



# Exor Linux BSP Development User Manual

---

UM0013 (v2.0) – 21 Mar 2021

User Manual

---

## Overview

This document will guide the user through the configuration and use of the Exor Linux BSP for development purposes.

---

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved. Technical data subject to change. All trademarks and trade names appearing in this document are property of their respective owners. Copyright © 2015 Exor International SpA-Verona-Italy, All Rights Reserved. **Disclaimer** Exor International SpA is providing this design, code, or information "as is." basis, without warranty of any kind, either expressed or implied, including, without limitation, warranties that the covered code is free of defects, merchantable, fit for a particular purpose or non-infringing. Each party bears the entire risk as to the quality and performance of the original code, upgraded code, and modifications, to the extent originating with and provided by such party. Should any covered code prove defective in any respect, you assume the cost of any resulting damages, necessary servicing, repair or correction. This disclaimer of warranty constitutes an essential part of this license. No use of any covered code is authorized hereunder except subject to this disclaimer.

---

## Table of Contents

1.	Introduction.....	5
2.	Boot sequence.....	6
3.	System Settings.....	8
3.1	Sections overview.....	9
3.2	Management.....	10
3.2.1	Making a component backup .....	11
3.2.2	Updating a component .....	11
3.3	Services.....	12
3.3.1	SSH Service .....	12
3.3.2	Avahi Daemon.....	12
3.3.3	Autorun scripts from external storage .....	12
4	Application Launcher .....	13
4.1	Kiosk mode.....	14
4.2	Creating custom packages .....	14
5	BSP structure .....	16
6	Using the serial port.....	19
6.1	Serial port pinout .....	19
6.1.1	Pinout for eSMART and eTOP6xxL series.....	19
6.1.2	Pinout for eXWare and eX7xx series .....	20
6.2	Configuring the serial port.....	20
6.2.1	Changing the operating mode .....	20
6.2.2	Configuring the port.....	21
7	USB tools .....	23
7.1	USB Updater .....	23
7.1.1	BSP update .....	23
7.1.2	BSP backup .....	24
7.2	Application Installer.....	24
8	Compiling applications for the target.....	26
8.1	Using QtCreator IDE.....	26
8.1.1	Build configuration.....	27
8.1.2	Application deploy .....	32
9	Other useful information.....	36

9.1 Device discovery.....	36
9.2 Enabling the serial console .....	36
9.3 Enabling root SSH login.....	37
9.4 Recovering from a damaged MainOS .....	38
10 Source code .....	39
10.1 Building the Linux kernel.....	39
10.1.1 Source code and configuration .....	39
10.1.2 Deploy.....	39
10.2 Building the U-Boot bootloader .....	40
10.3 BSP source code .....	40

## Version History

Version	Release date	Changes
3.0	21/03/2022	First Version based on 2.1.x and 3.1.x BSPs

## 1. Introduction

The goal of this document is to describe the relevant software components of the 1.3.x and 2.0.x BSP installed in the Linux based Exor products, guide the user through their use and help in getting started with custom software development on such targets. Supported products by this manual include:

- eSMART series panels
- eTOP6xxL series panels
- eX7xx/ex7xxM series panels
- JSmart/JSmartxxM series panels
- eXware series
- X10
- PLCM07

BSPs for each of the above targets are available for download here:

<https://www.exorint.com/en/jmobile-download>

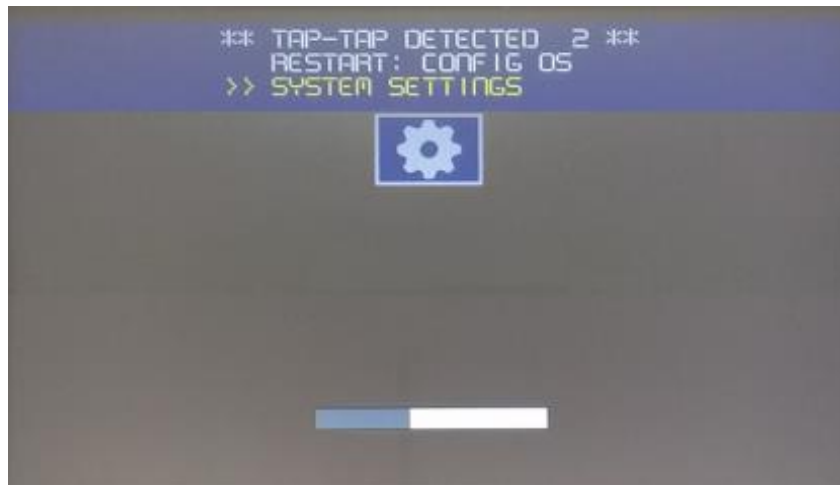
For datasheets, hardware specifications and other technical information, please refer to the hardware section on our website:

<https://www.exorint.com/en/hardware>

## 2. Boot sequence

At power on a progress bar and an optional splash image is shown on the screen. A custom splash image can be set from System Settings.

During this phase it's possible to interrupt the boot sequence to access the "tap-tap menu", to do so tap with the finger on the screen multiple times until a menu is shown on top of the screen:



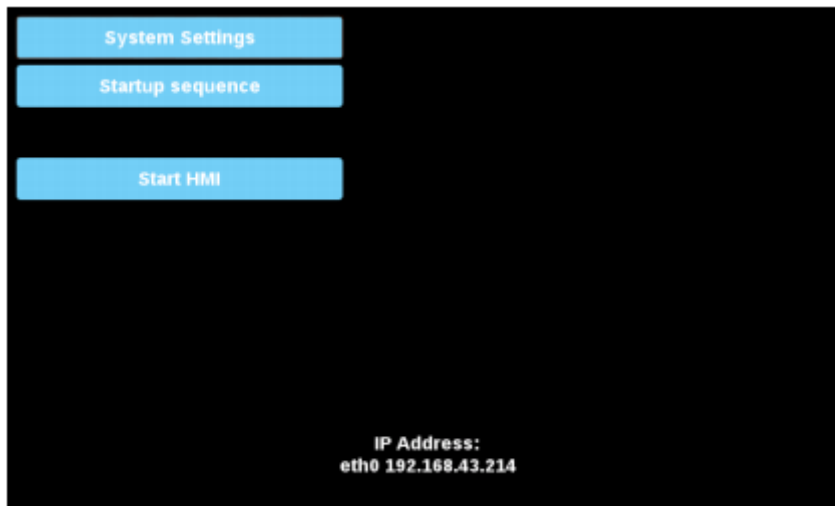
By keeping now the finger on the screen the "RESTART: CONFIG OS" option will remain selected and after 5 seconds the panel will reboot in the ConfigOS recovery mode ( see ).

If the screen is left untouched instead, the panel will proceed to the System Settings sub-menu:



Keeping the finger on the screen the user have the possibility to repeat the touchscreen calibration.

In either way the boot sequence is now interrupted meaning that no installed application will now start. The following interface is shown instead:



The above screen is also shown by default when there are no applications to start. These are the possible actions that can be taken from here:

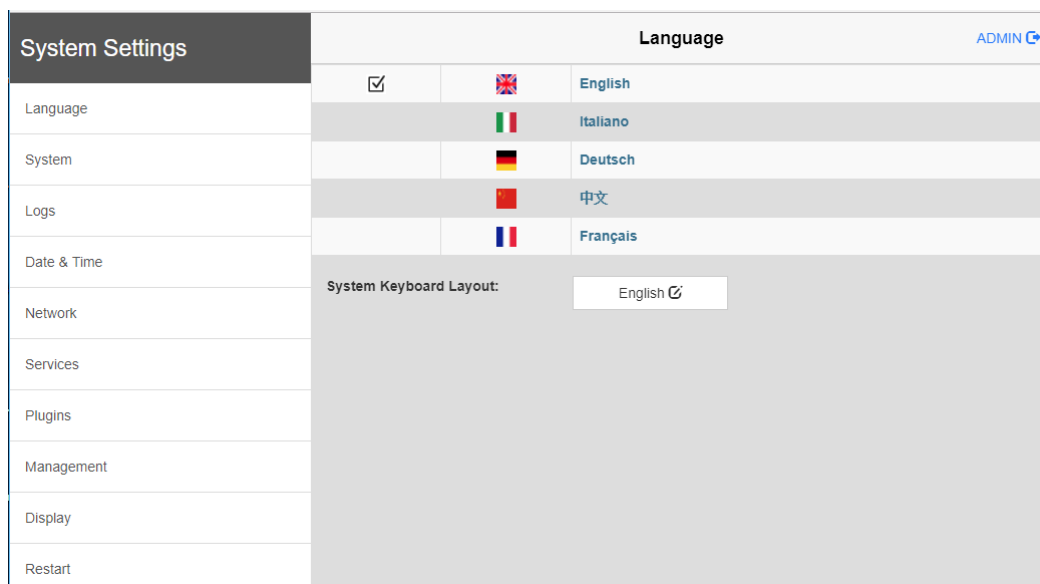
- **System Settings:** Access the panel settings. From here it's possible to read device information and configure and update the BSP ( see [3. System Settings](#) ).
- **Startup Sequence:** Install, uninstall, update and manage installed applications ( see [4. Application Launcher](#) )
- **Start HMI:** Resume the boot sequence by starting all the enabled applications. This option is not shown if there are no applications to start. The kiosk mode will however remain disabled ( see [4.1 kiosk mode](#) )

At the bottom the IP address of the connected network interfaces are shown

### 3. System Settings

The panel System Settings interface can be accessed in two different ways:

- Locally by interrupting the boot sequence at power on and then pressing the “System Settings” button.
- Remotely from a web browser.

