigation with the messages described before.

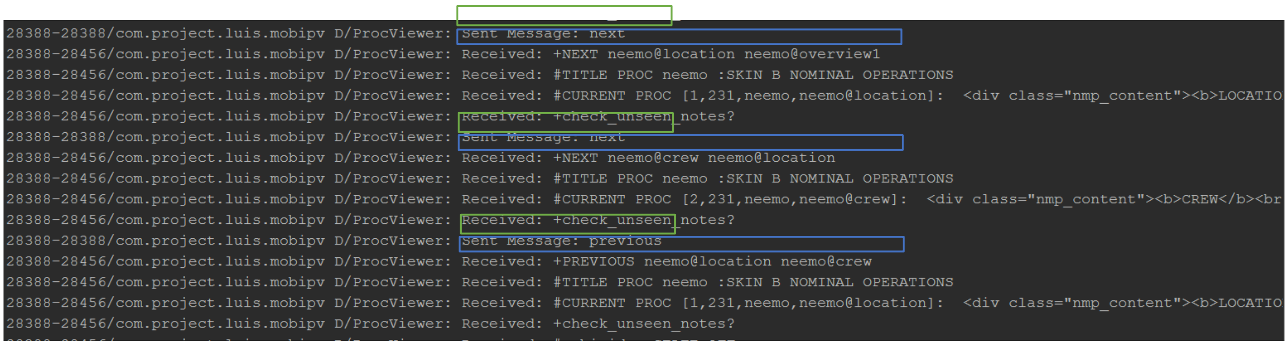
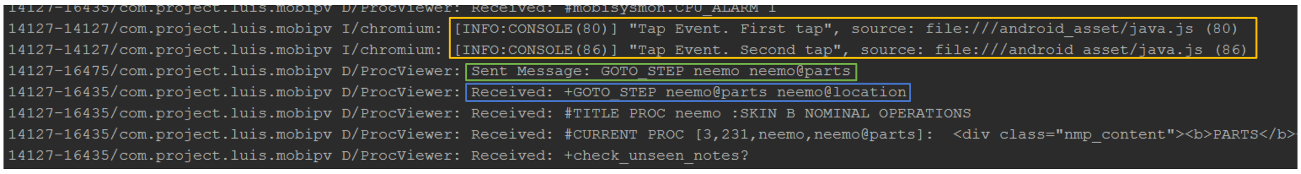

```
28388-28388/com.project.luis.mobipv D/ProcViewer: Sent Message: next
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: +NEXT neemo@location neemo@overview1
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: #TITLE PROC neemo :SKIN B NOMINAL OPERATIONS
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: #CURRENT PROC [1,231,neemo,neemo@location]:  <div class="nmp_content"><b>LOCATIO
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: +check_unseen_notes?
28388-28388/com.project.luis.mobipv D/ProcViewer: Sent Message: next
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: +NEXT neemo@crew neemo@location
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: #TITLE PROC neemo :SKIN B NOMINAL OPERATIONS
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: #CURRENT PROC [2,231,neemo,neemo@crew]:  <div class="nmp_content"><b>CREW</b><br
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: +check_unseen_notes?
28388-28388/com.project.luis.mobipv D/ProcViewer: Sent Message: previous
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: +PREVIOUS neemo@location neemo@crew
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: #TITLE PROC neemo :SKIN B NOMINAL OPERATIONS
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: #CURRENT PROC [1,231,neemo,neemo@location]:  <div class="nmp_content"><b>LOCATIO
28388-28456/com.project.luis.mobipv D/ProcViewer: Received: +check_unseen_notes?
```

Figure 101: Message exchange between the app and the core after doing next twice and then previous. The app messages are highlighted in green and the core messages are highlighted in blue. The rest of the messages are sent also by the core to give additional information

17. Double press on the fourth step, titled "PARTS" and check that the green marker jumps to this step.

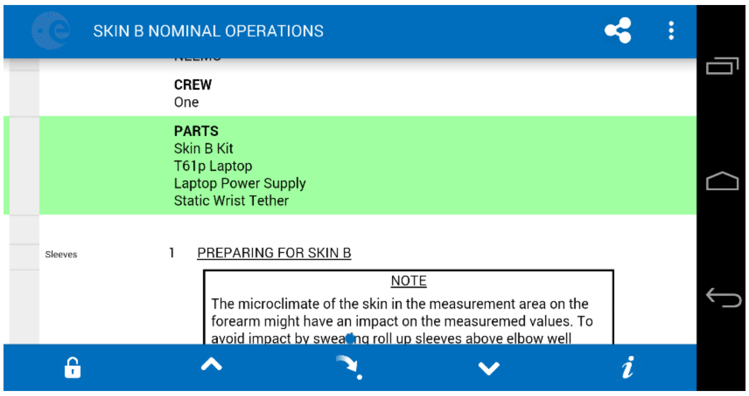
The messages when the double tap is done are shown in Figure 102. First the JavaScript file prints one message per tap (yellow highlighted messages). It is easy to identify this file because its path is printed. Following, the app sends the message “GOTO_STEP neemo neemo@parts” (green highlighted message) to the core, indicating the procedure name (“neemo”) and the step to jump (“neemo@parts”). Finally, the core confirms the movement of the green marker with the message “+GOTO_STEP neemo@parts neemo@location” (message highlighted in blue).



```
14127-16435/com.project.luis.mobipv D/ProcViewer: Received: #mobisysmon.CPU_ALARM 1
14127-14127/com.project.luis.mobipv I/chromium: [INFO:CONSOLE(80)] "Tap Event. First tap", source: file:///android_asset/java.js (80)
14127-14127/com.project.luis.mobipv I/chromium: [INFO:CONSOLE(86)] "Tap Event. Second tap", source: file:///android_asset/java.js (86)
14127-16475/com.project.luis.mobipv D/ProcViewer: Sent Message: GOTO_STEP neemo neemo@parts
14127-16435/com.project.luis.mobipv D/ProcViewer: Received: +GOTO_STEP neemo@parts neemo@location
14127-16435/com.project.luis.mobipv D/ProcViewer: Received: #TITLE PROC neemo :SKIN B NOMINAL OPERATIONS
14127-16435/com.project.luis.mobipv D/ProcViewer: Received: #CURRENT PROC [3,231,neemo,neemo@parts]:  <div class="nmp_content"><b>PARTS</b>
14127-16435/com.project.luis.mobipv D/ProcViewer: Received: +check_unseen_notes?
```

Figure 102: Message exchange between the app and the core when the double tap takes place and the step marker jumps to a new location. The JavaScript messages are highlighted in yellow, the app messages are highlighted in green and the core messages are highlighted in blue. The rest of the messages are sent also by the core to give additional information

The new position of the maker is “PARTS” (see Figure 103)



SKIN B NOMINAL OPERATIONS

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

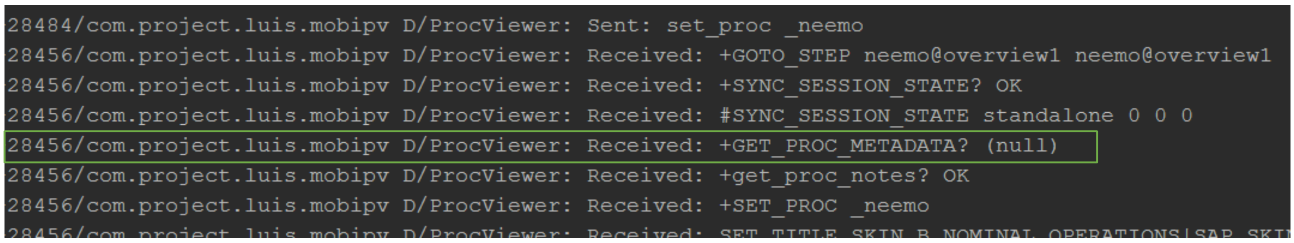
Sleeves   1   PREPARING FOR SKIN B

NOTE
The microclimate of the skin in the measurement area on the forearm might have an impact on the measuremed values. To avoid impact by sweating roll up sleeves above elbow well

Figure 103: ProcViewer screen showing the new step "PARTS" after the double tap

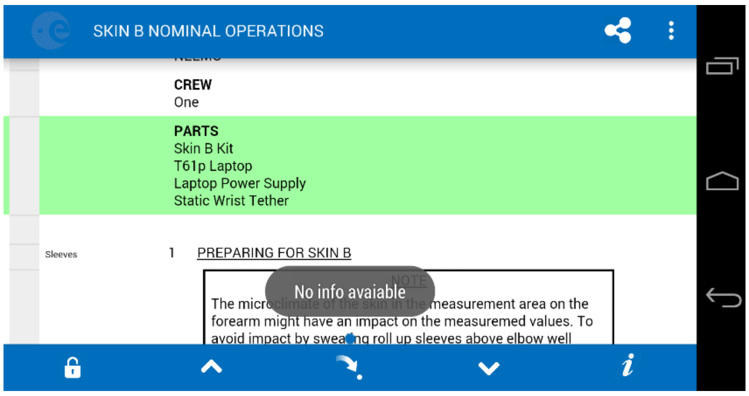
18. Press on info icon, denoted by *i*, although this step does not have any additional information, check that the message “No info available” is shown.

At the beginning, when the ProcViewer screen is loaded there is an exchange of messages shown in Figure 104, one of the messages is “SET_PROC_METADATA? (null)” (in green). This is the message that defines what is shown after pressing the button info and for this procedure, the message does not contain any additional information, so the message “No info available” is shown to the user as shows Figure 105



```
28484/com.project.luis.mobipv D/ProcViewer: Sent: set_proc _neemo
28456/com.project.luis.mobipv D/ProcViewer: Received: +GOTO_STEP neemo@overview1 neemo@overview1
28456/com.project.luis.mobipv D/ProcViewer: Received: +SYNC_SESSION_STATE? OK
28456/com.project.luis.mobipv D/ProcViewer: Received: #SYNC_SESSION_STATE standalone 0 0 0
28456/com.project.luis.mobipv D/ProcViewer: Received: +GET_PROC_METADATA? (null)
28456/com.project.luis.mobipv D/ProcViewer: Received: +get_proc_notes? OK
28456/com.project.luis.mobipv D/ProcViewer: Received: +SET_PROC _neemo
28456/com.project.luis.mobipv D/ProcViewer: Received: SET_TITLE SKIN B NOMINAL OPERATIONS|SAP SKI
```

Figure 104: Message exchange at the beginning of ProcViewer activity where the value of info is defined



SKIN B NOMINAL OPERATIONS

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Sleeves 1 PREPARING FOR SKIN B

NOTE
The microclimate of the skin in the measurement area on the forearm might have an impact on the measuremed values. To avoid impact by sweating roll up sleeves above elbow well

No info available

Figure 105: ProcViewer screen with message "No info available" after pressing info button

19. Press on the centre icon, denoted by and check that a Dialog appears with a list of steps. Look for step “3 Laptop Activation” and press on it. Check that the green marker jumps to this step.

The list of steps is obtained when the procedure is loaded by parsing the HTML file and storing the necessary information in an internal file and variable. After pressing the button on the centre of the bottom bar, the dialog shown in Figure 106 appears.

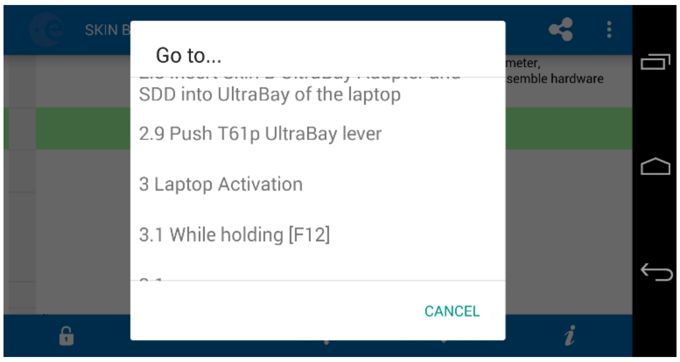
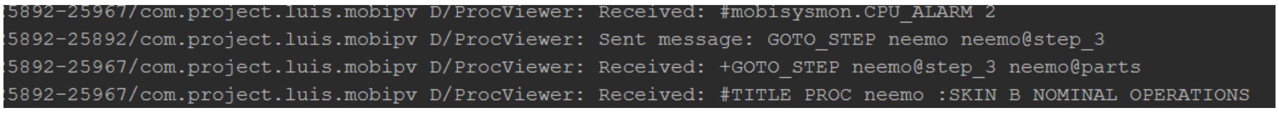


Figure 106: ProcViewer with "Go to..." dialog to jump to another step

Scrolling down the dialog of Figure 106, the step 3 appears and after pressing it the command “GOTO_STEP neemo neemo@step_3” is sent to the core, its parameters are similar to the ones explained in last step 17. The core confirms with the message: “GOTO_STEP neemo@step_3 neemo@parts” (See Figure 107).



```
5892-25967/com.project.luis.mobipv D/ProcViewer: Received: #mobisysmon.CPU_ALARM 2
5892-25892/com.project.luis.mobipv D/ProcViewer: Sent message: GOTO_STEP neemo neemo@step_3
5892-25967/com.project.luis.mobipv D/ProcViewer: Received: +GOTO_STEP neemo@step_3 neemo@parts
5892-25967/com.project.luis.mobipv D/ProcViewer: Received: #TITLE PROC neemo :SKIN B NOMINAL OPERATIONS
```

Figure 107: Exchange of messages when using "Go to..." tool to jump to Step 3

At the end the green marker is over Step 3 as shown in Figure 108

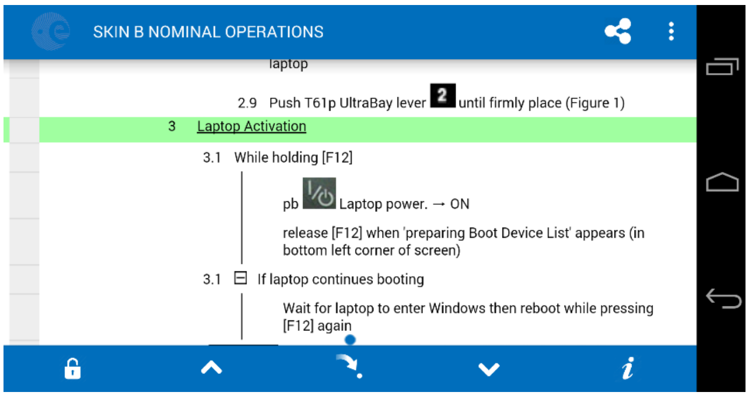
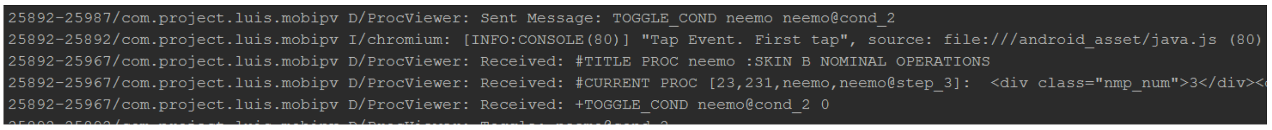

SKIN B NOMINAL OPERATIONS
laptop
2.9 Push T61p UltraBay lever 2 until firmly place (Figure 1)
3 Laptop Activation
3.1 While holding [F12]
pb Laptop power. → ON
release [F12] when 'preparing Boot Device List' appears (in bottom left corner of screen)
3.1 If laptop continues booting
Wait for laptop to enter Windows then reboot while pressing [F12] again

Figure 108: ProcViewer screen with the green marker over Step 3 after jumping using "Go to..." tool

20. Press on the toggle box, ⊞, on step 3.1 to minimize it.

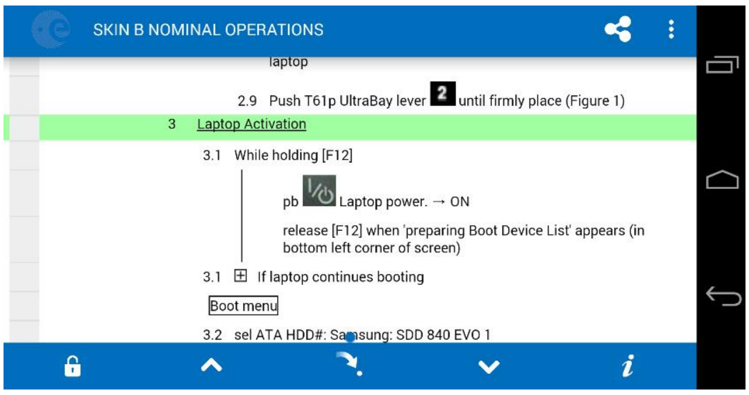
Figure 108 shows between the green marker and the bottom bar one step that has a toggle icon. The icon consists of a minus icon inside a box, when it is pressed, the JavaScript function calls a method within ProcViewer class and it sends the command "TOGGLE_COND neemo neemo@cond_2" to the core. Within the command the procedure name and the ID of the step, where the icon is, are reflected. Then the core answers with "TOGGLE_COND neemo@cond_2 0". The number "0" shows that the toggle has been contracted. The "1" would mean that the toggle has been expanded. The message exchange is displayed in Figure 109



```
25892-25987/com.project.luis.mobipv D/ProcViewer: Sent Message: TOGGLE_COND neemo neemo@cond_2
25892-25892/com.project.luis.mobipv I/chromium: [INFO:CONSOLE(80)] "Tap Event. First tap", source: file:///android_asset/java.js (80)
25892-25967/com.project.luis.mobipv D/ProcViewer: Received: #TITLE PROC neemo :SKIN B NOMINAL OPERATIONS
25892-25967/com.project.luis.mobipv D/ProcViewer: Received: #CURRENT PROC [23,231,neemo,neemo@step_3]:  <div class="nmp_num">3</div><
25892-25967/com.project.luis.mobipv D/ProcViewer: Received: +TOGGLE_COND neemo@cond_2 0
```

Figure 109: Message exchange when the toggle is contracted

The result is shown in Figure 110



SKIN B NOMINAL OPERATIONS

laptop

2.9 Push T61p UltraBay lever 2 until firmly place (Figure 1)

3 Laptop Activation

3.1 While holding [F12]

pb Laptop power. → ON

release [F12] when 'preparing Boot Device List' appears (in bottom left corner of screen)

3.1 ⊞ If laptop continues booting

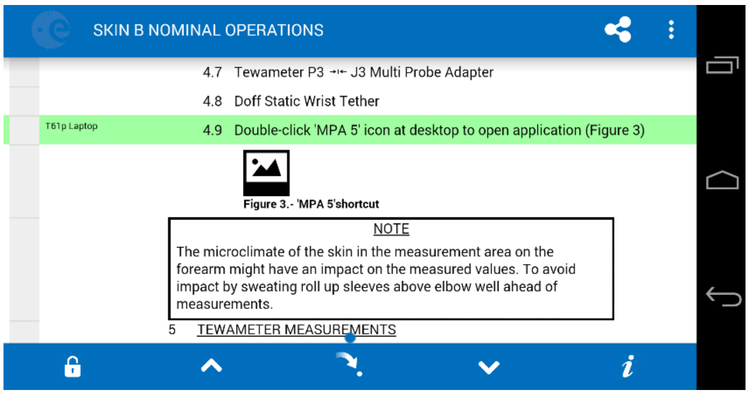
Boot menu

3.2 sel ATA HDD#: Samsung: SDD 840 EVO 1

Figure 110: ProcViewer screen when the toggle in Step 3.1 is contracted

21. Use again to jump to step 4.9 and press on the figure, check that an image is expanded.

Figure 111 shows what is displayed on the screen after jumping to Step 4.9. Just below the position of the green marker there is an icon of an image titled "Figure 3.- 'MPA 5'shortcut". This is a contracted image that has to be pressed to be expanded. The shown content after pressing on the image can be seen in Figure 112.