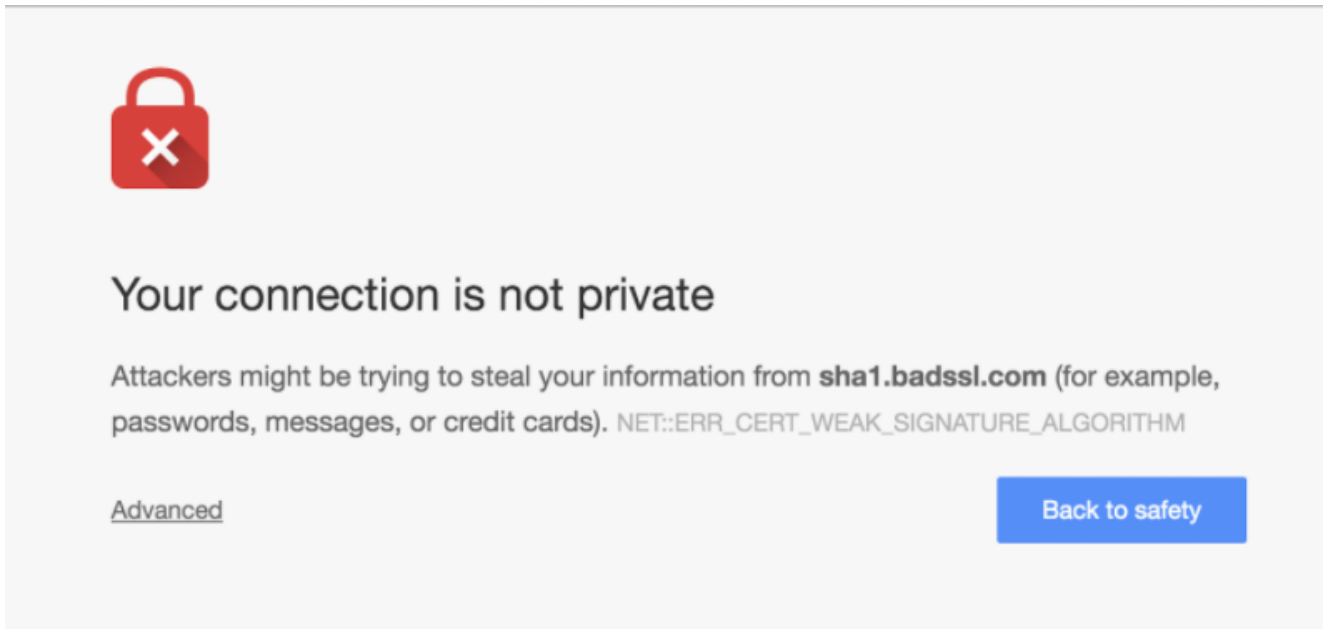


To access System Settings from the browser make sure the panel is properly connected to your network and reachable, then browse to this address:

```
https://<device-IP>/system_settings
```

A certificate sign warning may appear, on the Chrome browser it would look like this:





In this case click on “Advanced” and then “Proceed” to continue.

Authentication is required, the default username and password for the administrator is admin:admin.

### 3.1 Sections overview

Following is a brief description of each section that can be seen listed in the menu on the left:

- **Language:** Localization settings. The interface language and system keyboard layout and can be selected here.
- **System:** Here some general system information is shown like Linux kernel version, uptime and RAM usage.
- **Logs:** Log files can be set here to be persistent between reboots, and can be exported inside a single .zip archive file.
- **Date&Time:** Set system time and timezone. A NTP service can also be enabled for automatic time synchronization.
- **Network:** Set hostname, DNS and network interfaces configuration. By default DHCP is used, here this can be disabled by setting a static IPs.
- **Services:** Enable, disable and configure services like SSH and VNC server.
- **Plugins:** Lists the hardware plugins currently connected to the device. Shown only if plugins are supported.

- **Management:** Manage BSP components, updates and backups are possible.
- **Display:** Set screen orientation, backlight brightness and backlight timeout.
- **Restart:** Reboot the system. By choosing to reboot in ConfigOS the device will enter recovery mode.
- **Authentication:** Manage device users and SSL certificates

## 3.2 Management

Under the Management section the list of the BSP components is shown:

The screenshot shows a web interface titled "Management" with a "MENU" button on the left and an "ADMIN" button on the right. The main content area is divided into several sections:

- Config OS**: A section header.
- Main OS**: A section containing a table of details:
 

Type	ext4
Version	UN65HSXXM01002006
Date	2018-04-02T22:00:00.000Z
327 Mb / 478 Mb used	

 Below the table are three buttons: "Get" with a download icon, "Update" with a refresh icon, and "Check" with a gear icon.
- Settings**: A section header.
- Data**: A section header.
- Splash image**: A section header.
- FPGA**: A section header.

This includes:

- **MainOS and ConfigOS:** The two device operating systems residing on two separated disk partitions. The current installed version is displayed here.
- **Settings:** Disk partition where the device settings are stored. Clearing it will restore the device configuration to factory defaults.

- **Data:** Disk partition where the applications are installed. Clearing it will delete all the applications
- **Splash Image:** The image that is shown during the early boot phase.
- **Xloader, Bootloader and FPGA:** Other versioned components of the BSP. Some of these may only be found on some platforms

### 3.2.1 Making a component backup

All the BSP components can be exported using the “Get” action. When accessing the System Settings from a web browser files are directly downloaded on the PC.

### 3.2.2 Updating a component

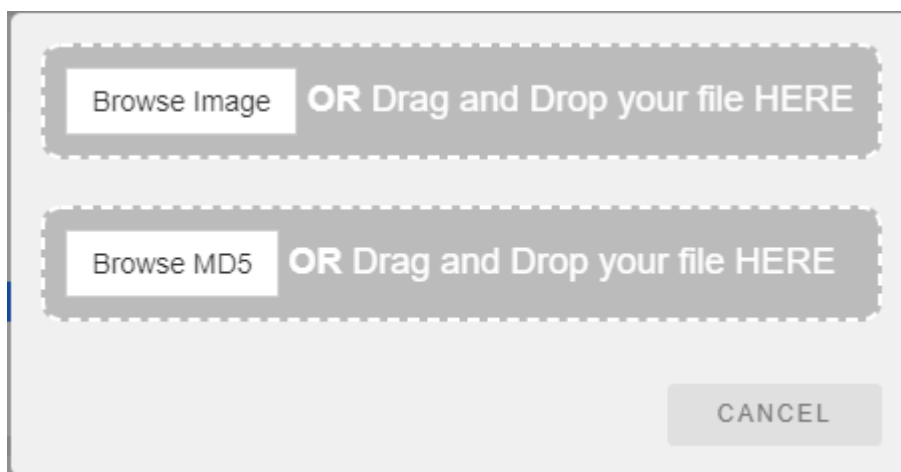
The “Update” action can be used to install a newer version of a BSP component or to restore it using an old backup.

Each update image should also come with a .md5 text file containing its md5 checksum. For example, a typical MainOS update is given with the two files like these:

```
un60-hsxx-mainos-1.0.303.rootfs.tar.gz
un60-hsxx-mainos-1.0.303.rootfs.tar.gz.md5
```

For packages generated with the “Get” action the .md5 file should be generated by the user.

To perform an update the user will be asked to separately supply both the image file and .md5 file:



The .md5 file will be used to test files for integrity before the actual update. Some components may require a system reboot to update, the progress of the process will be shown on screen.

## 3.3 Services

### 3.3.1 SSH Service

Both “user” and “admin” users can be used to login in SSH. Only the “admin” user can however gain root privileges by using sudo or directly becoming root with:

```
$ sudo su
```

Direct SSH root login is not supported. For development purposes it's however possible to temporarily enable it by making few changes to the BSP ( see [9.2 Enabling root SSH login](#) )

### 3.3.2 Avahi Daemon

Avahi is a so called zero-configuration networking service, it enables programs to publish and discover hosts and other services running on a local network by sending multicast DNS packets.

If two devices have this service enabled they can address each other using Avahi hostnames instead of their IP addresses. The Avahi hostname is always set to the device hostname followed by “.local” ( ex. HMI-e62c.local ). The current device hostname can be retrieved and changed in System Settings under Network section.