4.7 Tewameter P3 →← J3 Multi Probe Adapter

4.8 Doff Static Wrist Tether

T61p Laptop 4.9 Double-click 'MPA 5' icon at desktop to open application (Figure 3)

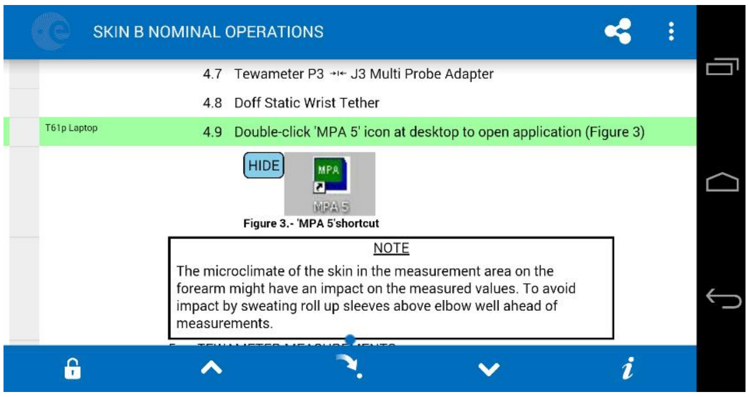
Figure 3.- 'MPA 5'shortcut

NOTE

The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

5 TEWAMETER MEASUREMENTS

Figure 111:ProcViewer screen after jumping to Step 4.9



4.7 Tewameter P3 →← J3 Multi Probe Adapter

4.8 Doff Static Wrist Tether

T61p Laptop 4.9 Double-click 'MPA 5' icon at desktop to open application (Figure 3)

Figure 3.- 'MPA 5'shortcut

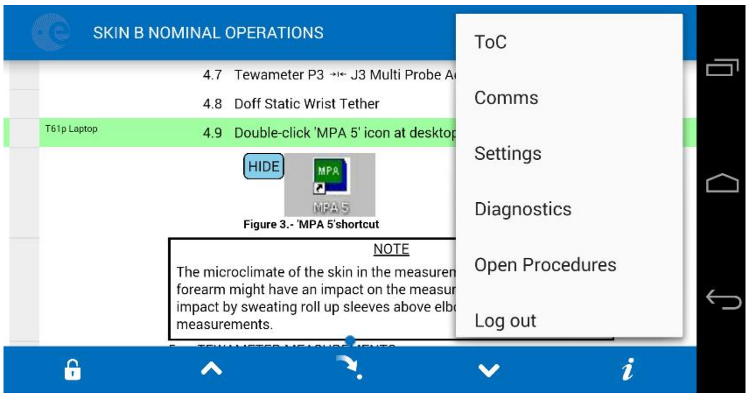
NOTE

The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

Figure 112: ProcViewer screen after expanding the image

22. Open now the Side Menu using ⋮ button. Select TOC to come back to the MainMenu Screen, select "Local Procedures", and open "RASPBERRY PI ASSEMBLY".

The Side Menu in this screen is shown in Figure 113 , similar to the menu in the rest of the activities but with an additional tab "Open Procedures" that shows a list of open procedures.



4.7 Tewameter P3 →← J3 Multi Probe A

4.8 Doff Static Wrist Tether

T61p Laptop 4.9 Double-click 'MPA 5' icon at desktop

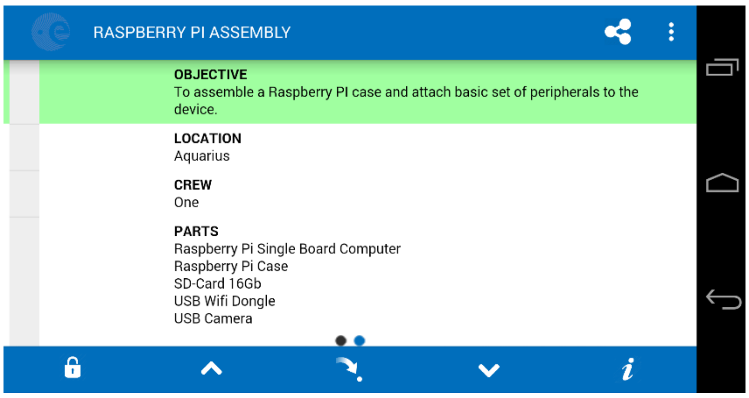
Figure 3.- 'MPA 5'shortcut

NOTE

The microclimate of the skin in the measuren
forearm might have an impact on the measur
impact by sweating roll up sleeves above elb
measurements.

Figure 113: Side Menu in ProcViewer Screen

After opening the new procedure, the green marker is located in the first step as shown in Figure 114.



OBJECTIVE
To assemble a Raspberry PI case and attach basic set of peripherals to the device.

LOCATION
Aquarius

CREW
One

PARTS
Raspberry Pi Single Board Computer
Raspberry Pi Case
SD-Card 16Gb
USB Wifi Dongle
USB Camera

Figure 114: ProcViewer Screen showing "RASPBERRY PI ASSEMBLY" procedure

23. When the procedure is loaded, select and jump to step 3.3, press the video image and that after pressing on play it starts playing. Pause it.

Similarly as before, when using the "Go to…" function, after pressing the button at the centre of the bottom bar, a menu is displayed with a list of all the steps of the procedure. Scrolling the list until "Step 3.3" appears, the last thing is to press on it to have as result the screen shown in Figure 115.

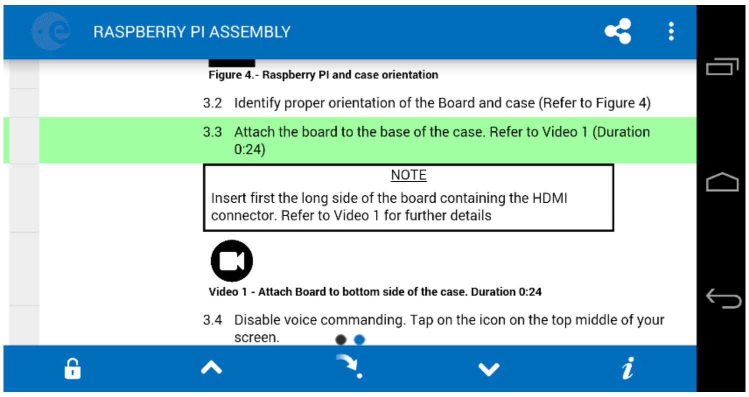


Figure 4.- Raspberry PI and case orientation

3.2 Identify proper orientation of the Board and case (Refer to Figure 4)

3.3 Attach the board to the base of the case. Refer to Video 1 (Duration 0:24)

NOTE

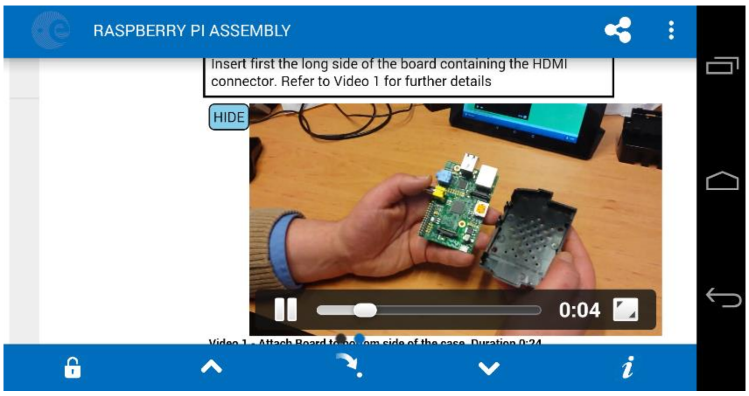
Insert first the long side of the board containing the HDMI connector. Refer to Video 1 for further details

Video 1 - Attach Board to bottom side of the case. Duration 0:24

3.4 Disable voice commanding. Tap on the icon on the top middle of your screen.

Figure 115: ProcViewer screen showing Step 3.3 of "RASPBERRY PI ASSEMBLY" procedure

When the video icon of Figure 115 is pressed, a multimedia video player is displayed allowing the user to play the video. Both moving images and audio show the user how to attach the Raspberry to the case (see Figure 116).



Insert first the long side of the board containing the HDMI connector. Refer to Video 1 for further details

Figure 116: ProcViewer screen with a video playing in "RASPBERRY PI ASSEMBLY" procedure

24. Check that on the lower part of the screen just above the lower bar there are two dots, one blue and one black.

In Figure 114, Figure 115 and Figure 116 at the bottom of the screen and just above the bottom bar there are two dots, in two different colours, some information can be obtained from these dots:

- The total amount of shown dots on the screen is related with the total number of open procedures at the moment, that means that the procedures are opened and easily accessible, the user does not have to access then and open from the Main Menu. As well as the position where the user is working (green marker) is not lost.

- The black and blue colours represent the position of the procedures in the list of open procedures. The black dot is the no active procedure, in this case “SKIN B NOMINAL OPERATIONS” and occupies the first position because it was the first one to be open. The blue dot represents the active procedure that it is “RASPBERRY PI ASSEMBLY”.

Actually if the user presses on the option “Open Procedures” on the Side Menu as shown in Figure 113, a new dialog appears and shows the list of procedures (see Figure 117), with the same order that the dots have. This list also offers the possibility of changing procedure by pressing on one.

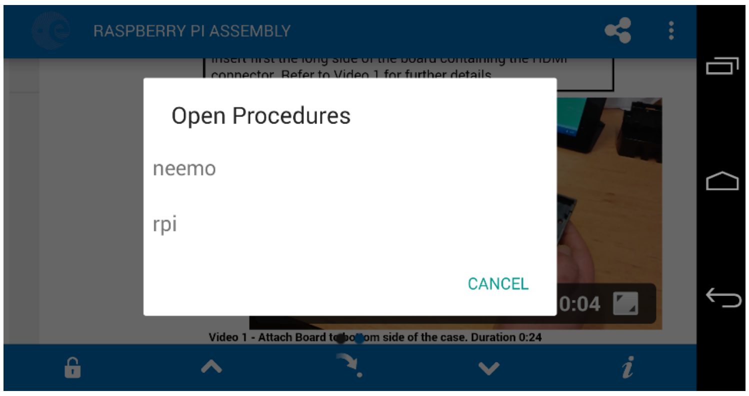

RASPBERRY PI ASSEMBLY
Open Procedures
neemo
rpi
CANCEL
0:04
Video 1 - Attach Board to bottom side of the case. Duration 0:24

Figure 117: "Open Procedures" dialog on ProcViewer screen

25. **Swipe left and check that the procedure “SKIN B NOMINAL OPERATIONS” is loaded (check the title on the action bar on the top left part of the screen). Check also that the blue dot is now a different one.**

The feature of swiping left and right allows the user navigate through open procedures list to change the active procedure. In Figure 118, the swipe action has just happened and a dialog (highlighted with a circle) is shown to the user, informing about the procedure that is going to be loaded. The tittle (highlighted with a rectangle) is also one of the first things that changes.

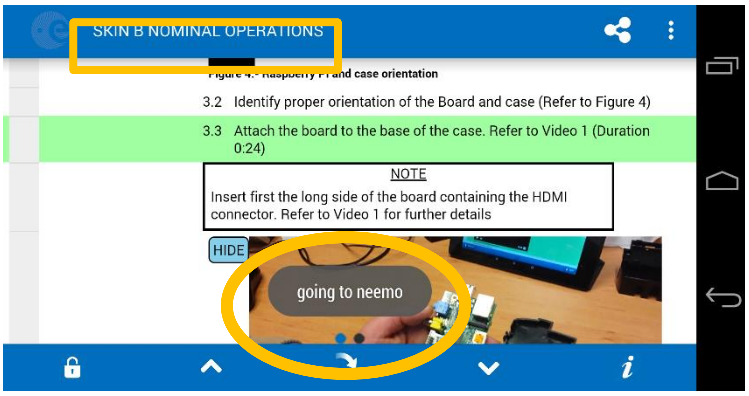

SKIN B NOMINAL OPERATIONS

Figure 4.- Raspberry Pi and case orientation

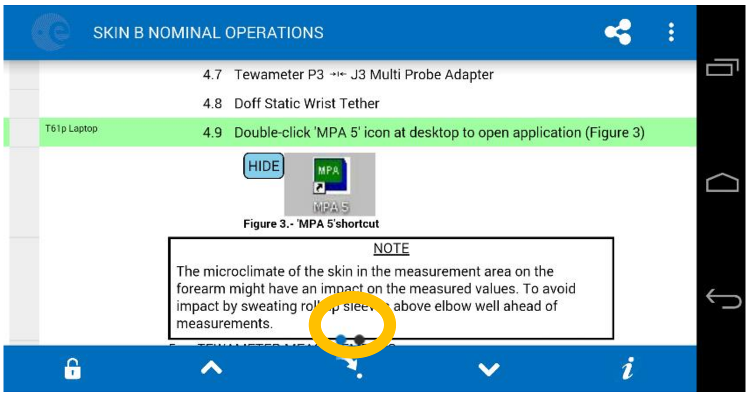
3.2 Identify proper orientation of the Board and case (Refer to Figure 4)

3.3 Attach the board to the base of the case. Refer to Video 1 (Duration 0:24)

NOTE
Insert first the long side of the board containing the HDMI connector. Refer to Video 1 for further details

Figure 118: ProcViewer screen changing active procedure after swiping left

Figure 119 shows the active procedure already changed


SKIN B NOMINAL OPERATIONS

4.7 Tewameter P3 →← J3 Multi Probe Adapter

4.8 Doff Static Wrist Tether

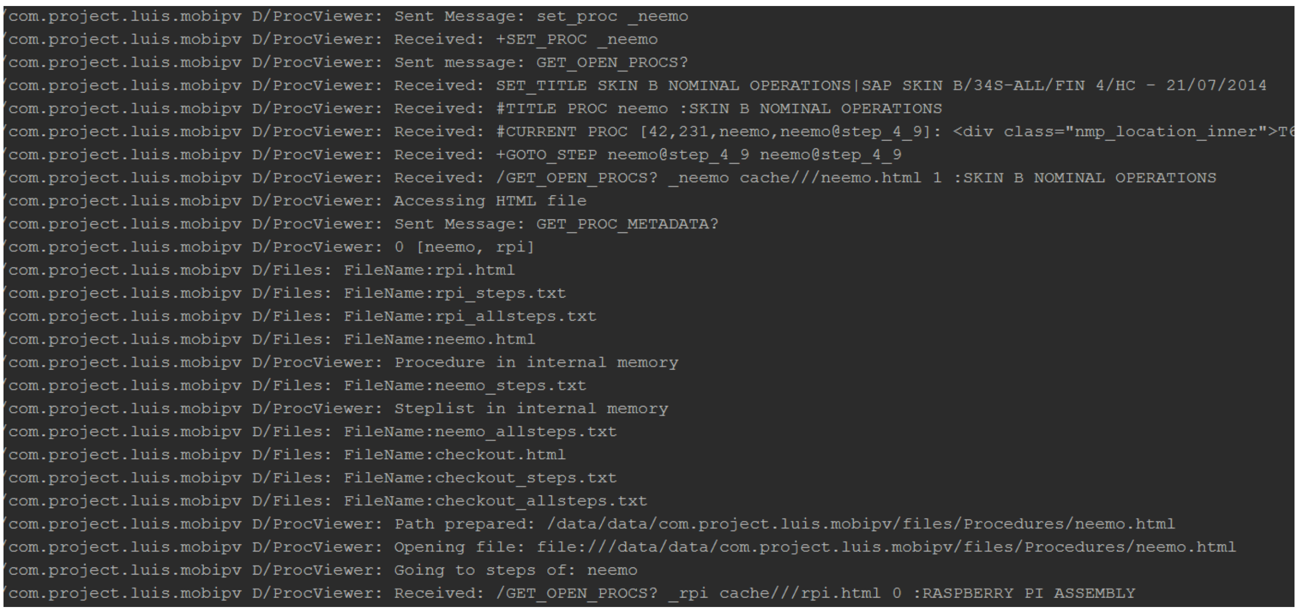
T61p Laptop 4.9 Double-click 'MPA 5' icon at desktop to open application (Figure 3)

Figure 3.- 'MPA 5'shortcut

NOTE
The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

Figure 119: “SKIN B NOMINAL OPERATIONS” procedure on ProcViewer screen just loaded after swiping

Now, let’s have a closer look to the message exchange (see Figure 120)



```
com.project.luis.mobipv D/ProcViewer: Sent Message: set_proc _neemo
com.project.luis.mobipv D/ProcViewer: Received: +SET_PROC _neemo
com.project.luis.mobipv D/ProcViewer: Sent message: GET_OPEN_PROCS?
com.project.luis.mobipv D/ProcViewer: Received: SET_TITLE SKIN B NOMINAL OPERATIONS|SAP SKIN B/34S-ALL/FIN 4/HC - 21/07/2014
com.project.luis.mobipv D/ProcViewer: Received: #TITLE PROC neemo :SKIN B NOMINAL OPERATIONS
com.project.luis.mobipv D/ProcViewer: Received: #CURRENT PROC [42,231,neemo,neemo@step_4_9]: <div class="nmp_location_inner">T
com.project.luis.mobipv D/ProcViewer: Received: +GOTO_STEP neemo@step_4_9 neemo@step_4_9
com.project.luis.mobipv D/ProcViewer: Received: /GET_OPEN_PROCS? _neemo cache///neemo.html 1 :SKIN B NOMINAL OPERATIONS
com.project.luis.mobipv D/ProcViewer: Accessing HTML file
com.project.luis.mobipv D/ProcViewer: Sent Message: GET_PROC_METADATA?
com.project.luis.mobipv D/ProcViewer: 0 [neemo, rpi]
com.project.luis.mobipv D/Files: FileName:rpi.html
com.project.luis.mobipv D/Files: FileName:rpi_steps.txt
com.project.luis.mobipv D/Files: FileName:rpi_allsteps.txt
com.project.luis.mobipv D/Files: FileName:neemo.html
com.project.luis.mobipv D/ProcViewer: Procedure in internal memory
com.project.luis.mobipv D/Files: FileName:neemo_steps.txt
com.project.luis.mobipv D/ProcViewer: Steplist in internal memory
com.project.luis.mobipv D/Files: FileName:neemo_allsteps.txt
com.project.luis.mobipv D/Files: FileName:checkout.html
com.project.luis.mobipv D/Files: FileName:checkout_steps.txt
com.project.luis.mobipv D/Files: FileName:checkout_allsteps.txt
com.project.luis.mobipv D/ProcViewer: Path prepared: /data/data/com.project.luis.mobipv/files/Procedures/neemo.html
com.project.luis.mobipv D/ProcViewer: Opening file: file:///data/data/com.project.luis.mobipv/files/Procedures/neemo.html
com.project.luis.mobipv D/ProcViewer: Going to steps of: neemo
com.project.luis.mobipv D/ProcViewer: Received: /GET_OPEN_PROCS? _rpi cache///rpi.html 0 :RASPBERRY PI ASSEMBLY
```

Figure 120: Message exchange when swiping from one procedure to other

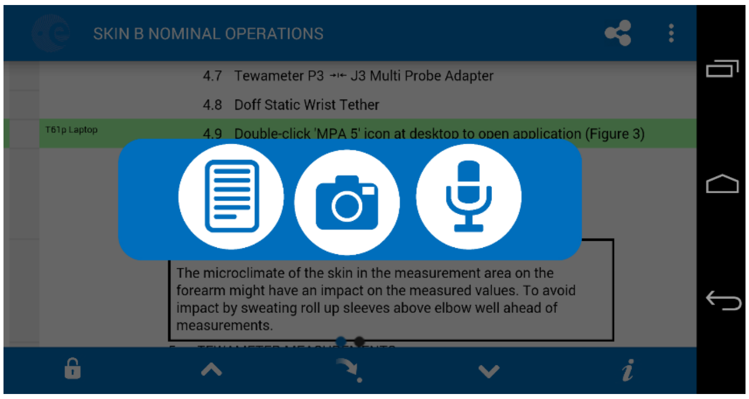
The steps to change the procedure, not only when swiping but also when pressing an option of the dialog shown in Figure 117, according to the messages in Figure 120, are:

a. "Sent Message: set_proc _neemo": By sending the command to the core, with the ID of the new procedure to be loaded, the app requests the core to change the active and therefore the procedure active on the screen.
b. "Received: +SET_PROC _neemo": The core confirms the request and changes the active procedure.
c. "Sent Message: GET_OPEN_PROCS?": After receiving the confirmation in Step b, the app asks for the list of open procedures.
d. The next messages that arrive are related with the tittle, the message "#TITLE PROC …" is read by the app to modify the title on the top bar, that is why the title is the first thing to change.
e. The message "+GOTO_STEP …" indicates the step where the procedure was left last time the user was working on it. As the screen is not yet loaded, this message is stored for a later use.
f. The first "/GET_OPEN_PROCS? …" arrives with the first open procedure, this message carries the number "1" between the path and the title of the procedure, this number indicates that this one is the active procedure, so the app starts loading it.
g. The message "Accessing HTML file" is printed, that means that the method within ProcViewer activity to load the procedure has been called. The first thing this method does is look for the procedure already stored in the internal app folders.

h. In this case the procedure is already stored and some messages containing the full list of stored procedures are printed. In this case, the app is trying to load “neemo” and the needed files are: “neemo.html” (the procedure itself with the new header added by the app), “neemo_steps.txt” (the list of steps that is shown in “Go to…” feature) and “neemo_allsteps.txt” (a more complete list of steps for helping the app when navigating or adding notes)
i. Once the file is found the message “Path prepared …” is printed
j. The message “Opening file …” occurs when the procedure path is being loaded in the WebView.
k. The message “Going to steps …” is printed when the app is going to place the green marker, here is when the message stored in Step e is used.

26. In the actual procedure and step, press on the left part of the screen, the area denoted by a darker colour. Check that the note taking dialog is shown.

The dialog is shown in Figure 121.



4.7 Tewameter P3 →|← J3 Multi Probe Adapter
4.8 Doff Static Wrist Tether
4.9 Double-click 'MPA 5' icon at desktop to open application (Figure 3)

The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

Figure 121: Note Taking Dialog in ProcViewer screen

27. Press on the text note icon, denoted by , and write “Hi” on the box, press then on Save. Check that the note icon appears on the left area of the procedure.

When pressing on the text note icon the dialog is expanded and a white box appears below the three buttons for adding text and a save button to record the text note. A keyboard is shown to the user to write after tapping over it. See Figure 122 with the text already written.

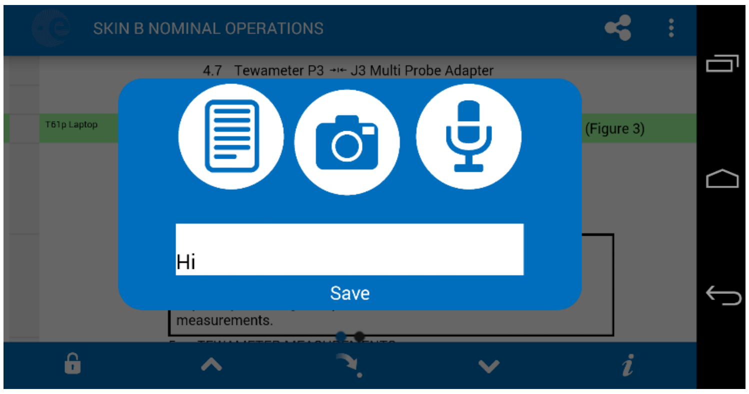
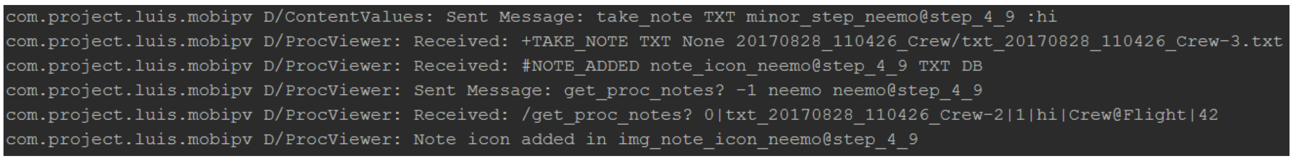

SKIN B NOMINAL OPERATIONS
4.7 Tewameter P3 ⇢ J3 Multi Probe Adapter
T61p Laptop
(Figure 3)
Hi
Save
measurements.

Figure 122: Text note written in white box on the Note Taking Dialog

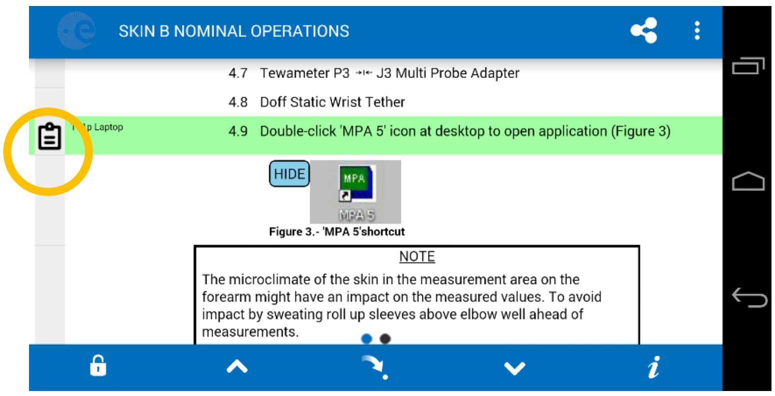
Save button executes a method that writes a command to the socket that asks the core to save the text note. The command is "take_note TXT minor_step_neemo@step_4_9 :hi", with the ID of the step and the text "hi" that is what the user has written. The system then confirms with the usual "+TAKE_NOTE …" and sends another message that notifies the addition of the note to the database "#NOTE_ADDED note_icon_neemo@step_4_9 TXT DB". Here is when the app, after receiving this message, asks for the note of the step included in this command ("neemo@step_4_9"). For this purpose, the app uses the command "get_proc_notes? -1 neemo neemo@step_4_9" and when the core answers with the file name, the path of the file, the type of note and the text of the note, the app stores all this information and prints the last message "Note icon added in …". See Figure 123 where all this message exchange is plotted.



```
com.project.luis.mobipv D/ContentValues: Sent Message: take_note TXT minor_step_neemo@step_4_9 :hi
com.project.luis.mobipv D/ProcViewer: Received: +TAKE_NOTE TXT None 20170828_110426_Crew/txt_20170828_110426_Crew-3.txt
com.project.luis.mobipv D/ProcViewer: Received: #NOTE_ADDED note_icon_neemo@step_4_9 TXT DB
com.project.luis.mobipv D/ProcViewer: Sent Message: get_proc_notes? -1 neemo neemo@step_4_9
com.project.luis.mobipv D/ProcViewer: Received: /get_proc_notes? 0|txt_20170828_110426_Crew-2|1|hi|Crew@Flight|42
com.project.luis.mobipv D/ProcViewer: Note icon added in img_note_icon_neemo@step_4_9
```

Figure 123: Message exchange when adding a text note

See Figure 124 where the note icon appears on the left of the step highlighted with a yellow circle.


SKIN B NOMINAL OPERATIONS

4.7 Tewameter P3 ⇥⇤ J3 Multi Probe Adapter

4.8 Doff Static Wrist Tether

4.9 Double-click 'MPA 5' icon at desktop to open application (Figure 3)

Figure 3.- 'MPA 5'shortcut

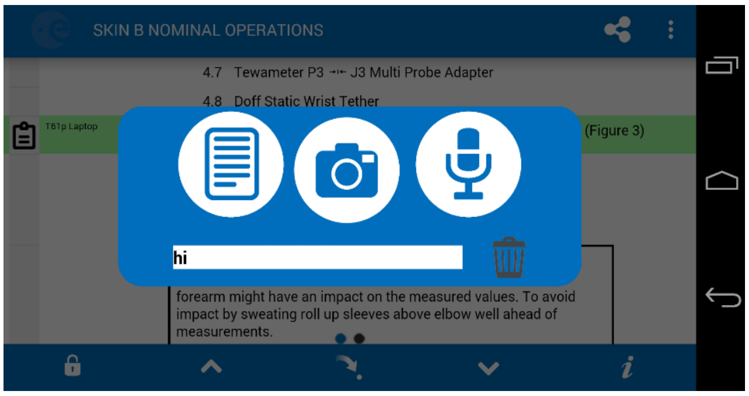
NOTE

The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

Figure 124: ProcViewer screen with the note icon on Step 4.9

28. Press now over the note icon and check the text is "Hi". Press away of the dialog to close it.

Figure 125 shows the dialog with the added text note. No messages happen here because all the needed information to show is stored in some variables in the app.


SKIN B NOMINAL OPERATIONS

4.7 Tewameter P3 ⇥⇤ J3 Multi Probe Adapter

4.8 Doff Static Wrist Tether

(Figure 3)

hi

forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

Figure 125: Note Taking Dialog showing the added text note and trash can to remove it

29. Press on the left part of the screen of the next step and open again the note taking dialog.

The result is similar to the one shown in Figure 121, a new Dialog is open with the three options

30. Press on photo/video note taking, denoted by 📷, and check that a new dialog appears showing what the camera is recording.

For this step and some others, the USB camera have to be connected to the smartphone, if the camera is available and there is not any problem, when the user presses over the camera icon, the app sends the message "preview_start" and the system answer with "+PREVIEW_START". Then the dialog changes into the dialog shown in Figure 126.

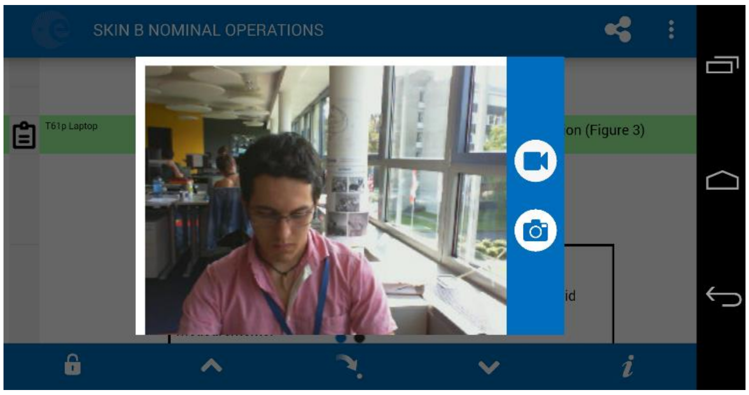


Figure 126: Camera dialog for recording video notes and taking photo notes

31. Press on the video recording button, denoted by , and check that the button changes to , then press again this last button. Check that the note icon appears on the left area of the procedure.

This camera dialog does not really access to the camera and the start video button does not really tell the camera to record a video. This button asks the core to access to the camera and start a video, this button implements the command "start_note video" that is sent to the core when it is pressed and changes into the stop button until the end of the recording. In this case, similarly to the starting video case, the stop video button implements the command "stop_note video", that is also sent to the core to stop and store the recording (see Figure 127).

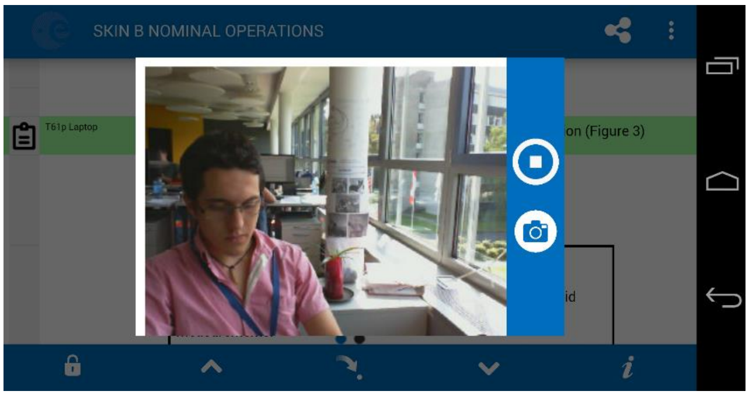


Figure 127: Camera dialog while the video is being recorded, the video camera button has changed into a stop button

32. Press on the note icon and press “Open video”. Watch the video and check that the content is correct.

Once it happens and, as before, the message exchange is the same to the one previously explained for Figure 123. The icon appears to the left of the step and information about the note is recorded in an internal variable, so when the user opens the note taking dialog that corresponds to this step the app knows that there is a video note and modifies the dialog as the one in Figure 128.

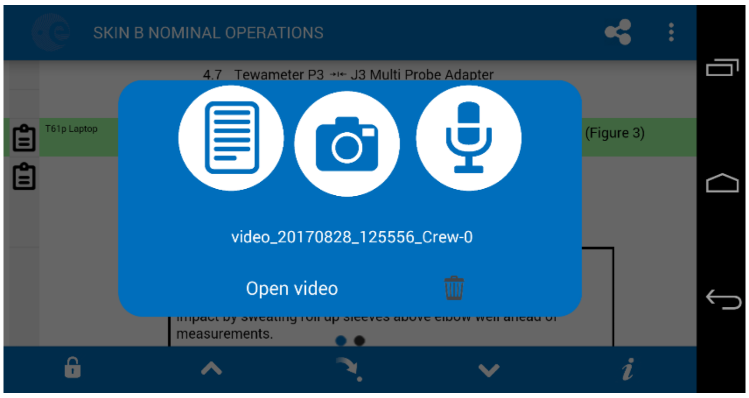

SKIN B NOMINAL OPERATIONS
4.7 Tewameter P3 J3 Multi Probe Adapter
T61p Laptop
(Figure 3)
video_20170828_125556_Crew-0
Open video
measurements.