On most Linux PC distributions Avahi service is usually already enabled by default while under Windows installing the Bonjour service is required. The Bonjour installer for Windows can be downloaded from the following link:

```
https://support.apple.com/kb/DL999?viewlocale=en\_US&locale=en\_US
```

Testing that everything works can be done with a simple ping:

```
$ ping <deviceHostname>.local
```

In order to just obtain the list of the devices connected on the local network it also possible to use the built-in discovery system without using Avahi ( see [10.1 Device discovery](#) )

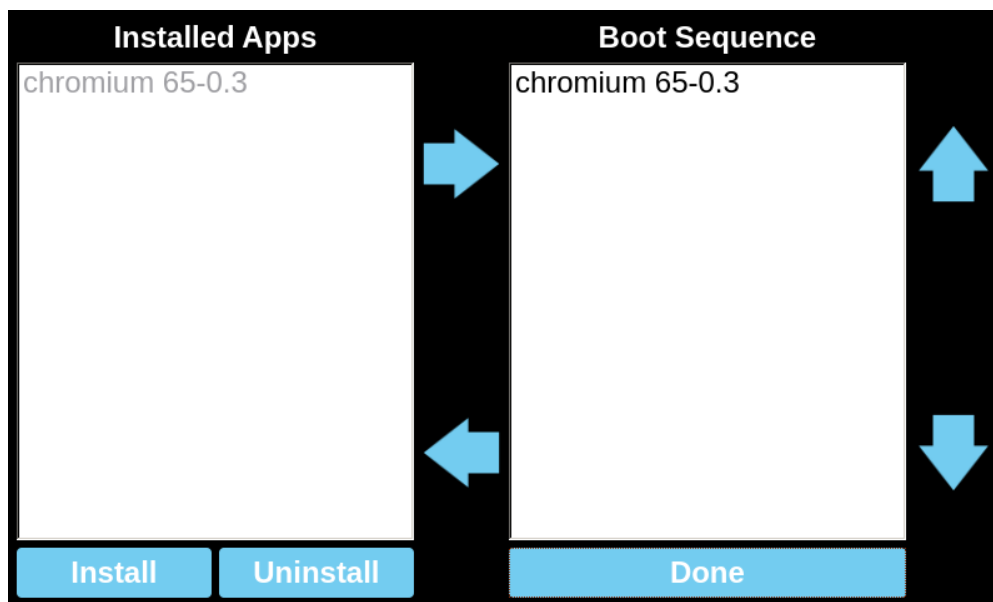
### 3.3.3 Autorun scripts from external storage

Enabling this option allows to automatically execute applications from a USB drive. A bash script named “autoexec.sh” needs to be placed in the root of the USB drive, this script, if found, is executed as soon the drive is plugged into the device.

The script will run with root privileges, for this reason it's suggested to disable this option as soon it's no more needed. There are already some USB tools available that can be used for configuration, updates and more ( see [7. USB Tools](#) ).

## 4 Application Launcher

Applications are managed by the Application Launcher. Configuration can be accessed by interrupting the boot sequence at power on and then pressing the “Startup Sequence” button.



The UI shows the complete list of all the currently installed applications on the left, under “Installed Apps”, and the list of those that are configured to automatically start at boot on the right, under “Boot Sequence”. Entries can be moved around by drag-and-drop or by using the arrow buttons, this can be used to enable/disable applications ( move entries between left and right columns ) and to define the boot order ( move entries up and down under the “Boot Sequence” column )

To uninstall an application simply select the related entry, press “Uninstall” and wait for the process to complete.

To install or update an application from an application package follow these steps instead:

- First the .zip package application should be made available to the devices. There are two common ways to do this:
  - Copy the package on a USB drive and plug it on the device.
  - Copy the package via SCP. To do this the SSH service must be first enabled in System Settings, “Services” -> “SSH Server”. Log in using the “admin” user ( default password is “admin” ) then place the file in a read-write location such /home/admin.
- Press “Install” in the “Startup Sequence” UI. A file browser will be shown. A folder can be opened with a double tap. Browse to the package location, select it and press “Ok”. If the file is on a plugged USB drive it will be found in /mnt/usbmemory.

- Wait for the process to complete. Once finished a new application entry should appear in the list.

As an alternative, it's also possible to install a package using the USB Application Installer tool ( see [7.2 Application Installer](#) )

## 4.1 Kiosk mode

The device without any application installed will boot in the configuration menu but once the startup sequence is populated with at least one application the device will be put in "kiosk" mode. In this case applications in the startup sequence list will be executed and the application launcher will keep track of each process. In order to ensure that the device remains functional the application launcher will automatically reboot the device if it detects that all of the application processes have for some reason ended.

If the boot is interrupted from the "tap-tap menu" ( see [2. Boot Sequence](#) ) the kiosk mode will be forced disabled and won't be restored even if the boot sequence is launched using the "Start HMI" button. In this case all application processes end the configuration menu will show up again instead.

## 4.2 Creating custom packages

It is possible to generate an application package containing some custom software. All the packages are required to contain in their root some specific files, most of them are bash scripts that are used by the Launcher as handles to perform operations such starting and stopping the application.

This is the list of the required files:

- **package.info**: It's an xml file that contains some application basic information:
  - **<name>** The application name.
  - **<installationFolder>** The name of the installation folder. The contents of the package will be installed in /mnt/data/hmi/<installationFolder>.
  - **<version>** Application version.
- **run.sh**: Bash script called by the Launcher to start the application. The script is expected to start the application in background and then return. Here the Launcher should be notified when the application is fully started and when it's terminated, this is done by issuing two different dbus calls using dbus-send:
  - `dbus-send --print-reply --system --dest=com.exor.JMLauncher '/' com.exor.JMLauncher.appLoaded string:"AppName"`

Tells the Launcher that the application is started successfully and that it should proceed to launch the next application in the startup sequence if any. AppName needs to be replaced with the name of the application as found in the package.info file

- `dbus-send --print-reply --system --dest=com.exor.JMLauncher '/' com.exor.JMLauncher.appFinished string:"AppName"`

Tells the Launcher that the application has ended. Again, AppName needs to be replaced with the name of the application as found in the package.info file.

- **stop.sh**: Bash script called by the Launcher to stop the application.
- **uninstall.sh**: Bash script called by the Launcher just before uninstalling the application. The whole `/mnt/data/hmi/<installationFolder>` will be deleted so here it's possible for example to save some configuration files outside this folder for restoring them in a later installation.
- **install.sh**: Bash script called by the Launcher just after extracting the package. Here it's possible for example to check for a saved configuration and restore it. At the end the Launcher should be notified that the installation is completed with the following `dbus-send` command:  
`dbus-send --print-reply --system --dest=com.exor.JMLauncher '/' com.exor.JMLauncher.installFinished int32:0 string:""`

The first `int32` parameter is the status code, values different from 0 means the installation was unsuccessful. If the installation failed, the string parameter can be set to an error message that will be displayed on the screen.

When creating a package it's important to remember that the zip archive should directly contain the above files in its root without any folder containing them.

## 5 BSP structure

The onboard eMMC is the primary boot and storage device. A number of partitions are defined to contain two different Linux operating systems and to serve various other purposes. The MainOS is the default operating system while the other one, called ConfigOS, is used for recovery. The bootloader will automatically boot the recovery system after 3 failed boot attempts, from there the user is able to access the System Settings and possibly restore the MainOS or other components.

Following is the list of all the eMMC partitions, the filesystem used is EXT4 with journaling.

- **Factory Partition** ( /dev/mmcbk1p1 )

The factory partition contains configuration parameters that either needs to be shared between MainOS and ConfigOS or are considered machine factory settings independent of the BSP:

- Screen orientation and calibration
- User passwords ( Linux shadow file )
- Certificates
- BSP settings factory defaults

This partition is mounted read-only in /mnt/factory. It can not be managed from System Settings, to update or backup this partition the USB Updater tool can be used instead ( see [7.1 USB Updater](#) ).

- **ConfigOS Partition** ( /dev/mmcbk1p2 )

Recovery operating system partition. The ConfigOS is almost identical to the MainOS, to ensure its reliability it's made however to be volatile, every configuration change done here is lost upon reboot.

The system may automatically reboot in ConfigOS to perform some operations like updating the MainOS. When in MainOS the partition is mounted read-only in /mnt/configos.

- **MainOS Partition** ( /dev/mmcbk1p3 )

Main operating system partition. The MainOs is where applications run. When in ConfigOS the partition is mounted read-only in /mnt/mainos.

- **Etc Partition** ( /dev/mmcbk1p5 )



In System Settings management section this partition is referred just as “Settings”. It initially contains a copy of the contents of the MainOS /etc folder and it’s mounted read-write over it. Basically all the BSP configuration parameters that are not saved in the factory partition can be found here.

When settings are restored to defaults the partition is reinitialized with files from the underling MainOS /etc folder.

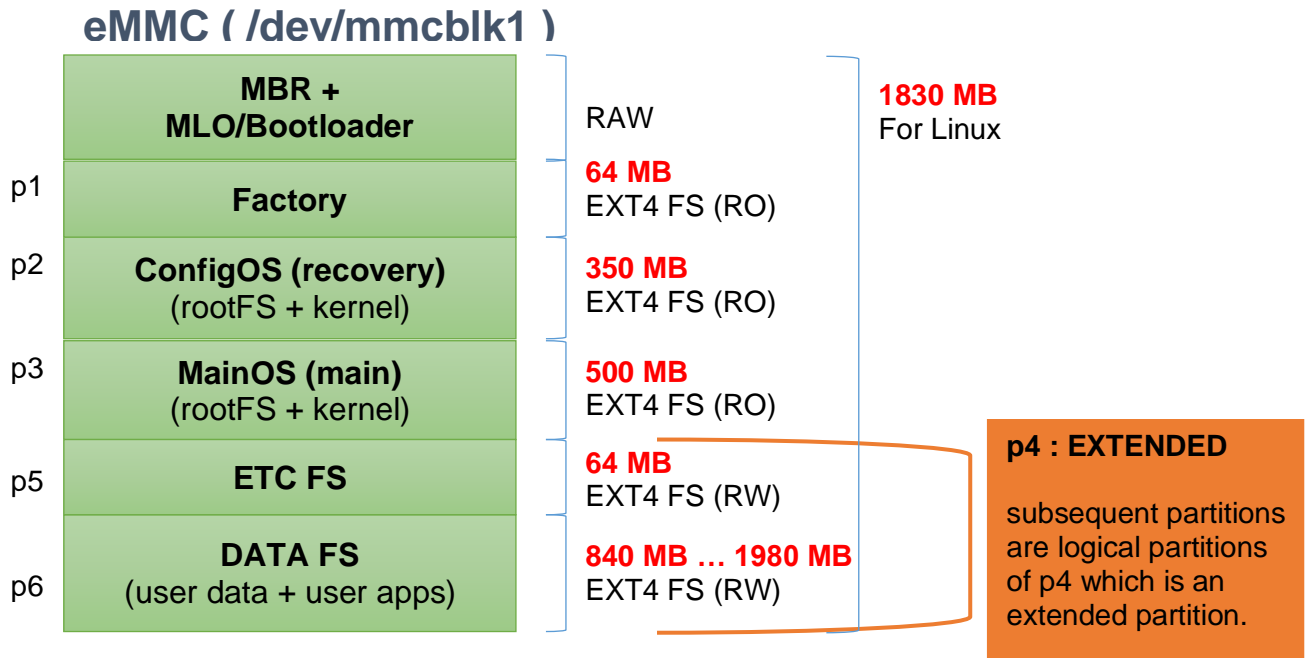
- **Data Partition** ( /dev/mmcblk1p6 )

This partition is mounted read-write in /mnt/data and it’s where the user can store his files, the Application Launcher also installs applications here in /mnt/data/hmi. The /home folder is relocated on this partition ( /mnt/data/home ) and it’s therefore writable too.

When updating remotely, packages are temporarily stored here. For this reason it’s important to make sure that at least 150MB of this partition are always left free.

This partitioning system has been defined this way to optimize reliability, data that it’s not required to be writable is always kept read-only and BSP settings data are separated from application data. Writing outside the data partition is not recommended, any changes made elsewhere could also be lost after a BSP update.

Following is a diagram of the eMMC partitioning. All the above partitions reside on the first 2GB, the second half is left unmanaged by Linux and can be used for optional other operating systems ( ex. Android ) .



## 6 Using the serial port

The serial port in Linux can be used by accessing the correct `/dev/ttyXX` device node. The actual name of the serial port device varies depending on the specific platform as shown below:

- eSMART, eX705 and eXware703                      `/dev/ttyO0`
- eTOP6xxL    `/dev/ttyS0`
- eX707/710/715/721, eXware707, X10           `/dev/ttymxc0`

It's also possible to make use of the `/dev/com1` symlink which is always created to point to the correct device and gives a platform independent way to access the serial port.

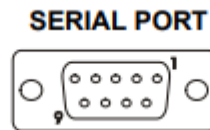
During development the serial port can also be used to get a console ( see [10.1 Enabling the serial console](#) ).

### 6.1 Serial port pinout

#### 6.1.1 Pinout for eSMART and eTOP6xxL series

##### RS-232

Pin	Description
1	GND
2	
3	TX
4	RX
5	
6	+5V output
7	CTS
8	RTS
9	

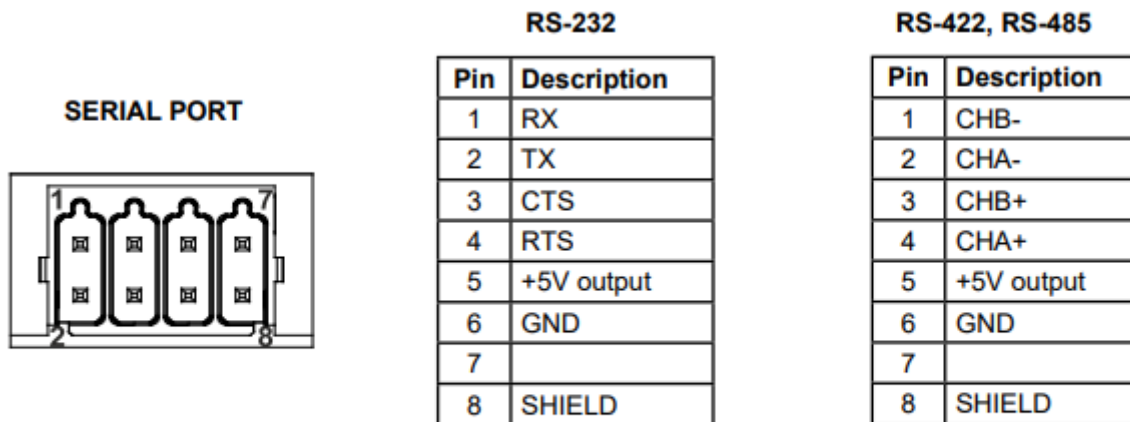


##### RS-422, RS-485

Pin	Description
1	GND
2	
3	CHA-
4	CHB-
5	
6	+5V output
7	CHB+
8	CHA+
9	

To operate in RS485 pins 4-3 and 8-7 must be connected externally.

## 6.1.2 Pinout for eXWare and eX7xx series



To operate in RS-485 pins 1-2 and 3-4 must be connected externally.

## 6.2 Configuring the serial port

Serial ports may support working in different modes, refer to the device datasheet to know the list of the supported modes. By default serial ports are always set to operate in RS232 115200n8.

### 6.2.1 Changing the operating mode

If supported, changing the operating mode can be done using the Linux TIOCSRS485 ioctl. Following is a piece of C code that shows the use of this ioctl to set the serial port to work in RS485, the name of the device below "/dev/ttyO0" should be changed accordingly.

```
fprintf(stderr, "\tOpen fd...");
int fd = open("/dev/ttyO0", O_RDWR);

if (fd < 0) {
    perror("Open device failure");
    return -1;
}

fprintf(stderr, " done!\n");
```

```
fprintf(stderr, "\tEnable RS485 mode...");
struct serial_rs485 rs485conf;

if (ioctl(fd, TIOCGRS485, &rs485conf) < 0) {
    perror("ioctl failure");
    return -2;
}

rs485conf.flags = SER_RS485_ENABLED | SER_RS485_RTS_ON_SEND;

if (ioctl(fd, TIOCSRS485, &rs485conf) < 0) {
    perror("ioctl failure");
    return -2;
}

fprintf(stderr, " done!\n");
```

The mode is determined by the definition of the `rs485conf.flags` flag that can be seen shown in bold in the code above. Here are the definitions of this flag for the different modes:

- RS232 mode ( default ):  
**rs485conf.flags** = 0;
- RS485 mode:  
**rs485conf.flags** = SER\_RS485\_ENABLED | SER\_RS485\_RTS\_ON\_SEND;
- RS422 mode:  
**rs485conf.flags** = SER\_RS485\_ENABLED | SER\_RS485\_RTS\_ON\_SEND | SER\_RS485\_RX\_DURING\_TX;

## 6.2.2 Configuring the port

Port baud rate, parity and other properties can be set as usually done under Linux. Under a C/C++ software the family of functions defined in `termios.h` can be used, refer to the manual for more information on this:

```
https://linux.die.net/man/3/termios
```

As a generic alternative way of doing this the “stty” command line tool, included in the BSP, can also be used. For example this line will set the port `/dev/ttyO0` to 115200 8o1 :

```
# stty -F /dev/ttyO0 115200 cs8 parenb parodd
```

For more information on stty refer to its manual:

<https://linux.die.net/man/1/stty>

## 7 USB tools

These are tools that can be placed on a USB stick and automatically executed on the panel just by plugging the drive. Some of them are available here:

```
http://download.exoreembedded.net:8080/Public/ExorPanels/USBTools
```

When preparing an USB drive the only thing to remember is that files needs to be extracted so that the “autoexec.sh” file is directly found in the drive’s root.

All the USB tools have the following features in common:

- If the device has a display the process status and progress is shown.
- If the operation was successful the buzzer is played 3 times in the end. If a failure occurred instead, the buzzer is played with a 3s period until reboot.
- If required, the device will automatically reboot once the USB drive is unplugged.
- A log of the last execution is saved on the USB key inside the “lastRun.log” file.

In order to use these tools the “Autorun scripts from external storage” option needs to be enabled in System Settings, “Services” section.