Figure 128: Note Taking Dialog when a video note is available

For watching the video there is a class that changes the view and implements a video player that accesses to the path where the video note is (see Figure 129).

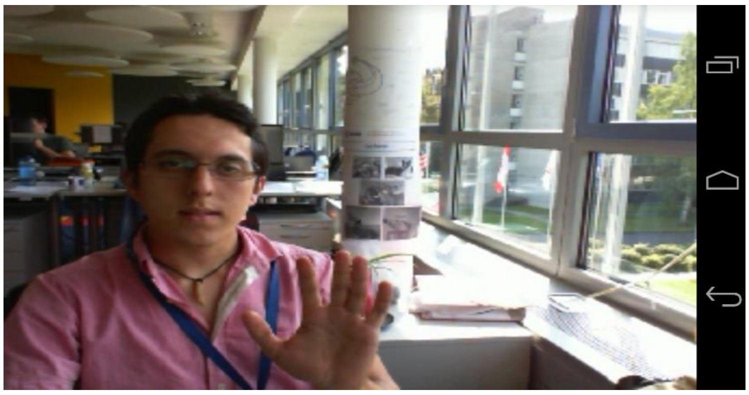
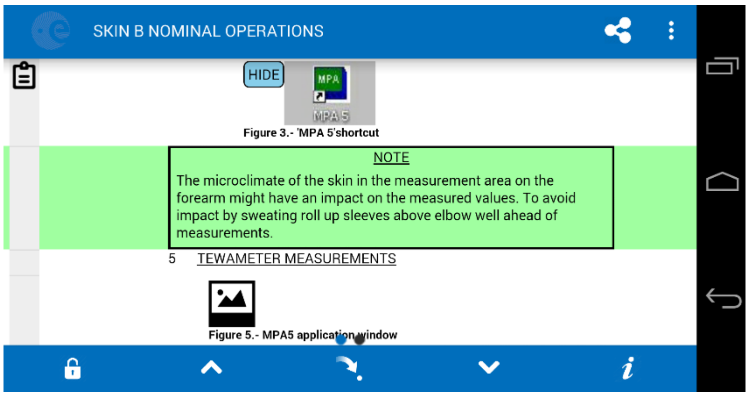


Figure 129: Video Player playing the video note

33. Press Android back button, ⮌ and next button twice

To close the Video Player view, the user has to press Android return button and by this way go back to the ProcViewer screen (see Figure 130)



SKIN B NOMINAL OPERATIONS
HIDE
MPA
MPA 5
Figure 3.- 'MPA 5'shortcut

NOTE

The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

5 TEWAMETER MEASUREMENTS

Figure 5.- MPA5 application window

Figure 130: ProcViewer screen after recording the video note

34. Open again the note taking dialog and press again on photo/video button, this time press on . Check that the note icon appears on the left area of the procedure.

The Note Taking Dialog screen is the same that the screen in Figure 121, because there is not any note recorded for this step, again the camera icon opens a new dialog showing what the camera is recording as in Figure 126.

Pressing on the photo button involves that the app sends the message “take_note photo”, and the core saves a photo as a note for this step. The app closes the dialog and the new note icon appears on the margin. If the user presses on it the note taking dialog will be opened as shown in

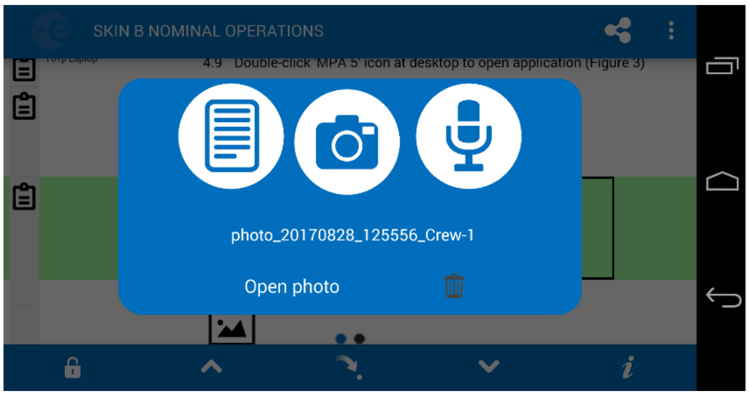

SKIN B NOMINAL OPERATIONS
4.9 Double-click MPA 5 icon at desktop to open application (Figure 3)
photo_20170828_125556_Crew-1
Open photo

Figure 131: Note taking Dialog when there is a recorded photo note

35. **Press on the note icon and press “Open photo”. Watch the photo and check that the content is correct.**

When the app opens the photo, it loads an image view looking for the it in the path where the core has located it (see Figure 132).

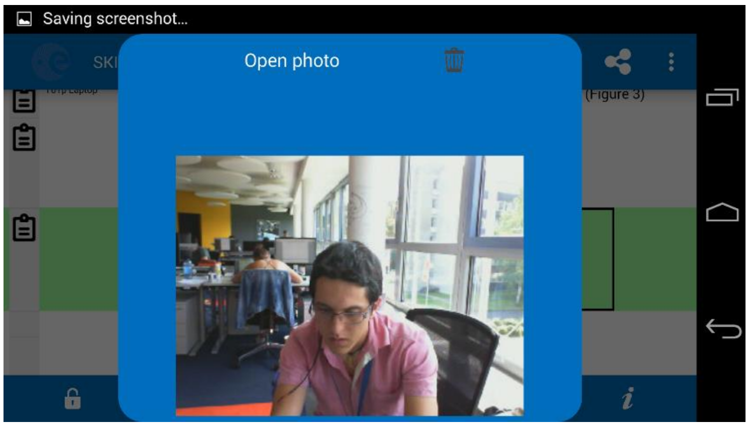


Figure 132: Note Taking Dialog when the photo note is opened

36. **Press away of the dialog to close it and open the note taking dialog of the next step and select audio recording, denoted by . Check that the icon changes to and the**

message "Recording…" is shown. Say something and press stop button, check that the dialog closes and the message "Stopping…" is displayed.

Similarly to the photo or video note taking, the audio button implements a command that is sent to the core and asks it to start a recording from the microphone. The command is "start_note audio" and the system confirms the starting of the recording process with "+start_note audio".

This answer from the core activates the method that plots the message "Recording…" and changes the audio note recording into a stop one (see Figure 133).

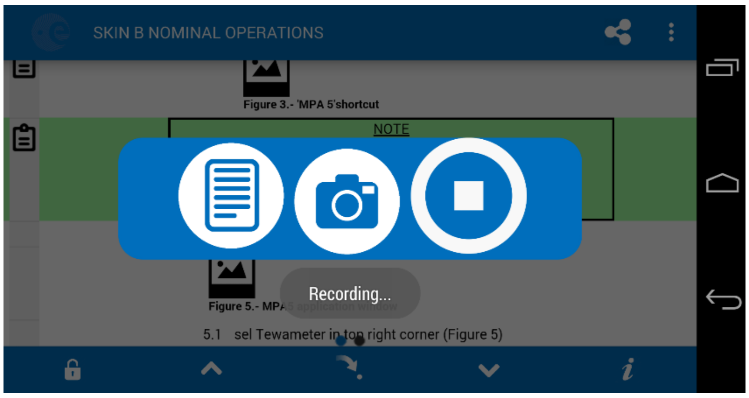
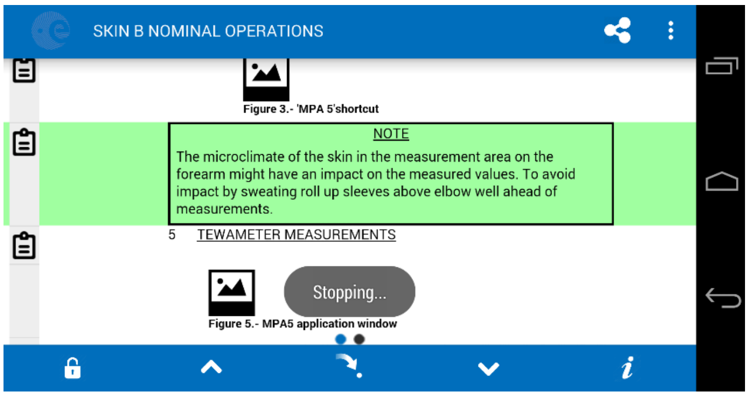

SKIN B NOMINAL OPERATIONS
Figure 3.- 'MPA 5'shortcut
NOTE
Recording...
Figure 5.- MPA5 application window
5.1 sel Tewameter in top right corner (Figure 5)

Figure 133: Note Taking Dialog recording audio, with a pop up message and the stop button instead of the recording audio one

Pressing the stops button involves the app sends the core the command "stop_note audio", the dialog closes, the text "Stopping…" pops up, so the user knows the recording has been successfully stopped, and the message "#NOTE_ADDED …" arrives as before. Finally, the app adds the note icon and introduces the note parameters to an internal variable (see Figure 134).



SKIN B NOMINAL OPERATIONS

Figure 3.- 'MPA 5'shortcut

NOTE

The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

5 TEWAMETER MEASUREMENTS

Stopping...

Figure 5.- MPA5 application window

Figure 134: ProcViewer screen after the note taking dialog is closed and the audio recording is finished

37. **Open again note taking dialog of the same step and listen to the audio file by pressing on play. After listening to it press on the trash can to delete it. Check that the note icon on the left part disappears.**

With the collected information from the message of the core "#NOTE_ADDED …", the app builds the dialog shown in Figure 135 by adding the title of the note, a play button, a progress bar, a time counter and the trash bin to remove it. In the figure the note is being played because the user has pressed play button.

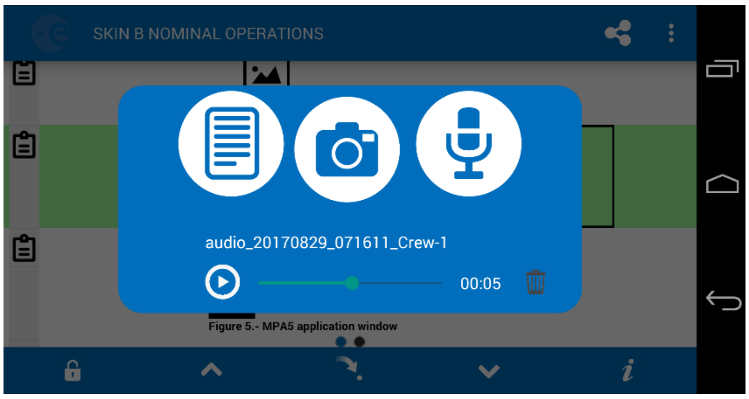
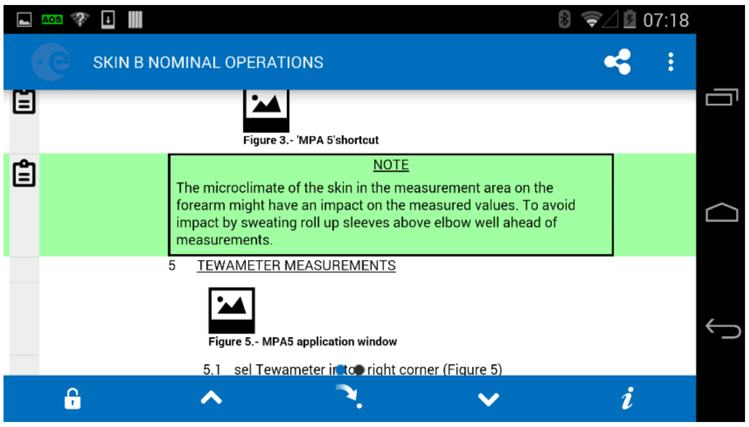

SKIN B NOMINAL OPERATIONS
audio_20170829_071611_Crew-1
00:05
Figure 5.- MPA5 application window

Figure 135: Note taking Dialog with an audio note saved, the audio note is being played

Pressing the trash bin sends to the core the message "del_note CURRENT + audio_20170829_071611_Crew1" and the system confirms with "+del_note CURRENT + audio_20170829_071611_Crew1". By this way the app reads the message and removes the note from the internal variable were it was stored and hides the note icon. See Figure 136. This same process happens when another note is removed, no matter if it is text, a photo or a video.


07:18
SKIN B NOMINAL OPERATIONS
Figure 3.- 'MPA 5'shortcut

NOTE
The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

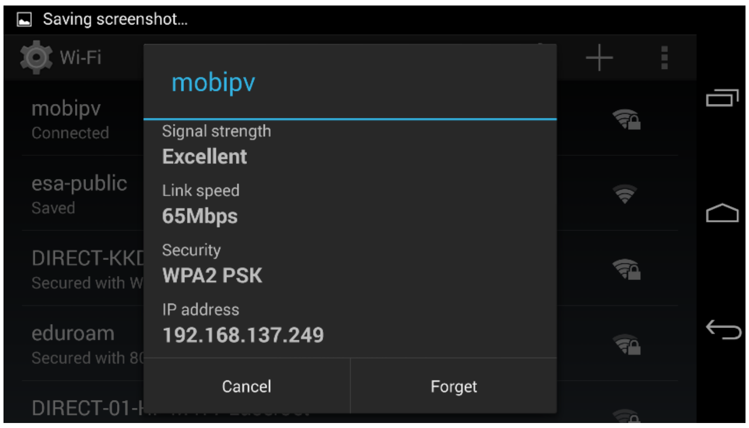
5 TEWAMETER MEASUREMENTS

Figure 5.- MPA5 application window
5.1 sel Tewameter in top right corner (Figure 5)

Figure 136: ProcViewer screen after the note icon has been removed

38. Display now the Android tools menu by swiping down from the top part of the screen. Press on the top right icon and press again on WiFi menu. On the new screen press on the WLAN where the Nexus 5 is connected, that should be the same created by the laptop and write down the IP address.

From this point a cooperative session is going to be stablished simulating that a laptop (on ground) is connected to the app (on the ISS) and both have to work together in a procedure. For this purpose, the mobiPV core in the virtual machine of the laptop needs the IP address of the Nexus 5, which is 192.168.137.249 and is connected to the wireless access point created by the laptop and whose name is “mobipv” (see Figure 137).


Saving screenshot...

mobipv

Signal strength
Excellent

Link speed
65Mbps

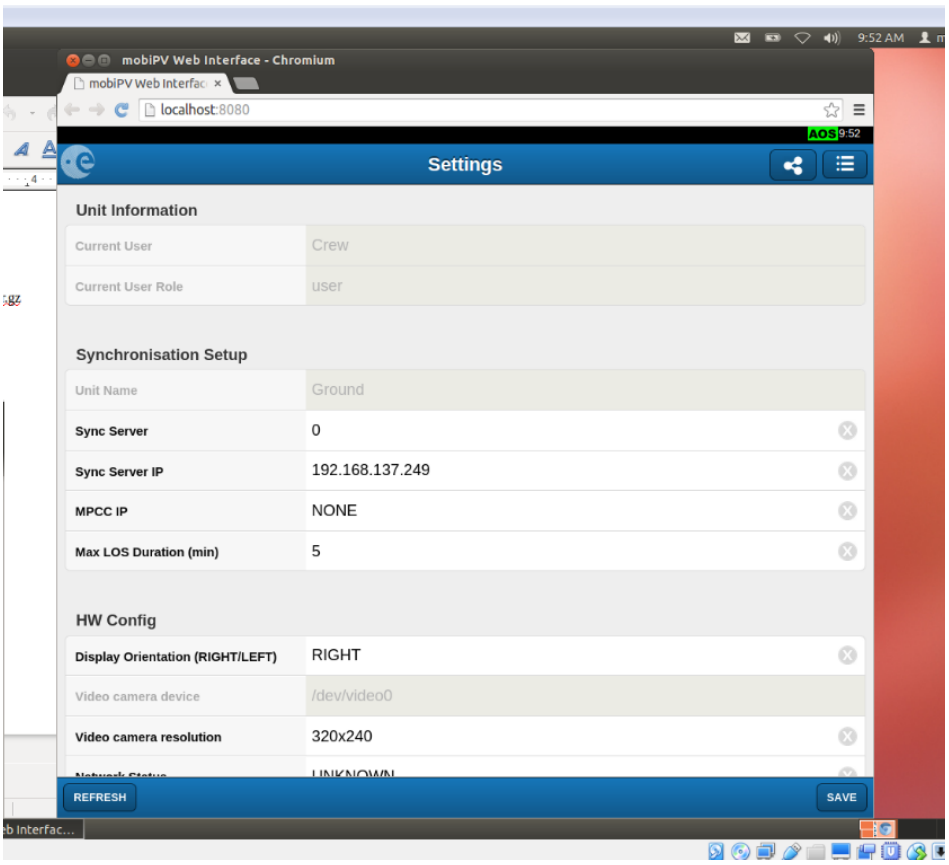
Security
WPA2 PSK

IP address
192.168.137.249

Figure 137: WLAN information on the Nexus 5

39. On the other mobiPV core go to Settings and on “Sync Server” write “0” and on “Sync Server IP” write the IP address of the Nexus 5. Restart the mobiPV system.