### 7.1 USB Updater

This tool can be used to update BSP components and generate backups just like it can be done from System Settings. Tool’s behavior is partially determined by some option variables defined inside the “src/config” configuration file.

The tool requires the user to authenticate using the admin user’s password if it has been changed from the default. Authentication can be done automatically if the CFG\_ADMIN\_PASSWORD variable inside the configuration file is set to the current admin password.

#### 7.1.1 BSP update

The tool will update every BSP component for which an update package is found. Packages should be placed inside the “src” folder together with their .md5 files and should be renamed in the following way:

- |                           |                    |
|---------------------------|--------------------|
| • mainos.tar.gz + *.md5   | MainOS             |
| • configos.tar.gz + *.md5 | ConfigOs           |
| • data.tar.gz + *.md5     | Data partition     |
| • settings.tar.gz + *.md5 | Settings partition |
| • factory.tar.gz + *.md5  | Factory partition  |
| • u-boot.img + *.md5      | Bootloader         |

- MLO.img + \*.md5                      Xloader ( if supported )
- splashimage.bin                        Splash image
- fpga.bin                                 FPGA ( if supported )

By default, before starting updating, the signature of each update package is checked to verify its compatibility with the platform, if the check fails the process will fail without any action taken. This check can be turned off by setting CFG\_CHECK\_SIGNATURE to 0.

### 7.1.2 BSP backup

If CFG\_ENABLE\_BACKUP is set to 1 the tool will also do a BSP backup before updating. CFG\_BACKUP\_COMPONENTS can be set to a space separated list of components to backup, valid component names are:

- mainos                                  MainOS rootfs tar.gz
- configos                                ConfigOS rootfs tar.gz
- data                                     Data partition tar.gz
- settings                                Settings, etc partition tar.gz
- factory                                 Factory partition tar.gz
- bootloader                             Bootloader binary
- xloader                                 Xloader binary ( if supported )
- splash                                  Splash image ( if found )
- fpga                                     FPGA binary ( if supported )

If no list is specified, the default is to backup everything except FPGA.

Generated files can be found in the USB drive, inside a folder named backup\_<date>, this also includes all the corresponding \*.md5 files. These package files are compatible with the System Settings ( see 3.2.2 *Updating a component* ).

This feature can easily be used to clone devices starting from a single preconfigured panel:

- Configure a device with final versions of each BSP component and wanted panel settings, applications, application settings, boot sequence etc.
- Use the tool to do a backup ( CFG\_ENABLE\_BACKUP=1 without update packages)
- Disable the backup ( CFG\_ENABLE\_BACKUP=0 ) and move all files from the generated backup folder to the “src” folder.

The USB drive can now be used multiple times to configure other devices just like the original one.

## 7.2 Application Installer

This tool can be used to automatically install an Application Launcher package ( see [4. Application Launcher](#) ). The package to install should be placed along the other files renamed as either “package.zip” or “UpdatePackage.zip”.



At the end of the operation the system will be automatically restarted.

## 8 Compiling applications for the target

The standard Zeus SDK containing the cross-compiler and all the necessary libraries is available for download here:

- 64 bit devices (JSmartxxM, eX7xxM and X10):

```
http://download.exoreembedded.net:8080/Public/ExorPanels/SDK-1.3.x/un60-xxxx-sdk-1.3.4.sh
```

- 32 bit devices (all others):

```
http://download.exoreembedded.net:8080/Public/ExorPanels/SDK-2.0.x/un78-xxxx-sdk-2.0.2.sh
```

The current version of the SDK is based on Zeus 3.0.4, uses GCC 9.2.0 and comes with Qt 5.13.2 libraries. To use it a Linux machine is required, installation is as simple as executing the SDK installer script. For example:

```
$ chmod +x un60-xxxx-sdk-1.3.4.sh
$ sudo ./un60-xxxx-sdk-1.3.4.sh
```

Make sure to use different folders for the 2 SDK (64 and 32bit) during installation, otherwise some file will be overwritten and the first SDK installed will not work. The following instructions refer to default path.

For further support on how to use the SDK it's also possible to refer to the official Yocto manual:

```
https://www.yoctoproject.org/docs/2.1/sdk-manual/sdk-manual.html
```

### 8.1 Using QtCreator IDE

We are now going through the complete process of configuring the environment from scratch. This guide takes version 4.5.2 of QtCreator as reference, a simple graphical installer can be downloaded from here:

```
https://download.qt.io/official\_releases/qtcreator/4.5/
```

The file to download is the Linux .run executable and can be started from shell:

```
$ chmod +x qt-creator-opensource-linux-x86_64-4.5.2.run
$ ./qt-creator-opensource-linux-x86_64-4.5.2.run
```

It's important to start QtCreator from a shell after having done the SDK environment source:

- 64 bit devices (JSmartxxM, eX7xxM and X10):

```
$ source /opt/exorintos-2.0.x/2.0.x/environment-setup-aarch64-poky-linux
$ qtcreeator
```

- 32 bit devices (all other):

```
$ source /opt/exorintos-1.3.x/1.3.x/environment-setup-cortexa8hf-neon-poky-linux-
gnueabi
$ qtcreeator
```

As an alternative it's also possible to create a bash script containing the above two lines to start the IDE more easily.

Before proceeding with the IDE configuration make sure that the SDK has been installed on the system.

### 8.1.1 Build configuration

From the "Tools" menu, select "Options..." -> "Build & Run", then follow these steps:

- 1) In the "Compilers" tab click on "Add" -> "GCC" -> "C" and select the cross compiler picking it from the SDK installation folder. If the SDK has been installed in the default location the correct path is:

- 64 bit devices (JSmartxxM, eX7xxM and X10):  
`/opt/exorintos-2.0.x/2.0.x/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gcc.`

- 32 bit devices (all other):  
`/opt/exorintos-1.3.x/1.3.x/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gcc.`

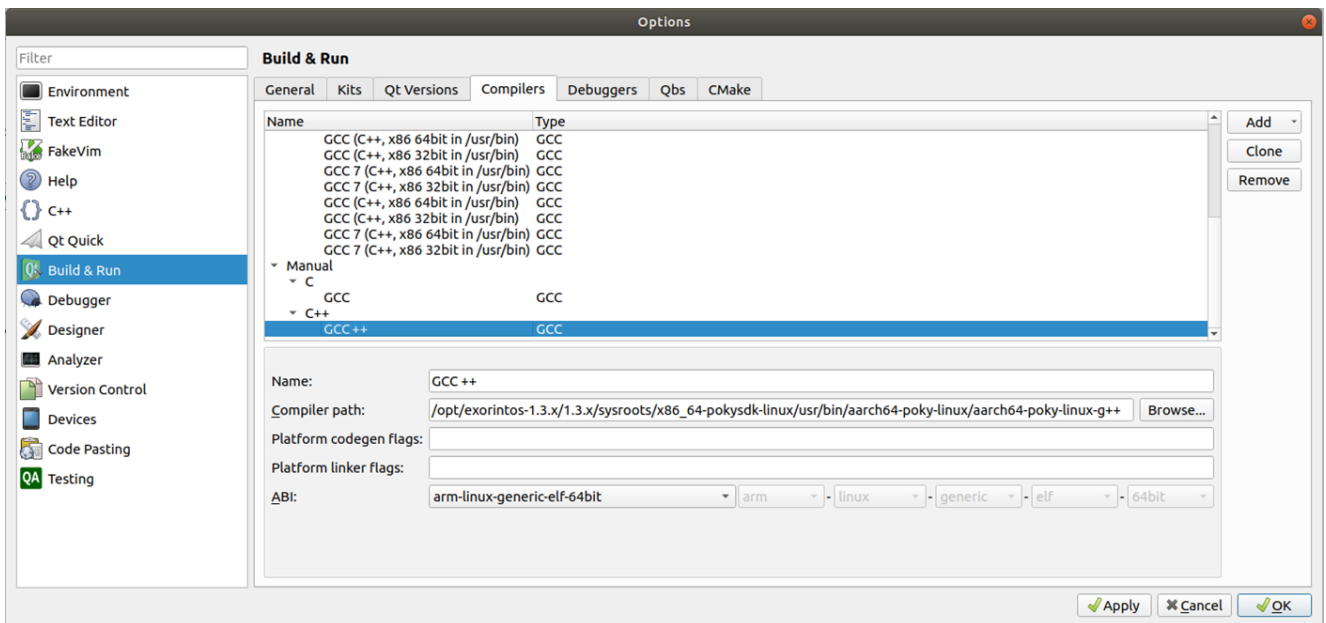
Optionally edit "Name" to give a more meaningful name for the entry, select "arm-linux-generic-elf-64bit" or "arm-linux-generic-elf-32bit" as ABI and finally click "Apply".

- 2) From the same tab now click "Add" -> "GCC" -> "C++" and select:

- 64 bit devices (JSmartxxM, eX7xxM and X10):  
`/opt/exorintos-2.0.x/2.0.x/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-g++ .`

- 32 bit devices (all other):  
`/opt/exorintos-1.3.x/1.3.x/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++.`

Again, select "arm-linux-generic-elf-64bit" or "arm-linux-generic-elf-32bit" as ABI and click "Apply".