Figure 138 shows mobiPV interface in the Linux virtual machine. The interface is on the Settings window where the IP has been modified to the Nexus 5 one taken in the previous step


mobiPV Web Interface - Chromium

localhost:8080

Settings

Unit Information

Current User	Crew
Current User Role	user

Synchronisation Setup

Unit Name	Ground
Sync Server	0
Sync Server IP	192.168.137.249
MPCC IP	NONE
Max LOS Duration (min)	5

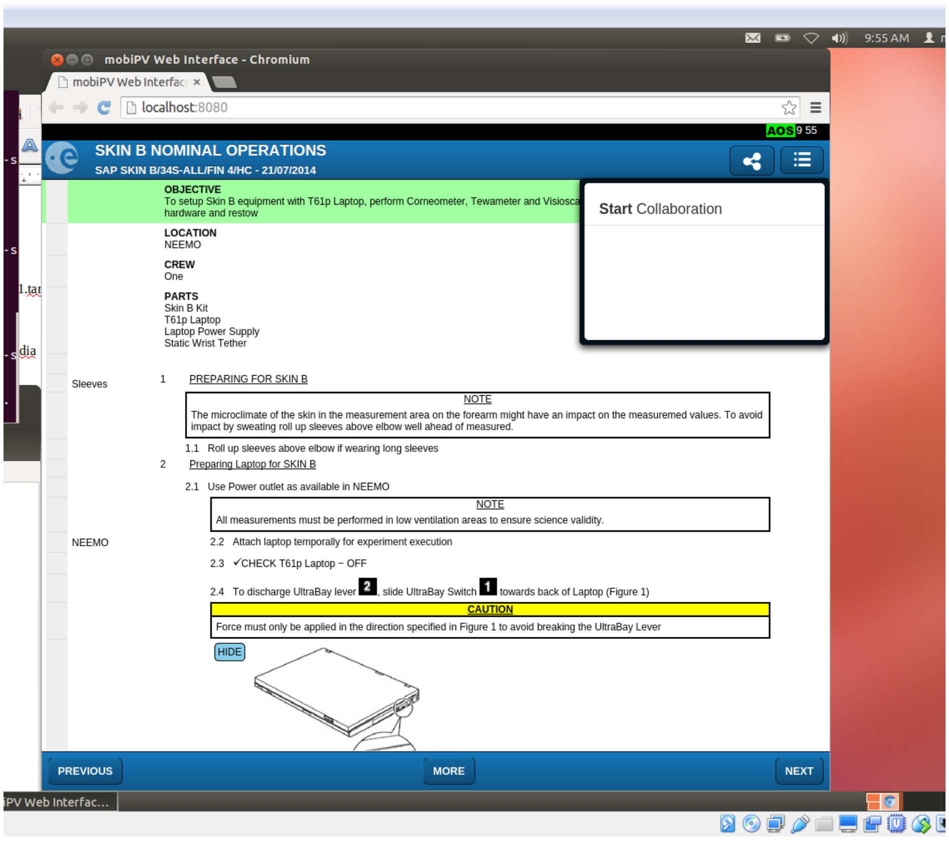
HW Config

Display Orientation (RIGHT/LEFT)	RIGHT
Video camera device	/dev/video0
Video camera resolution	320x240
Network Status	UNKNOWN

Figure 138: mobiPV, running on a Linux virtual machine, interface on Settings screen

40. On the same Linux mobiPV, open "SKIN B NOMINAL OPERATIONS" procedure and press now Sync Session on the action bar on the top of the screen denoted by . Check that a new dialog pops and press on "Start Collaboration". Check that on the app one message has just arrived, denoted by , on the top left part of the screen.

Figure 139 shows the ProcViewer screen on the mobiPV software interface running on the Linux virtual machine. The open procedure is "SKIN B NOMINAL OPERATIONS" and as it has just started, the step is the first one (see Figure 139)



SKIN B NOMINAL OPERATIONS

SAP SKIN B/34S-ALL/FIN 4/HC - 21/07/2014

Start Collaboration

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visiosca
hardware and restow

LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Sleeves 1 PREPARING FOR SKIN B

NOTE
The microclimate of the skin in the measurement area on the forearm might have an impact on the measuremed values. To avoid impact by sweating roll up sleeves above elbow well ahead of measured.

1.1 Roll up sleeves above elbow if wearing long sleeves

2 Preparing Laptop for SKIN B

2.1 Use Power outlet as available in NEEMO

NOTE
All measurements must be performed in low ventilation areas to ensure science validity.

NEEMO 2.2 Attach laptop temporally for experiment execution

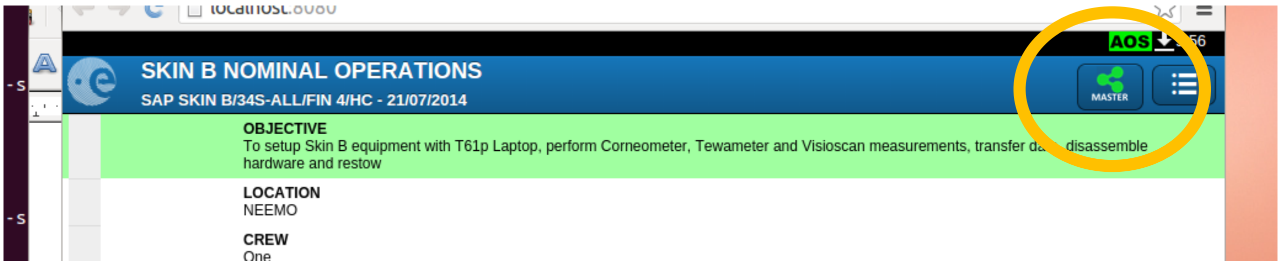
2.3 ✓CHECK T61p Laptop − OFF

2.4 To discharge UltraBay lever 2, slide UltraBay Switch 1 towards back of Laptop (Figure 1)

CAUTION
Force must only be applied in the direction specified in Figure 1 to avoid breaking the UltraBay Lever

Figure 139: mobiPV, running on a Linux virtual machine, interface on ProcViewer screen

On the top right corner of the interface in Figure 139, there is a window opened with the text "Start Collaboration". Pressing on this text involves that this core will send a message to the core in "192.168.137.249" (the core running on the Nexus 5) and inform it that a session has been established. The icon on the interface changes then to green and the text "MASTER" is added (see Figure 140).



SKIN B NOMINAL OPERATIONS

SAP SKIN B/34S-ALL/FIN 4/HC - 21/07/2014

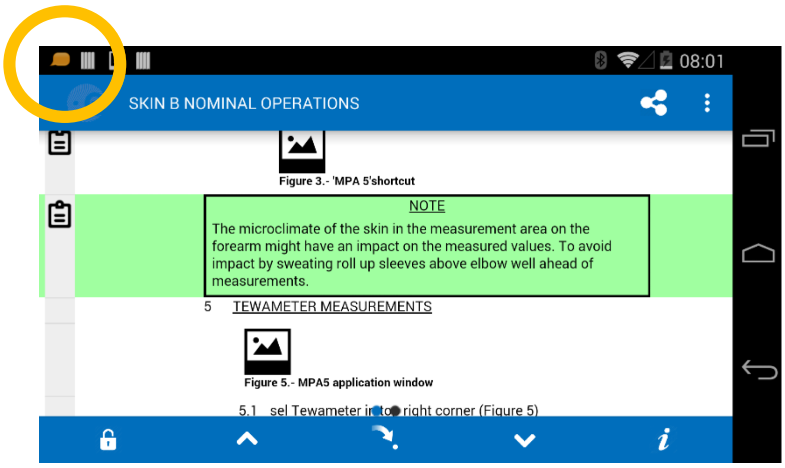
OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer da disassemble
hardware and restow

LOCATION
NEEMO

CREW
One

Figure 140: mobiPV interface on Linux has been set as Master on a Collaborative Session

As the core in the Nexus 5 receives the message from the core in Linux, it resends it to mobicontrol component where the app is connected. When this message is read by the app, a new method is called and an orange chat bubble is added to the Android State Bar (see Figure 141).



08:01
SKIN B NOMINAL OPERATIONS
Figure 3.- 'MPA 5'shortcut

NOTE

The microclimate of the skin in the measurement area on the forearm might have an impact on the measured values. To avoid impact by sweating roll up sleeves above elbow well ahead of measurements.

5 TEWAMETER MEASUREMENTS

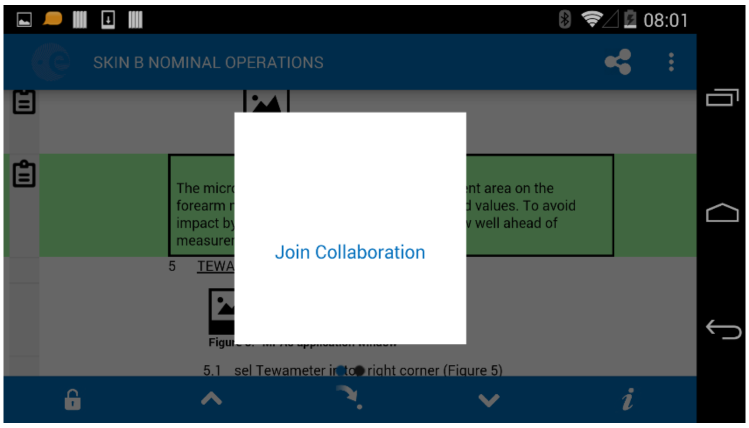
Figure 5.- MPA5 application window
5.1 sel Tewameter in top right corner (Figure 5)

Figure 141: ProcViewer screen with an orange chat bubble on the State Bar

Some steps later the meaning of this icon and what involves pressing on it will be discussed.

41. Press in the app and press on join collaboration. Check that the green marker jumps to the same position where it is in the Linux mobiPV.

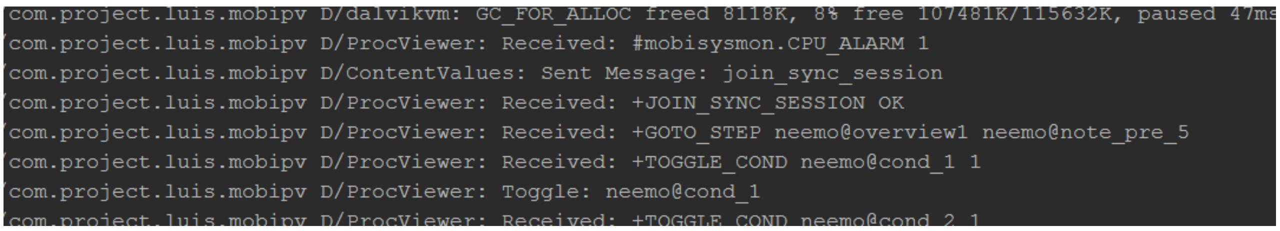
Pressing on the Collaboration icon opens a dialog where the text "Join Collaboration" is printed. The fact that there is an already started session is the reason why this text is plotted and not "Start Collaboration", for example (see Figure 142).



08:01
SKIN B NOMINAL OPERATIONS
Join Collaboration
5.1 sel Tewameter in top right corner (Figure 5)

Figure 142: ProcViewer screen where the Collaboration dialog is open

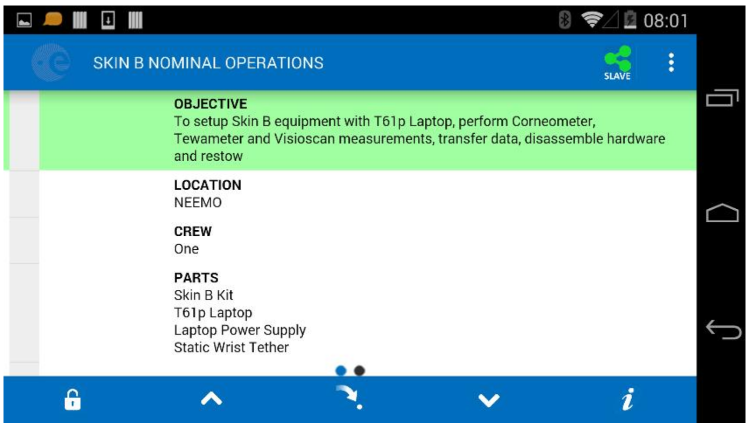
When the user presses the text on Figure 142, the app sends the message "join_sync_session" and the Linux laptop core takes somehow control over the Nexus 5 core. The Nexus 5 core sends the confirmation of the message to the app and that means that now Nexus 5 core is working in “SLAVE” mode. Immediately after the Nexus 5 is working on the collaboration session, the smartphone core sends to the app the new position of the green marker with the command “+GOTO_STEP …” and some other “+TOGGLE_COND” messages to display all the toggle boxes of the procedure (see Figure 143).



```
com.project.luis.mobipv D/dalvikvm: GC_FOR_ALLOC freed 8118K, 8% free 107481K/115632K, paused 47ms
com.project.luis.mobipv D/ProcViewer: Received: #mobisysmon.CPU_ALARM 1
com.project.luis.mobipv D/ContentValues: Sent Message: join_sync_session
com.project.luis.mobipv D/ProcViewer: Received: +JOIN_SYNC_SESSION OK
com.project.luis.mobipv D/ProcViewer: Received: +GOTO_STEP neemo@overview1 neemo@note_pre_5
com.project.luis.mobipv D/ProcViewer: Received: +TOGGLE_COND neemo@cond_1 1
com.project.luis.mobipv D/ProcViewer: Toggle: neemo@cond_1
com.project.luis.mobipv D/ProcViewer: Received: +TOGGLE_COND neemo@cond_2 1
```

Figure 143: Message exchange from the core and app when the system joins to the collaboration session

The Collaboration icon on the app changes to green colour and the text “SLAVE” is added (see Figure 144).



08:01

SKIN B NOMINAL OPERATIONS SLAVE

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

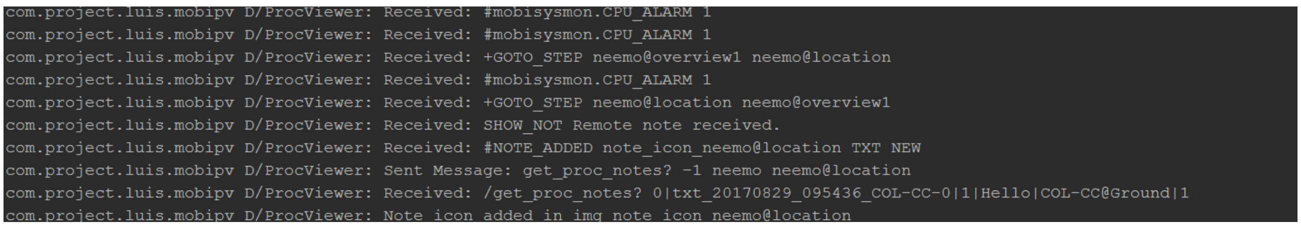
Figure 144: ProcViewer screen while mobiPV is working as slave in a collaboration session

42. Do “NEXT” in Linux laptop system and check that the green marker in the Nexus app also moves and the screen scrolls automatically. Check that pressing on the app does not have any reaction. Add also a note in Linux, verify that the note icon and the note appears in the app.

Although the user presses on next button, in the app, to move the green marker, as the system is working on slave mode, the marker is not going to move. This is because the app only moves the marker when receives a confirmation "+NEXT" from the core, and the core is now just listening to the other core running on Linux.

On the other hand, if next is pressed on the Linux system, the app is going to receive a "+GOTO_STEP" message from its core to move the green marker, that as well as "+NEXT" and "+PREVIOUS" messages it works to move the marker.

After pressing on next and adding the text note "hello" on the Linux mobiPV interface the message exchange is showed on Figure 145.



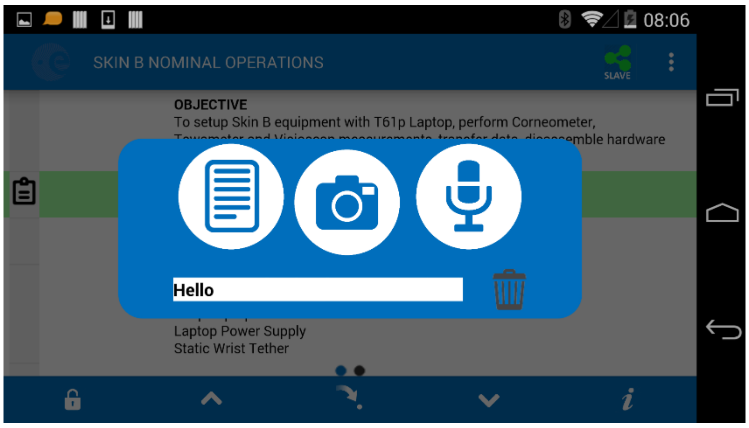
```
com.project.luis.mobipv D/ProcViewer: Received: #mobisysmon.CPU_ALARM 1
com.project.luis.mobipv D/ProcViewer: Received: #mobisysmon.CPU_ALARM 1
com.project.luis.mobipv D/ProcViewer: Received: +GOTO_STEP neemo@overview1 neemo@location
com.project.luis.mobipv D/ProcViewer: Received: #mobisysmon.CPU_ALARM 1
com.project.luis.mobipv D/ProcViewer: Received: +GOTO_STEP neemo@location neemo@overview1
com.project.luis.mobipv D/ProcViewer: Received: SHOW_NOT Remote note received.
com.project.luis.mobipv D/ProcViewer: Received: #NOTE_ADDED note_icon_neemo@location TXT NEW
com.project.luis.mobipv D/ProcViewer: Sent Message: get_proc_notes? -1 neemo neemo@location
com.project.luis.mobipv D/ProcViewer: Received: /get_proc_notes? 0|txt_20170829_095436_COL-CC-0|1|Hello|COL-CC@Ground|1
com.project.luis.mobipv D/ProcViewer: Note icon added in img note icon neemo@location
```

Figure 145: Message exchange between the app and the core. In this case the core asks the app to move the marker and add a note, after the Linux core has asked for it

From Figure 145, the message exchange can be divided in two parts:

- Firstly, the message "+GOTO_STEP …" is read by the app and moves the green marker to "neemo@location" step.
- Secondly, the message "#NOTE_ADDED …" arrives and the app ask for the note of this step with "get_proc_notes? -1 …". The final answer of the system is "/get_proc_notes? -1 …" with the note information of this step. Once the app has the information, it shows the icon on the left of the step.

The final result after the message is shown in Figure 146, where the green marker has advanced and the text note is shown.



OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer,
Tewameter and Visioscan measurements, transfer data, disassemble hardware

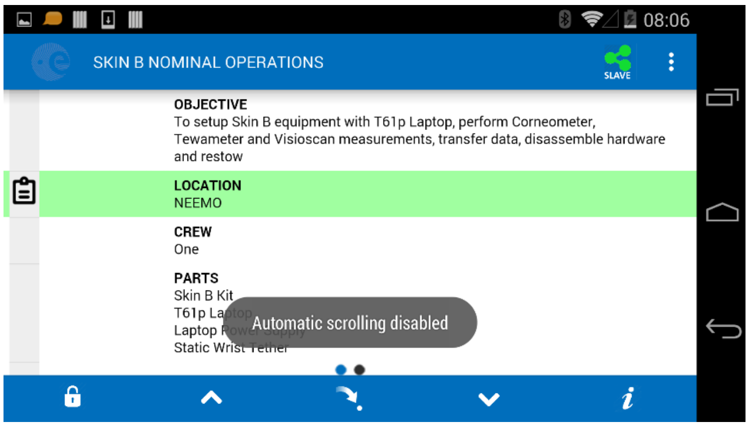
Hello

Laptop Power Supply
Static Wrist Tether

Figure 146: Note Taking Dialog with the text note taken in the Linux core. On the background the green marker is located in the second step.

43. Press now the automatic scrolling disabling button on the app, denoted by . Check that the message "Automatic scrolling disabled" pops up.

Pressing on the button on the left part of the bottom bar disables the automatic scrolling. Imagine that the astronaut wants to read one step and people on ground press next button, so the screen would scroll automatically down. With this option activated the screen of the app remain fixed. See the message that is displayed in Figure 147.



OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer,
Tewameter and Visioscan measurements, transfer data, disassemble hardware
and restow

LOCATION
NEEMO

CREW
One

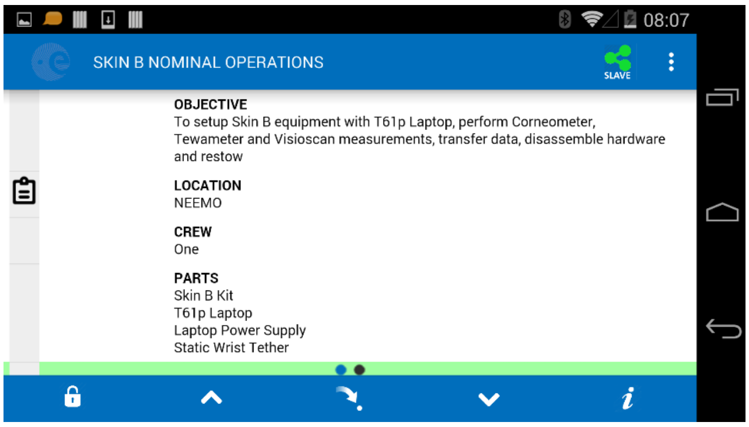
PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Automatic scrolling disabled

Figure 147: ProcViewer screen where the message "Automatic scrolling disabled" has just been plotted after pressing the lock button

44. Do "NEXT" in Linux mobiPV and check that now the screen does not scroll automatically.

Although the green marker in the app will move forward, the screen will be maintained in the same position. See Figure 148 where the master user has pressed three times over next button but the screen has not been automatically scrolled down.



08:07

SKIN B NOMINAL OPERATIONS

SLAVE

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

LOCATION
NEEMO

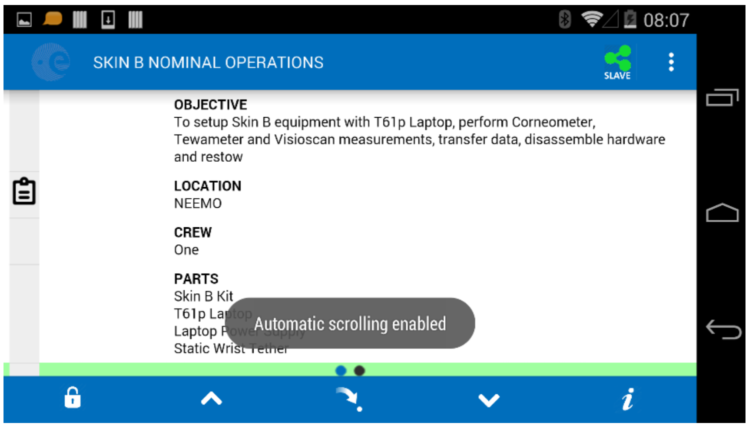
CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Figure 148: ProcViewer screen showing the green marker on the bottom part of the screen while the screen keeps on the same position where it was before Linux user presses next button

45. Enable again the automatic scrolling by pressing again 🔓.

Figure 149 shows the message plotted on the screen after the user presses again the lock button.



08:07

SKIN B NOMINAL OPERATIONS

SLAVE

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

LOCATION
NEEMO

CREW
One

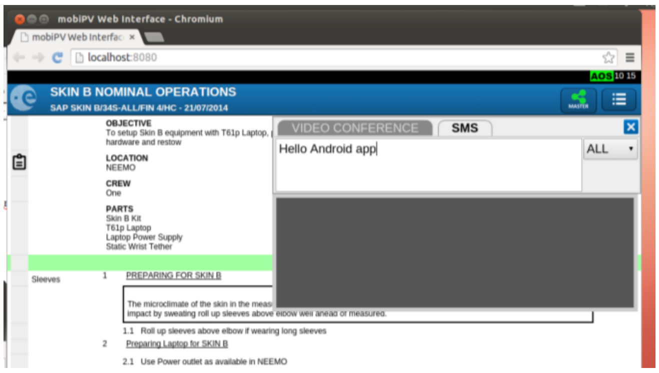
PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Automatic scrolling enabled

Figure 149: ProcViewer screen where the message "Automatic scrolling enabled" has just been plotted after pressing the lock button

46. In Linux mobiPV, press on comms and the SMS tab, write and send a message.

The option comms in the Linux core interface includes a video conference functionality and the SMS, as the video conference has not been implemented yet in the app, the only test is going to be done with the SMS feature. Figure 150 shows the text message written on the mobiPV Linux interface. After pressing enter key the message is sent.


mobiPV Web Interface - Chromium
mobiPV Web Interfac
localhost:8080
AOS 10 15
SKIN B NOMINAL OPERATIONS
SAP SKIN B/34S-ALL/FIN 4/HC - 21/07/2014
MASTER
VIDEO CONFERENCE
SMS
ALL
Sleeves

OBJECTIVE
To setup Skin B equipment with T61p Laptop,
hardware and restow

LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Hello Android app

1 PREPARING FOR SKIN B

The microclimate of the skin in the meas
impact by sweating roll up sleeves above elbow well ahead of measured.

1.1 Roll up sleeves above elbow if wearing long sleeves

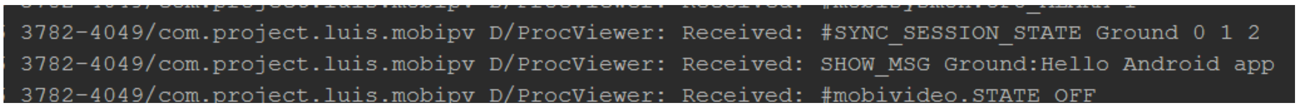
2 Preparing Laptop for SKIN B

2.1 Use Power outlet as available in NEEMO

Figure 150: mobiPV Linux interface showing the SMS feature displayed and the message "Hello Android app" written in it

47. Check that the message is received. Display again the Android tools menu and press on the where the text "New message" should be together with the name of the sender, that is "Ground", and the text message.

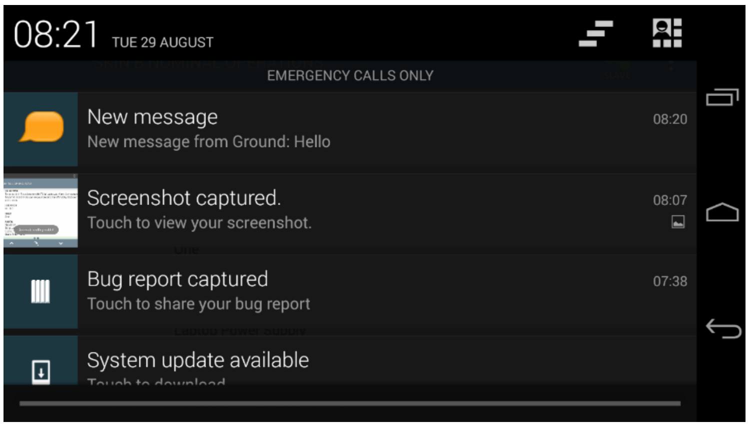
When a new message is received the core sends a message to mobicontrol process where the app is listening, the message is "SHOW_MSG Ground:Hello Android app" (See Figure 151), where Linux core is taking the role of Ground in this simulation. As before, when the first orange bubble appeared on the top of the screen, all the messages that stat with this text "SHOW_MSG Ground", cause that the orange bubble appears.



```
3782-4049/com.project.luis.mobipv D/ProcViewer: Received: #SYNC_SESSION_STATE Ground 0 1 2
3782-4049/com.project.luis.mobipv D/ProcViewer: Received: SHOW_MSG Ground:Hello Android app
3782-4049/com.project.luis.mobipv D/ProcViewer: Received: #mobivideo.STATE OFF
```

Figure 151: Message sent from the core, so the app receives the notification that a chat message has arrived.

If the user displays the Android Top Menu, the content of Figure 152 will be shown. There is a notification of a new message coming from Ground that contains the text "Hello". This information given by the notification is extracted from the received message from the core and adapted to this feature.


08:21 TUE 29 AUGUST

EMERGENCY CALLS ONLY

New message 08:20
New message from Ground: Hello

Screenshot captured. 08:07
Touch to view your screenshot.

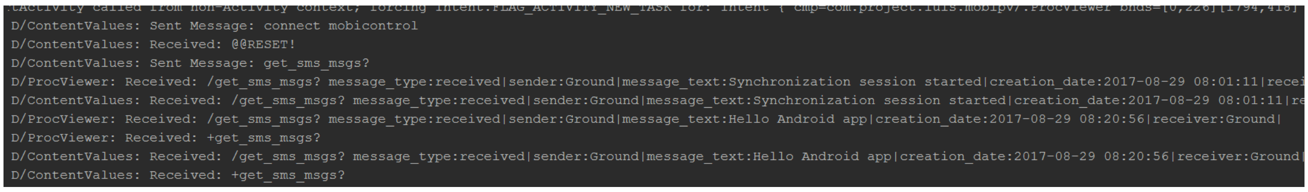
Bug report captured 07:38
Touch to share your bug report

System update available
Touch to download

Figure 152: Android Top Menu showing the notification info.

48. After pressing check that a new Dialog is displayed, scroll to the end of the dialog and check the last message is the same written before.

After the user presses on the notification a new activity of the app is launched, so a new dialog is shown, this dialog implements a typical SMS chat. To build this dialog, the new activity asks the core for information about all the messages that have been sent and received by the users in the system. For doing so the message exchange between the core and the app is shown in Figure 153.

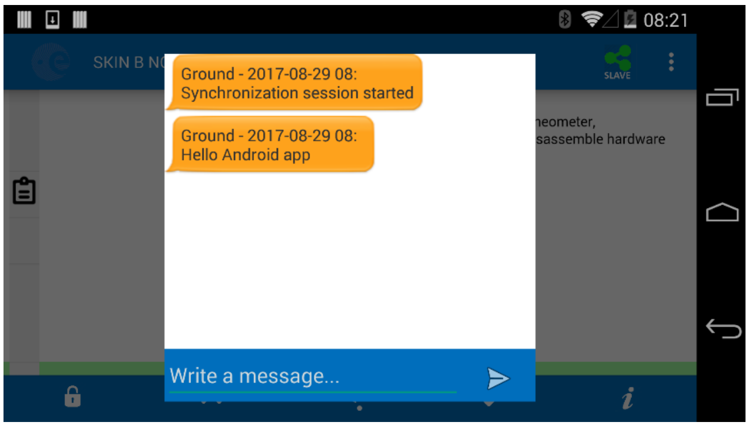


```
:tActivity called from non-Activity context; forcing Intent.FLAG_ACTIVITY_NEW_TASK for: Intent { cmp=com.project.luis.mobipv7.ProcViewer bnds=[0,226][1794,418]
D/ContentValues: Sent Message: connect mobicontrol
D/ContentValues: Received: @@RESET!
D/ContentValues: Sent Message: get_sms_msgs?
D/ProcViewer: Received: /get_sms_msgs? message_type:received|sender:Ground|message_text:Synchronization session started|creation_date:2017-08-29 08:01:11|recei
D/ContentValues: Received: /get_sms_msgs? message_type:received|sender:Ground|message_text:Synchronization session started|creation_date:2017-08-29 08:01:11|re
D/ProcViewer: Received: /get_sms_msgs? message_type:received|sender:Ground|message_text:Hello Android app|creation_date:2017-08-29 08:20:56|receiver:Ground|
D/ProcViewer: Received: +get_sms_msgs?
D/ContentValues: Received: /get_sms_msgs? message_type:received|sender:Ground|message_text:Hello Android app|creation_date:2017-08-29 08:20:56|receiver:Ground|
D/ContentValues: Received: +get_sms_msgs?
```

Figure 153: Message exchange between the core and the app when this last one asks for the messages of the chat

The message exchange can be divided in:

- The app connects to the socket of this new activity by sending the command “connect mobicontrol” and the core replies with “@@RESET”
- The message “get_sms_msgs?” and the different responses with the messages information arrive, (Figure 153). These responses are duplicated because the socket of ProcViewer screen is still operative. But the app checks if the messages are duplicated, so they are not printed duplicated.
- With the end of the message exchange with “+get_sms_msgs?”, the app prints all the messages (see Figure 154).


Ground - 2017-08-29 08:
Synchronization session started

Ground - 2017-08-29 08:
Hello Android app

Figure 154: SMS screen with the messages received from the Linux core

In Figure 154, there are two messages, the first one corresponds to the moment when the collaboration session was stablished and the second one corresponds with the message sent by the Linux core. Both messages contain information from the user, the creation date and time, this information is extracted from the messages shown in Figure 153.

49. **Write a message and press the arrow on the right to send it. Check that the new message appears in a blue bubble and as well in the Linux mobiPV SMS window.**

Clicking on the text "Write a message…" of Figure 154 opens the typical Android keyboard to let the user write some text, the text "Hello mobiPV core running in a Linux virtual machine" is sent (Figure 155)

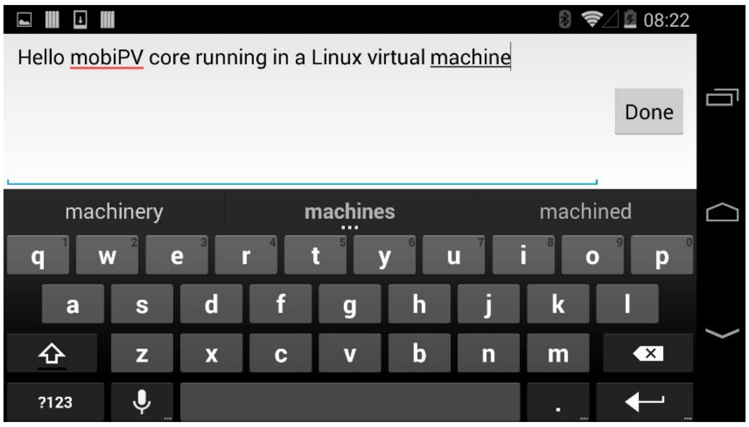
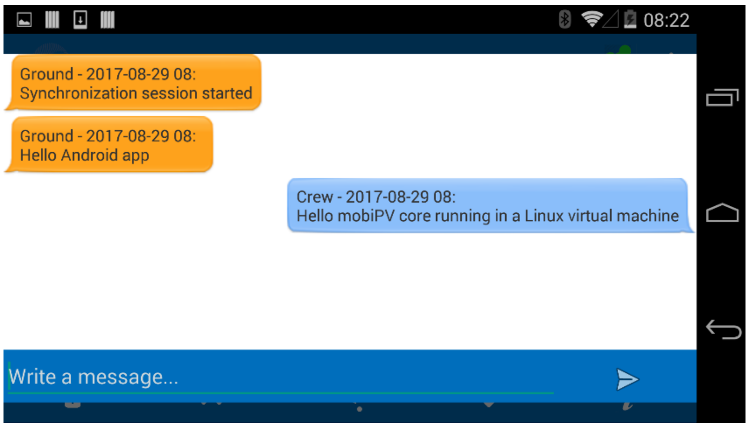

Hello mobiPV core running in a Linux virtual machine

Figure 155: Android keyboard to write the message

The command to send the messages is "COM_txt NOT ALL + text". After the core confirms the reception of the message, the app asks again with "get_sms_msgs?" and all the text messages are again received and stored for plotting them. At the moment of plotting, the app identifies the user and gives a screen position and bubble colour to every user, so blue means that the message was written by the user using the core where the app is running and the orange by other core. See the result in Figure 156 and Figure 157.

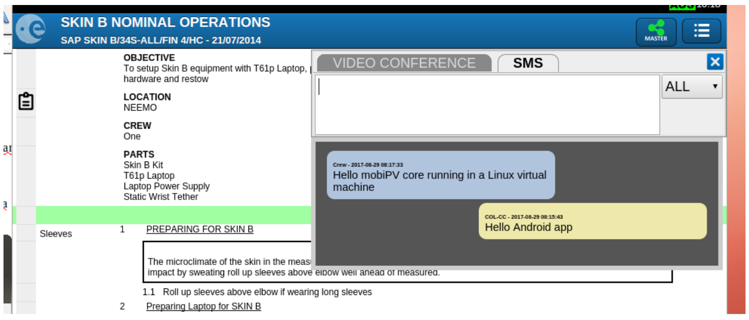


Ground - 2017-08-29 08:
Synchronization session started

Ground - 2017-08-29 08:
Hello Android app

Crew - 2017-08-29 08:
Hello mobiPV core running in a Linux virtual machine

Figure 156: SMS screen in the app with the new message



SKIN B NOMINAL OPERATIONS
SAP SKIN B/34S-ALL/FIN 4/HC - 21/07/2014

OBJECTIVE
To setup Skin B equipment with T61p Laptop,
hardware and restow

LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Sleeves 1 PREPARING FOR SKIN B

The microclimate of the skin in the meas
impact by sweating roll up sleeves above elbow well ahead of measured.