3) From “Debuggers” tab press “Add” and select gdb from the same directory. The default location is:

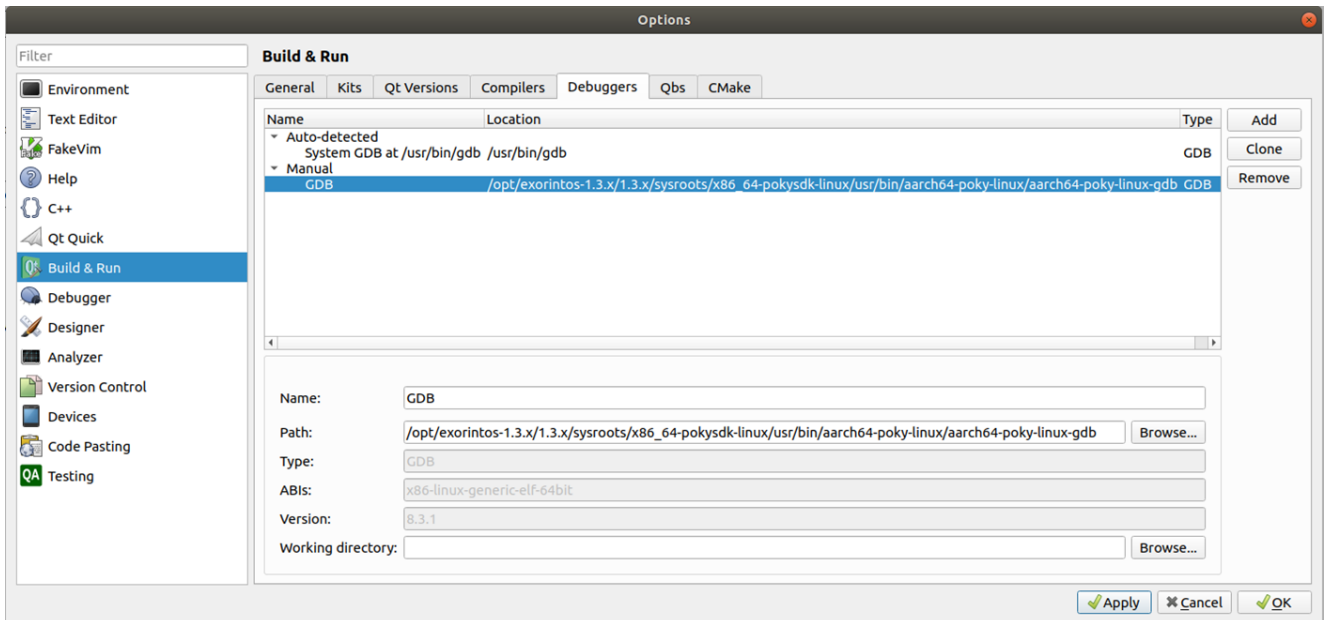
- 64 bit devices (JSmartxxM, eX7xxM and X10):

`/opt/exorintos-2.0.x/2.0.x/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb.`

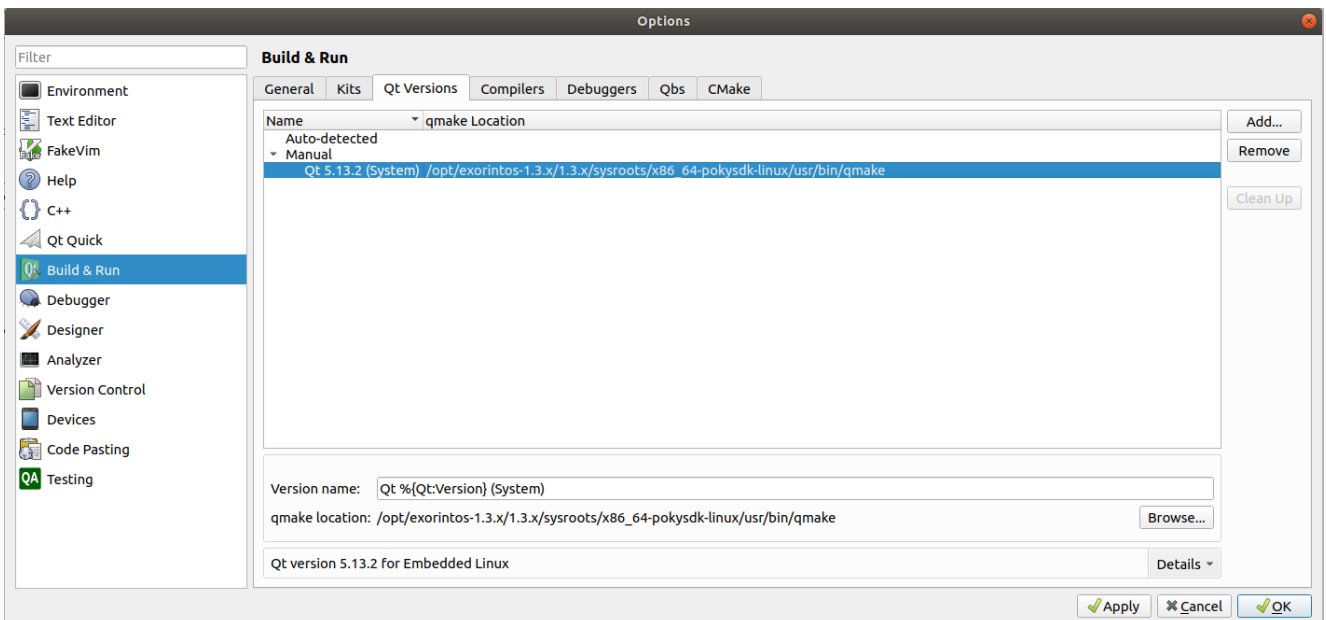
- 32 bit devices (all other):

`/opt/exorintos-1.3.x/1.3.x/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb.`

Optionally edit “Name”, then click “Apply”.



- 4) From “Qt Versions” tab, press “Add..”. The default path to select for the 32bit SDK is: `/opt/exorintos-1.3.x/1.3.x/sysroots/x86_64-pokysdk-linux/usr/bin/qmake` (adjust the path for the 64bit SDK). QtCreator should automatically recognize the Qt version selected. Press “Apply”.  
This file will be overwritten if you installed the 2 different SDK in the same default folder and this is why you need to use different folder.

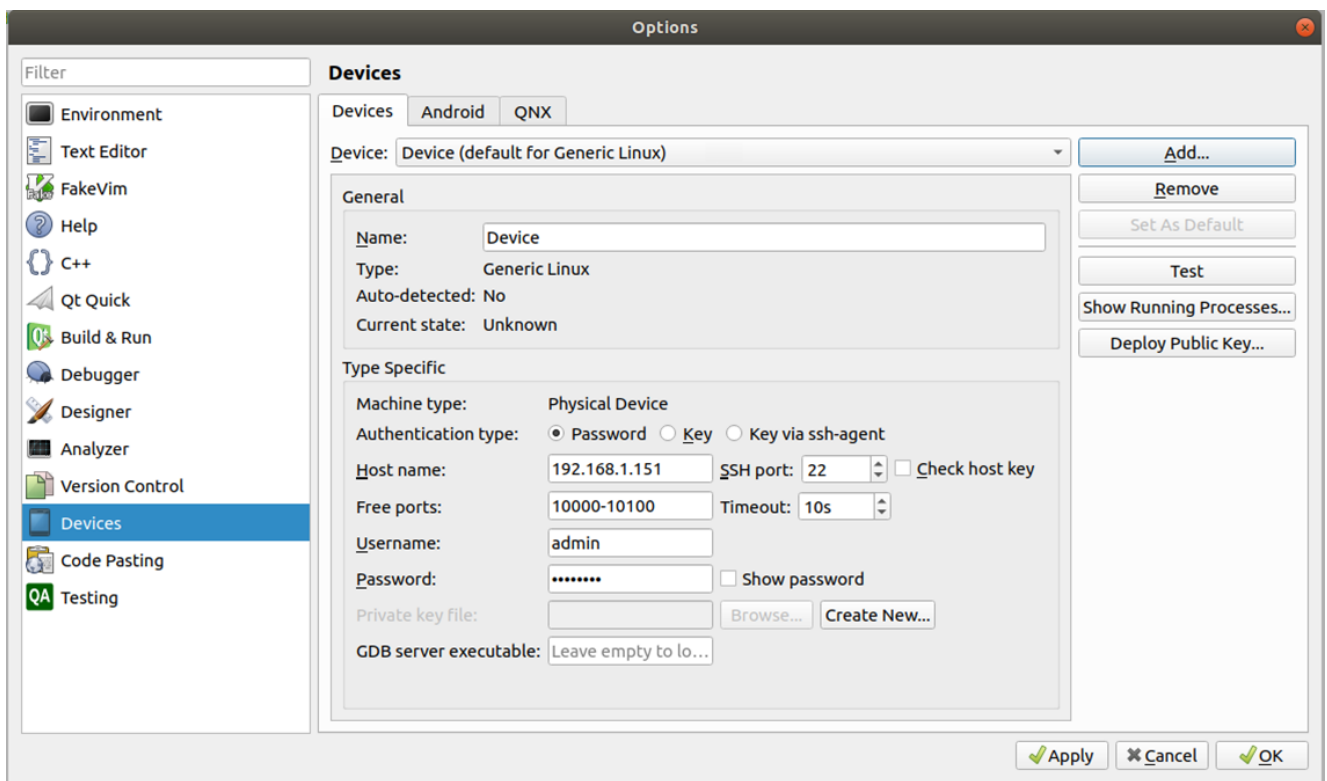


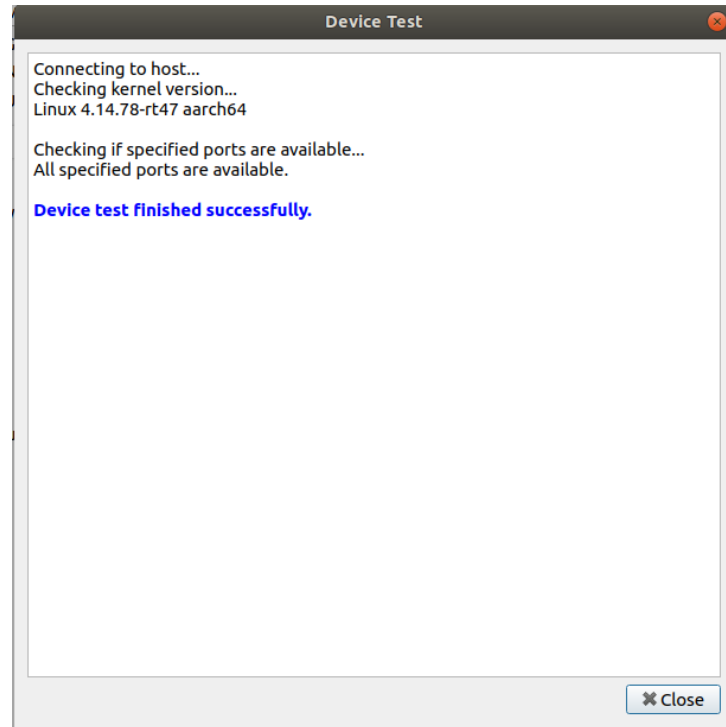
- 5) This step is required for configuring automatic application deployment on the target.

Move to the “Devices” section. To define a new devices from scratch, click on “Add..”, select “Generic Linux Device” and press “Start Wizard”. Fill in these informations:

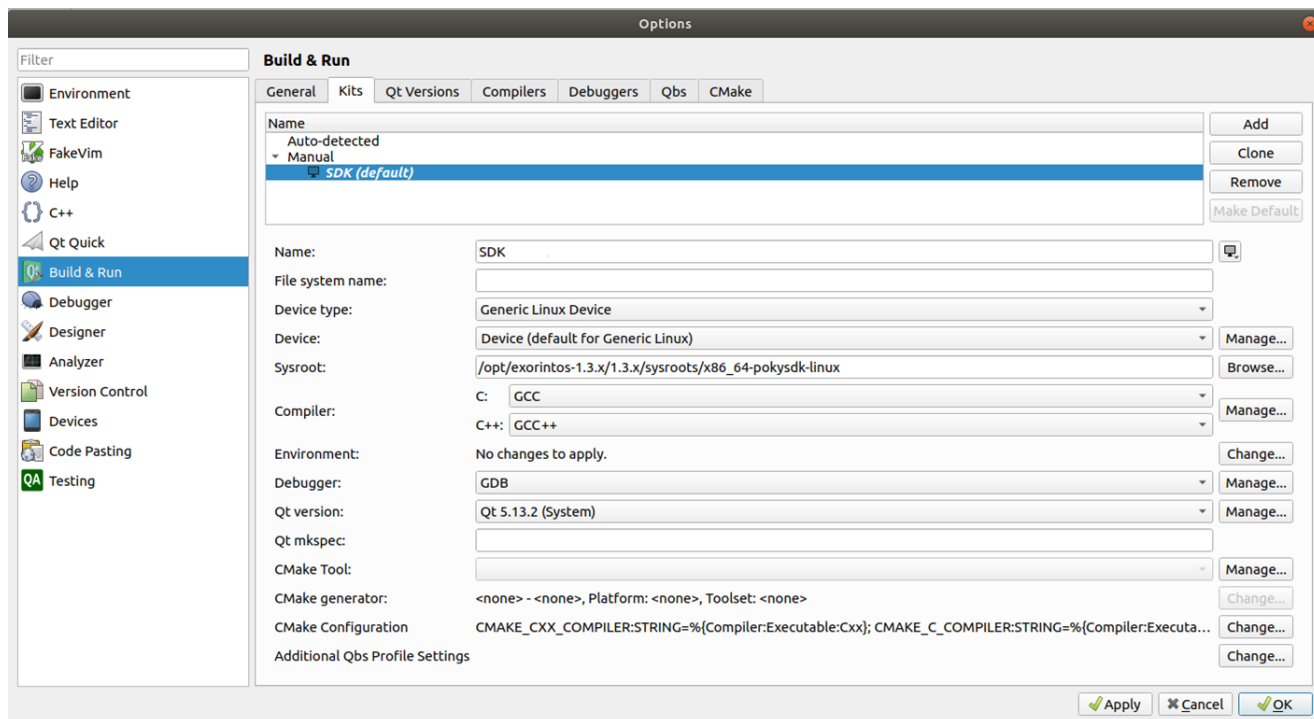
- **Name:** select a name for the device.
- **Host name:** enter the device IP address.
- **Username:** `admin`.
- **Authentication type:** set to “Password”.
- **Password:** password set on first start up.

Click “Next” and then “Finish”. Qt Creator will attempt a test connection, if the device is already powered on and reachable everything should be ok.





- 6) Finally move again to the “Kits” tab. Combine all pieces together in a new kit. Click “Add” and fill in as follows:
- **Name:** choose a name for the kit.
  - **Device Type:** select “Generic Linux Device”.
  - **Device:** select the device configured in 5).
  - **Sysroot:** if the SDK is installed in the default location, the path to select for the 32bit SDK is: `/opt/exorintos-1.3.x/1.3.x/sysroots/x86_64-pokysdk-linux/`
  - **Compiler:** select C and C++ compilers by name as configured in 1) and 2).
  - **Debugger:** select debugger by name as configured in 3).
  - **Qt version:** select qt version added in 4).