1.1 Roll up sleeves above elbow if wearing long sleeves

2 Preparing Laptop for SKIN B

Crew - 2017-08-29 08:17:33
Hello mobiPV core running in a Linux virtual machine

COL-CC - 2017-08-29 08:15:43
Hello Android app

Figure 157: SMS window in Linux core interface with the new message

50. Press in the app and press on "STOP collaboration". Open the Side Menu in the app and press on Logout.

As now the collaboration session is established and the smartphone core is in it, the app reads a state message sent from the core and builds the screen differently as shown in Figure 158. The app gives the option of requesting the Master role of the collaboration, so the Linux mobiPV would be now following the activity on the smartphone or abandoning the session, which is the option selected.

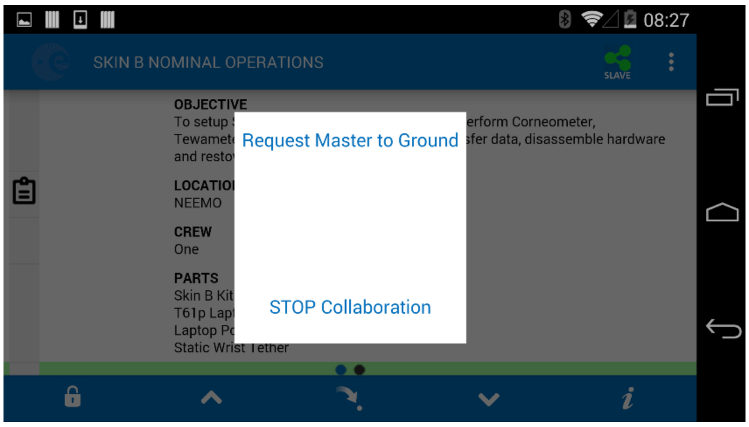


Figure 158: Collaboration Dialog when the core of the Nexus 5 belongs to it as slave

After abandoning the session, the collaboration icon comes back to white colour, the last thing to be done is press on logout on the Side Menu and Login screen is displayed (see Figure 159).

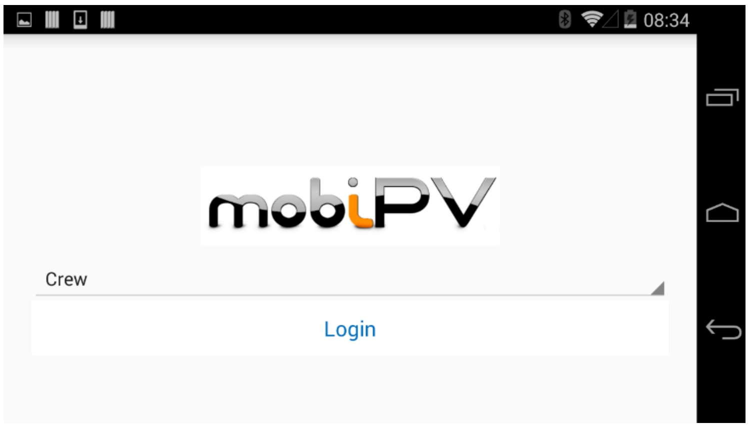


Figure 159: Login screen at the end of the test

5 Discussion of the results

The feedback the test has given to the project has been important during the approximately 15 minutes it has lasted, since there have been no interruptions and almost all the implemented features have been tested.

The message exchange using the TCP socket between the core and the app have proved to be a reliable way of interacting with it, that does not fail if it is correctly coordinated with the rest of Java activities and methods. This coordination between the flowing messages and how the user interface applies the changes and animations is probably the most important achievement of the project.

Having the incoming message flow in the secondary thread, and from this one call other methods in the main thread when the message activates the listener or event, is probably the best approach to have meanwhile the user interacting with the app in the main thread. So while the user is working, pressing buttons or calling methods that speak with the core in the main thread, the core is talking to the background of the app and from there activating another methods and launching events. With this way of working the app has demonstrated to be reliable, that is an important feature for a system that is going to work in a space ship as first approach.

Another achieved target has been the almost full integration of the features, and improvement of them, that the old mobiPV app included, except the video streaming option to do multiconference. Some of these featured related with the views are presented during the following lines. In Figures 160-168, the image on the left corresponds to the old app via a WebView and the image on the right corresponds to the new Android app developed in this project.

- The login screen appears to be more harmonic and integrated in the new app than in the old app version.

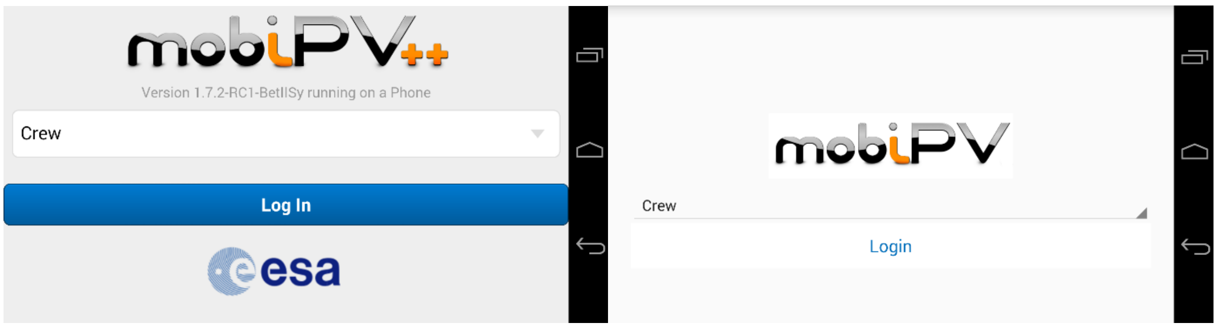


Figure 160: Login screen comparison

- Main Menu screen implements images on the new app while only the text after analysing the JSON file in the old one.

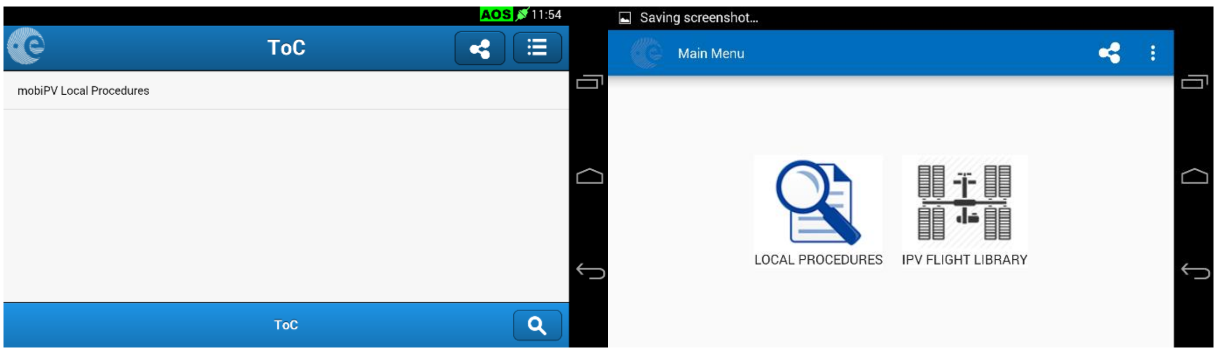


Figure 161: Main Menu comparison

- The procedure list screen is more clear in the new app. Also for navigating in the old app the bottom bar is needed and the Android return button does not work. The navigation in the new app is then easier because as it is native Android code, the app can make use of this useful button (see Figure 162).

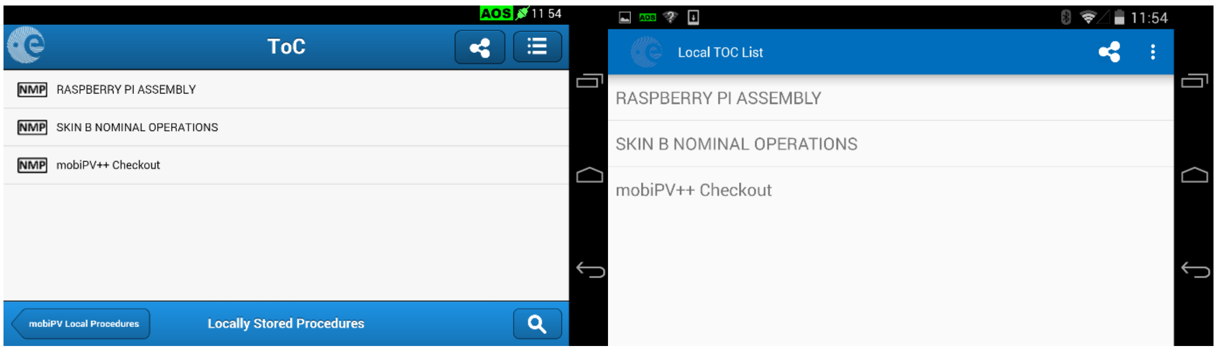
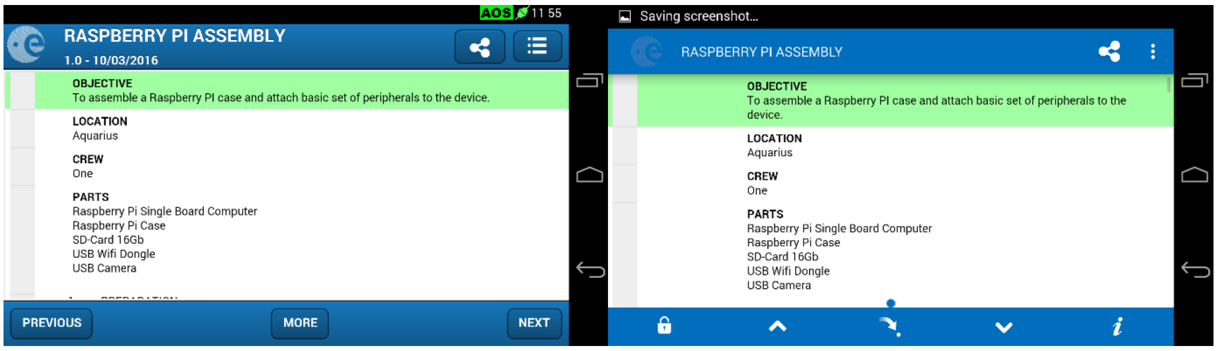


Figure 162: Procedure list comparison. The navigation through the procedure tree in the old app has to be done using the bottom bar while in the new app is done with the return button

- The ProcViewer screens are similar, the main difference is found in the tools and their position on the bars. While the new app brings all the tools to the screen (see Figure 163), in the old app the tools menu has to be displayed and is not as comfortable to manage (see Figure 164).



OBJECTIVE
To assemble a Raspberry PI case and attach basic set of peripherals to the device.
LOCATION
Aquarius
CREW
One
PARTS
Raspberry Pi Single Board Computer
Raspberry Pi Case
SD-Card 16Gb
USB Wifi Dongle
USB Camera

Figure 163: ProcViewer screen comparison

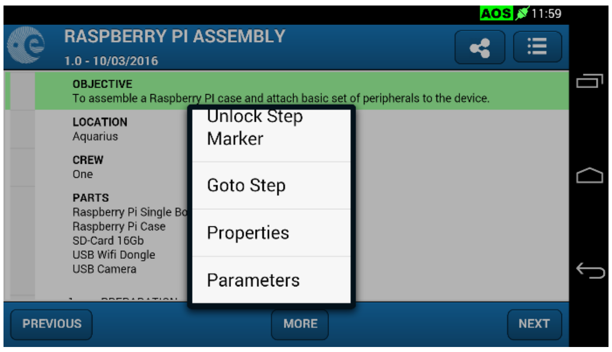
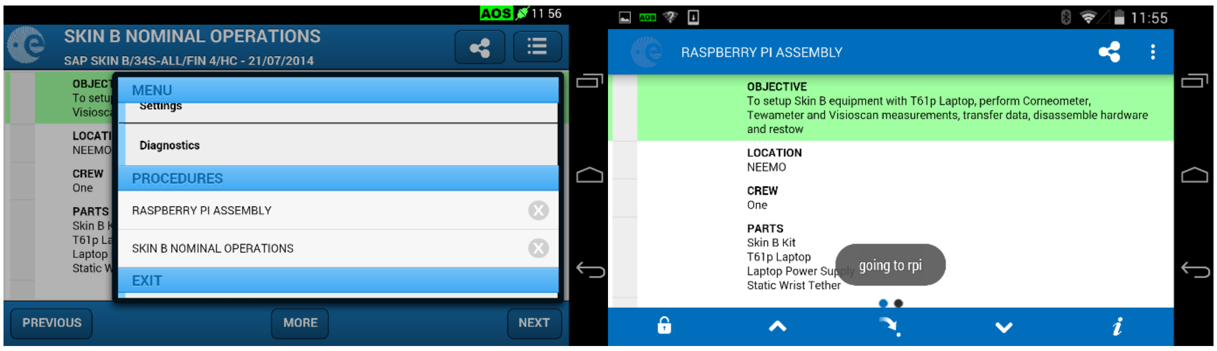


Figure 164: Other tools menu in the old app

- The navigation between one procedure and other is also different, while in the old app the user has to display the Menu and select a different procedure, the app, that also includes this feature in the Side Manu, can navigate through procedures by swiping left and right on the screen. See Figure 165, where on the new app the procedure is being changes by swiping.



OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

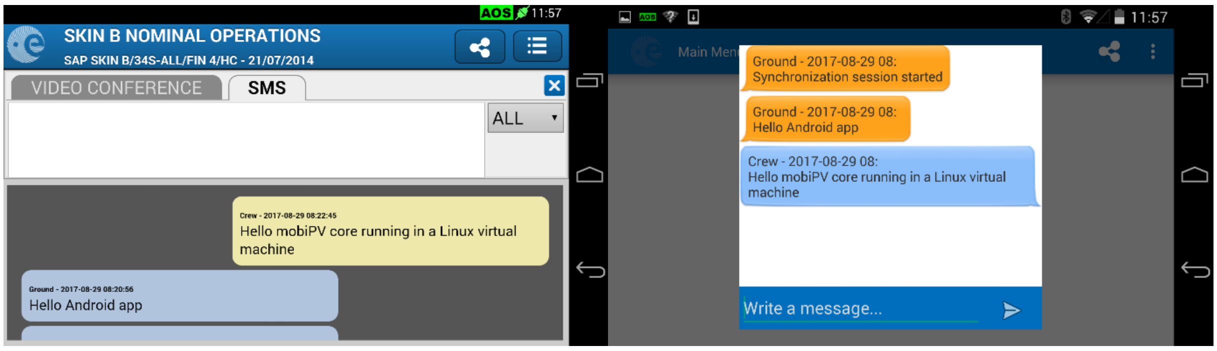
LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Figure 165: Procedure navigation through open procedures comparison

- The Chat Dialog in the new app is more similar to standard and well known chat apps available for Android, on the other hand the chat on the old app is not so intuitive (see Figure 166).


AOS 11:57
SKIN B NOMINAL OPERATIONS
SAP SKIN B/34S-ALL/FIN 4/HC - 21/07/2014
VIDEO CONFERENCE
SMS
ALL
11:57
Main Menu
Write a message...

Crew - 2017-08-29 08:22:45
Hello mobiPV core running in a Linux virtual machine

Ground - 2017-08-29 08:20:56
Hello Android app

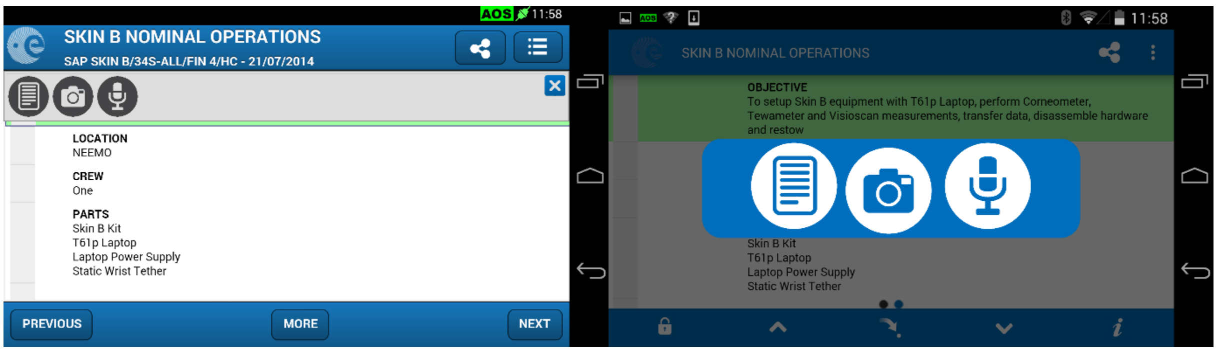
Ground - 2017-08-29 08:
Synchronization session started

Ground - 2017-08-29 08:
Hello Android app

Crew - 2017-08-29 08:
Hello mobiPV core running in a Linux virtual machine

Figure 166: Chat dialog comparison

- The Note Taking Dialog is more aligned and intuitive in the new app (see Figure 167).


AOS 11:58
SKIN B NOMINAL OPERATIONS
SAP SKIN B/34S-ALL/FIN 4/HC - 21/07/2014
PREVIOUS
MORE
NEXT
11:58
SKIN B NOMINAL OPERATIONS

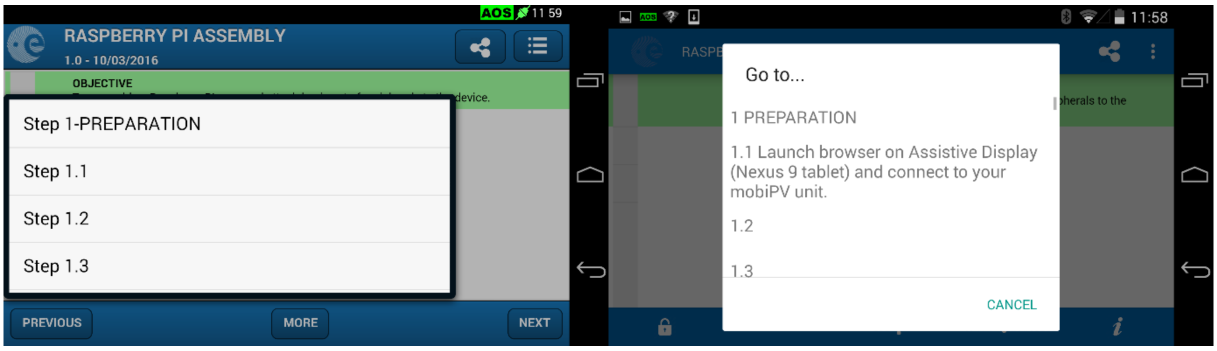
LOCATION
NEEMO
CREW
One
PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Figure 167: Note taking dialog comparison

- The dialogs in Android native code are managed in an easier way, while in the web app, they can be shifted to one side and are not so well integrated with the rest of the screen (see Figure 168).



Step 1-PREPARATION
Step 1.1
Step 1.2
Step 1.3

Go to...
1 PREPARATION
1.1 Launch browser on Assistive Display (Nexus 9 tablet) and connect to your mobiPV unit.
1.2
1.3

Figure 168: Dialog comparison

One of the main differences that cannot be valued by just watching the different screens is how the systems interact between them.

- The old Android app was just a WebView loading the URL where the core was launching the WebApp. The communication methods of the app or web browser that the app was implementing was made with a WebSocket (see again Figure 14) and it was mandatory the use of an additional component, mobiWebGW, that works as a gateway element since the communication with mobicontrol element core must be done through traditional TCP sockets.
- The new Android app, written in native code, communicates with the core using a TCP sockets. This time and coming back again to Figure 14, the app uses the same kind of connection to mobicontrol than the mobiPV component mobiWebGW and avoids the creation and use of WebSockets and then the use of the mobiWebGW gateway.

A WebSocket is designed to run just from browsers and runs over TCP/IP. WebSockets implement a permanent connection to the server. On the other hand, the TCP socket is more powerful and generic, it also runs over TCP/IP but is not restricted to the browser.

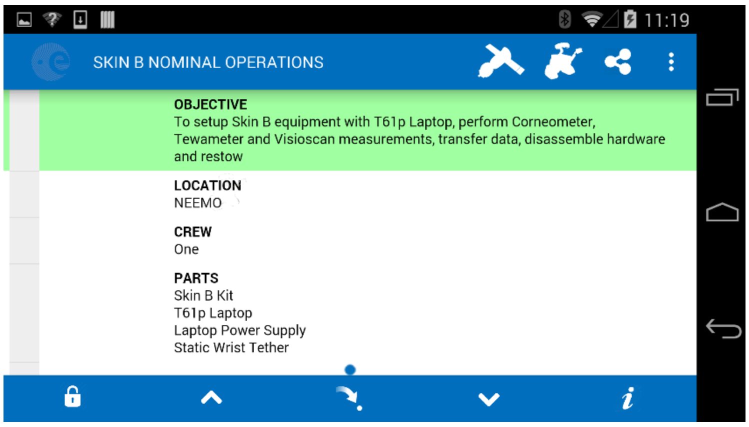
6 Conclusiones y líneas futuras

Resumiendo los principales aspectos del resultado final, ha habido muchas mejoras, tener una aplicación Android escrita en código nativo Android da al usuario una mejor experiencia al trabajar con procedimientos en dispositivos como teléfonos móviles, que tienen requerimientos especiales, como la pantalla pequeña o el uso de aplicaciones Ontouch instaladas en ellos.

El código nativo en Android para mobiPV hace el mejor uso de las herramientas que Android ofrece y consigue una mayor eficiencia al tener transiciones más rápidas entre pantallas o evitando el uso de WebSockets, implementados en la aplicación antigua, mientras se mantienen las mismas herramientas y características de la original. Este exitoso salto del proyecto mobiPV, a Android, prueba y abre la puerta a que más y más tecnologías espaciales pueden ser adaptadas a esta plataforma abierta.

Como todos los proyectos, el final puede no llegar nunca y muchos servicios más pueden ser añadidos a esta app, por ejemplo:

- Añadir la única característica que no se ha podido añadir debido a problemas de tiempo. Esta característica es el streaming de vídeo para soportar la multiconferencia entre usuarios en una sesión de colaboración.
- Reemplazar completamente la aplicación antigua, esto es ser capaz de cargar librerías nativas C/C++ de los componentes multimedia en la aplicación Android.
- Adición de más propiedades visuales para hacer la app más intuitiva y mejorar la experiencia final de usuario.
- Llevar a cabo un test real en la ISS con un usuario real como un astronauta.
- Combinar esta solución Android con otros proyectos espaciales para facilitar la actividad "sobre el campo" de exploradores espaciales.
  Como toda la actividad y trabajo de los astronautas se basa en seguir procedimientos, por ejemplo, la app podría ser usada para, al mismo tiempo que el astronauta sigue un procedimiento, conducir un rover en la superficie lunar. De igual modo, se puede usar para ayudar al astronauta mientras pilota una nave espacial si algunos parámetros del vehículo quieren ser chequeados. Quién sabe si el aspecto de la pantalla de ProcViewer en algún tiempo es similar a la mostrada en la Figure 169, donde en la parte superior incluye una nave espacial y un icono de un róver para interactuar con estas tecnologías.


11:19

SKIN B NOMINAL OPERATIONS

OBJECTIVE
To setup Skin B equipment with T61p Laptop, perform Corneometer, Tewameter and Visioscan measurements, transfer data, disassemble hardware and restow

LOCATION
NEEMO

CREW
One

PARTS
Skin B Kit
T61p Laptop
Laptop Power Supply
Static Wrist Tether

Figure 169: Apariencia futurista de ProcViewer

Las posibilidades con Android son infinitas y de igual modo la cantidad de soluciones y mejoras que esta aplicación tiene, ¿hay límites para seguir mejorando mobiPV en Android? ¿existe un techo para las tecnologías espaciales en la plataforma Android?

REFERENCES

[1] F. A. W. M. M. O. D. Boyd A, “mobiPV: A new, wearable real-time collaboration software for Astronauts using mobile computing solutions,” in *SpaceOp*, Daejeon, Korea, 2016.

[2] NASA, “About NEEMO (NASA Extreme Environment Mission Operations),” 31 July 2015. [Online]. Available: https://www.nasa.gov/mission_pages/NEEMO/about_neemo.html.

[3] W. M. P. A. B. R. G. N. H. U. Konig K, “Multiphoton tomography of astronauts. Multiphoton Microscopy in the Biomedical Sciences XV,” 5 March 2015.

[4] E. S. Agency, “ESA,” 16 September 2014. [Online]. Available: http://www.esa.int/spaceinimages/Images/2014/09/Andreas_and_mobiPV. [Accessed 13 July 2017].

[5] E. S. Agency, “Youtube,” 12 September 2014. [Online]. Available: https://www.youtube.com/watch?v=UurNYX-Mk-g. [Accessed 13 July 201].

[6] E. S. Agency, “ESA,” 22 July 2015. [Online]. Available: http://blogs.esa.int/iriss/2015/07/22/steps-for-mobipv/. [Accessed 13 July 2017].

[7] Android Developers, “WebView,” [Online]. Available: https://developer.android.com/guide/webapps/webview.html. [Accessed 30 July 2017].

[8] D. M. Oliveira, “mobiPV++ System Design Document,” estec, Noordwijk, 2014.

[9] D. M. Oliveira, “mobiPV++ User Manual,” estec, Noordwijk, 2015.

[10] ASUS, “Laptop ROG-G73SW,” [Online]. Available: https://www.asus.com/Laptops/ROG-G73SW/. [Accessed 13 July 2017].

[11] Google, “Nexus,” [Online]. Available: https://www.google.com/nexus/. [Accessed 20 April 2017].

[12] Microsoft, “Microsoft LifeCam Cinema,” [Online]. Available: https://www.microsoft.com/accessories/es-es/products/webcams/lifecam-cinema/h5d-00002. [Accessed 2017 July 13].

[13] Debian, “What is GNU/Linux?,” [Online]. Available: https://www.debian.org/releases/stable/i386/ch01s02.html.en. [Accessed 2017 July 13].

[14] Ubuntu, “The Ubuntu story,” [Online]. Available: https://www.ubuntu.com/about/about-ubuntu. [Accessed 2017 July 13].

[15] Google, “Android Studio,” [Online]. Available: https://developer.android.com/studio/releases/index.html. [Accessed 2017 July 13].

[16] J. T. Gironés, El gran libro de Android, Barcelona: marcombo, 2015.

[17] SunilOS, “JAVA Variables and Operators,” 28 Nov 2015. [Online]. Available: https://www.slideshare.net/sunilos/java-variables-and-operators-55602182. [Accessed 2017 Aug 2017].

[18] Oracle, “Online version VirtualBox manual,” [Online]. Available: https://www.virtualbox.org/manual/ch01.html. [Accessed 14 July 2017].

[19] Ubuntu, “Virtual networking,” [Online]. Available: https://help.ubuntu.com/lts/serverguide/network-configuration.html. [Accessed 14 July 2017].

[20] A. Developers, “Run Apps on a Hardware Device,” [Online]. Available: https://developer.android.com/studio/run/device.html. [Accessed 15 July 2017].

[21] WikiHow, “How to Create a WiFi Hotspot Using the Command Prompt,” [Online]. Available: http://www.wikihow.com/Create-a-WiFi-Hotspot-Using-the-Command-Prompt. [Accessed 15 July 2017].

[22] G. Developers, “https://developers.google.com/android/images,” [Online]. Available:

https://developers.google.com/android/images. [Accessed 22 July 2017].
[23] D. M. Oliveira, “mobiPV ESTEC Operations Manual,” Noordwijk, 2015.
[24] Android Developers, “Intent,” [Online]. Available: https://developer.android.com/reference/android/content/Intent.html. [Accessed 26 July 2017].
[25] Android Developers, “View,” [Online]. Available: https://developer.android.com/reference/android/view/View.html. [Accessed 29 July 2017].
[26] A. Developers, “Button,” [Online]. Available: https://developer.android.com/reference/android/widget/Button.html. [Accessed 29 July 2017].
[27] Skytek, “IPV,” [Online]. Available: http://www.skytek.com/our-porfolio/ipv/. [Accessed 30 July 2017].
[28] W3schools, “CSS,” [Online]. Available: https://www.w3schools.com/css/css_howto.asp. [Accessed 30 July 2017].
[29] Android Developers, “String,” [Online]. Available: https://developer.android.com/reference/java/lang/String.html. [Accessed 3 August 2017].
[30] A. Developers, “Toasts,” [Online]. Available: https://developer.android.com/guide/topics/ui/notifiers/toasts.html. [Accessed 2017 July 23].

Appendix 1

mobiPV app test

Description

The purpose of this test is to verify mobiPV app and its features. From the beginning until the end the app should be able to run the whole test without failures.

For this test is necessary to have another mobiPV core running on a device connected to the same LAN, so the cooperation features can also be tested. For the performing of the test is needed to have a laptop with a WLAN opened where the Nexus 5 is connected and a Linus system connected to the same network where mobiPV is running.

Procedure

1. Open mobiPV app on the Desktop denoted by the icon
2. SplashScreen is shown for some seconds. Check that some Super User permissions are given to the app. Then the screen changes into Login Screen.
3. Press on Login with Crew user selected, the screen will change into MainMenu Screen
4. Press on the three dots on the top right corner of the Action Menu denoted by ⋮ on white to open the Side Menu.
5. Press on Settings option, Settings screen will be opened.
6. Check that all the options and the three button on the bottom are added. Check as well that, for example, the option "Video camera device" is blocked and cannot be changed.
7. Press Return Android button to come back to the Main Menu Screen. Denoted by ⮌ .
8. Press again on the Side Menu button and press now on Logout. The Login Screen will be loaded.
9. In the user selector, choose special user "sysop", press on "Login", check that a Dialog appears asking for the password.
10. Write "superuser" and press on "Confirm". Check that a message appears with the text "Wrong password, try again"
11. Press again on Login and write "supersecret", press on "Confirm" and check that the MainMenu screen is loaded.
12. Open the Side Menu and select again Settings, check now that the option "Video camera device" is available to be edited.
13. Go back to Main Menu and open the Side Menu, press on "Diagnostics". Diagnostics Screen will be opened.
14. Run a Diagnostics by pressing the button on the left bottom and check that some results appear and there are three kind of messages: INFO, denoted by ℹ , OK, denoted by ✓ , and FAIL, denoted by ✖
15. Go back to Main Menu Screen and press on "Local Procedures", the screen LocalTOCList will be loaded. Check that there are three procedures and press on "SKIN B NOMINAL OPERATIONS". The screen ProcViewer will be opened.
16. Check that the green marker is located at the beginning of the procedure, press on next button, denoted by ⌄ , on the bottom menu twice and check that the green marker moves forward

two positions. Press now on the previous button, denoted by , located in the same menu once and check that the green marker goes backwards one position. Check that at the end of this step the green marker is on the second position.

17. Double press on the fourth step, titled “PARTS” and check that the green marker jumps to this step.
18. Press on info icon, denoted by , although this step does not have any additional information, check that the message “No info available” is shown.
19. Press on the centre icon, denoted by and check that a Dialog appears with a list of steps. Look for step “3 Laptop Activation” and press on it. Check that the green marker jumps to this step.
20. Press on the toggle box, ⊞, on step 3.1 to minimize it.
21. Use again to jump to step 4.9 and press on the figure, check that an image is expanded.
22. Open now the Side Menu using ⋮ button. Select TOC to come back to the MainMenu Screen, select “Local Procedures”, and open “RASPBERRY PI ASSEMBLY”.
23. When the procedure is loaded, select and jump to step 3.3, press the video image and that after pressing on play it starts playing. Pause it.
24. Check that on the lower part of the screen just above the lower bar there are two dots, one blue and one black.
25. Swipe left and check that the procedure “SKIN B NOMINAL OPERATIONS” is loaded (check the title on the action bar on the top left part of the screen). Check also that the blue dot is now a different one.
26. In the actual procedure and step, press on the left part of the screen, the area denoted by a darker colour. Check that the note taking dialog is shown.
27. Press on the text note icon, denoted by , and write “Hi” on the box, press then on Save. Check that the note icon appears on the left area of the procedure.
28. Press now over the note icon and check the text is “Hi”. Press away of the dialog to close it.
29. Press on the left part of the screen of the next step and open again the note taking dialog.
30. Press on photo/video note taking, denoted by , and check that a new dialog appears showing what the camera is recording.

31. Press on the video recording button, denoted by , and check that the button changes to , then press again this last button. Check that the note icon appears on the left area of the procedure.
32. Press on the note icon and press "Open video". Watch the video and check that the content is correct.
33. Press Android back button, and next button twice
34. Open again the note taking dialog and press again on photo/video button, this time press on . Check that the note icon appears on the left area of the procedure.
35. Press on the note icon and press "Open photo". Watch the photo and check that the content is correct.
36. Press away of the dialog to close it and open the note taking dialog of the next step and select audio recording, denoted by . Check that the icon changes to and the message "Recording…" is shown. Say something and press stop button, check that the dialog closes and the message "Stopping…" is displayed.
37. Open again note taking dialog of the same step and listen to the audio file by pressing on play. After listening to it press on the trash can to delete it. Check that the note icon on the left part disappears.
38. Display now the Android tools menu by swiping down from the top part of the screen. Press on the top right icon and press again on WiFi menu. On the new screen press on the WLAN where the Nexus 5 is connected, that should be the same created by the laptop and write down the IP address.
39. On the other mobiPV core go to Settings and on "Sync Server" write "0" and on "Sync Server IP" write the IP address of the Nexus 5. Restart the mobiPV system.
40. On the same Linux mobiPV, open "SKIN B NOMINAL OPERATIONS" procedure and press now Sync Session on the action bar on the top of the screen denoted by . Check that a new dialog pops and press on "Start Collaboration". Check that on the app one message has just arrived, denoted by , on the top left part of the screen.
41. Press in the app and press on join collaboration. Check that the green marker jumps to the same position where it is in the Linux mobiPV.

42. Do “NEXT” in Linux system and check that the green marker in the Nexus app also moves and the screen scrolls automatically. Check that pressing on the app does not have any reaction.
43. Press now the automatic scrolling disabling button on the app, denoted by . Check that the message “Automatic scrolling disabled” pops up.
44. Do “NEXT” in Linux mobiPV and check that now the screen does not scroll automatically.
45. Enable again the automatic scrolling by pressing again .
46. In Linux mobiPV, press on comms and the SMS tab, write and send a message.
47. Check that the message is received. Display again the Android tools menu and press on the where the text “New message” should be together with the name of the sender, that is “Ground”, and the text message.
48. After pressing check that a new Dialog is displayed, scroll to the end of the dialog and check the last message is the same written before.
49. Write a message and press the arrow on the right to send it. Check that the new message appears in a blue bubble and as well in the Linux mobiPV SMS window.
50. Open the Side Menu in the app and press on Logout.