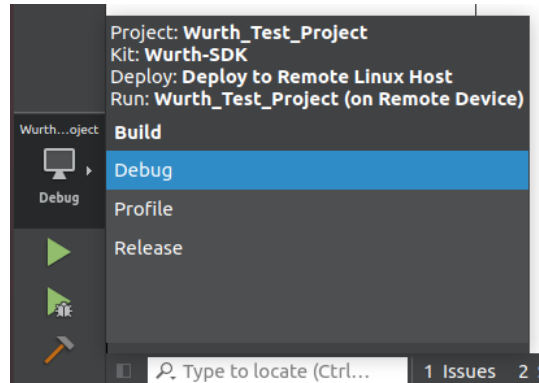


## 8.1.2 Application deploy

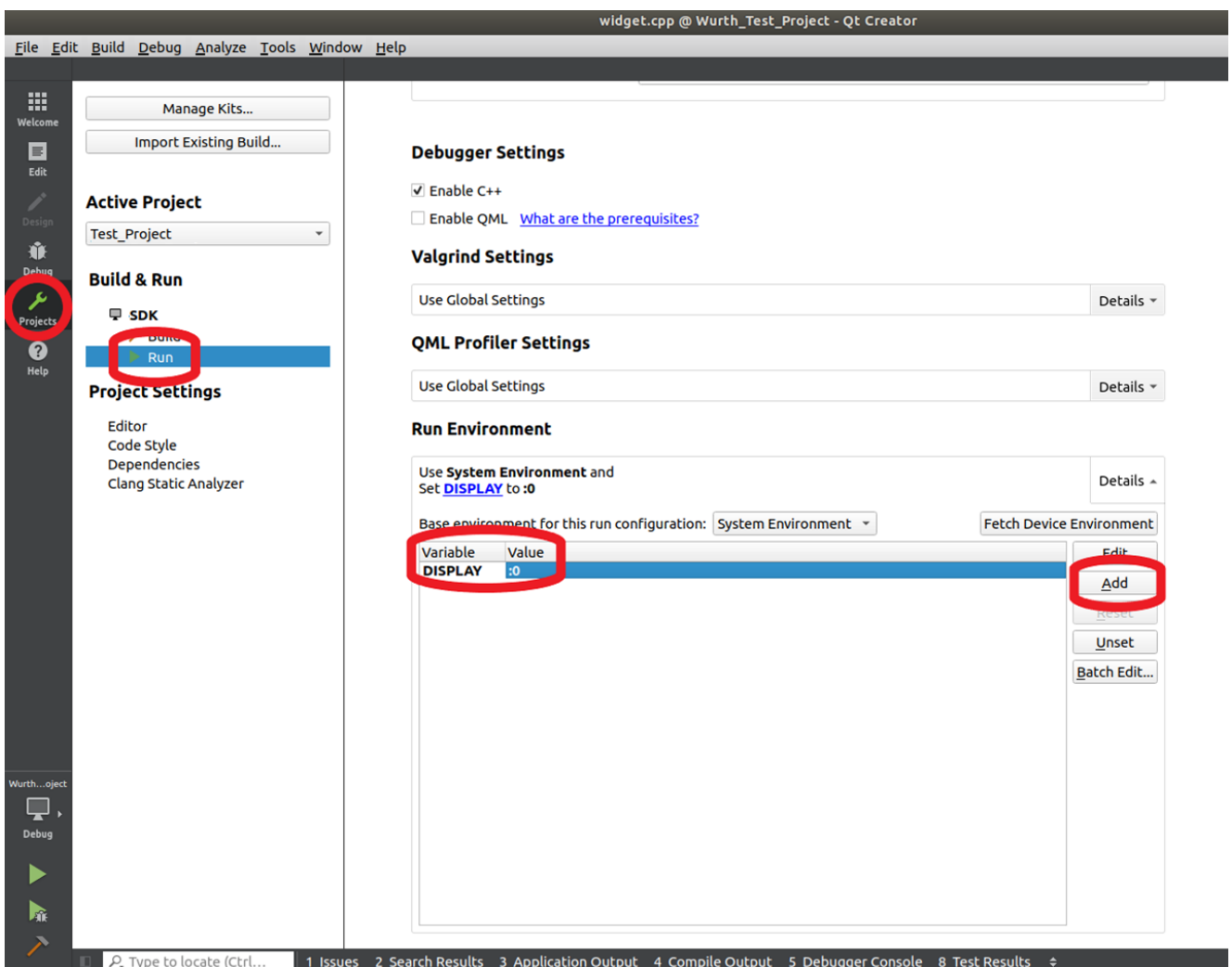
Before starting here, make sure that QtCreator has been correctly configured for application deployment and that the device is reachable. QtCreator will install and run the software over SSH so make also sure that it's enabled on the device from System Settings, "Services" -> "SSH Service".

- 1) First, let's create a dummy Qt project. Select "File" -> "New File or Project..." -> "Qt Widgets Application" and click "Choose". Enter a project name, press "Next". Make sure that in the "Kit Selection" wizard dialog the SDK kit for the target is selected.
- 2) Make sure the target kit the one currently in use by checking in the menu on the left shown below:





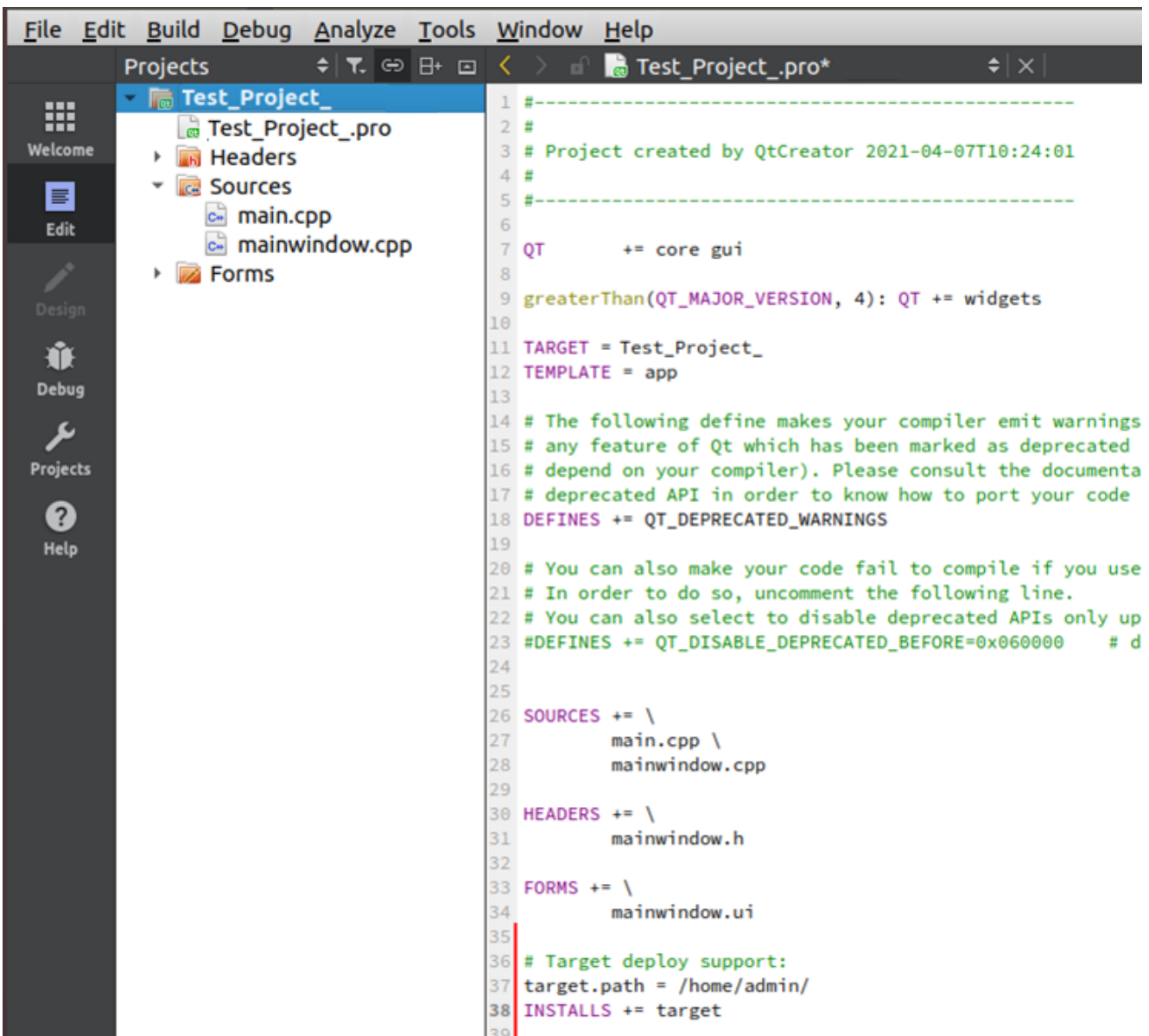
- 3) For 1.3.x BSPs based on Xorg there's an extra step. Click on "Projects" in the menu on the left and then "Run" as shown in the screenshot below. Scroll down to the "Run Environment" options, press "Add" and define a variable `DISPLAY` with value `:0`.



4) Now edit the .pro project file to add these two lines:

```
target.path = /home/admin/  
INSTALLS += target
```

This will define where the application will be installed on the device (/home/admin)



```
File Edit Build Debug Analyze Tools Window Help  
Projects Test_Project_.pro*  
Test_Project_  
  Test_Project_.pro  
  Headers  
  Sources  
    main.cpp  
    mainwindow.cpp  
  Forms  
1  #-----  
2  #  
3  # Project created by QtCreator 2021-04-07T10:24:01  
4  #  
5  #-----  
6  
7  QT      += core gui  
8  
9  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets  
10  
11 TARGET = Test_Project_  
12 TEMPLATE = app  
13  
14 # The following define makes your compiler emit warnings  
15 # any feature of Qt which has been marked as deprecated  
16 # depend on your compiler). Please consult the documenta  
17 # deprecated API in order to know how to port your code  
18 DEFINES += QT_DEPRECATED_WARNINGS  
19  
20 # You can also make your code fail to compile if you use  
21 # In order to do so, uncomment the following line.  
22 # You can also select to disable deprecated APIs only up  
23 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # d  
24  
25  
26 SOURCES += \  
27     main.cpp \  
28     mainwindow.cpp  
29  
30 HEADERS += \  
31     mainwindow.h  
32  
33 FORMS += \  
34     mainwindow.ui  
35  
36 # Target deploy support:  
37 target.path = /home/admin/  
38 INSTALLS += target  
39
```

- 5) Finally press the green play button in the menu on the left or use the “Ctrl+R” shortcut. QtCreator should compile the application and an empty Qt window should appear on the device.



## 9 Other useful information

### 9.1 Device discovery

The BSP does have a built-in discovery system. Devices respond to a broadcast discovery message by communicating their IP address and hostname.

The source code of a Qt4 example using this system is available here:

```
http://download.exoreembedded.net:8080/Public/ExorPanels/Other/SelfInfo/selfinfo\_qt4\_src.zip
```

### 9.2 Enabling the serial console

Having the serial console enabled it's not a supported operating mode, however it could be useful for debugging purposes during development.

It's possible to enable it by accessing the device via SSH. Make sure the SSH Service is enabled, log in with the admin user and type the following commands:

```
# root permissions are required
$ sudo su
# The file resides on the rootfs /etc folder
$ umount -l /etc
$ mount -o remount,rw /
# Open /etc/inittab with a text editor
$ nano /etc/inittab
```

Inside `/etc/inittab`, line 31 needs to be uncommented and modified to specify the correct serial port device name, check [5. Using the serial port](#) for the name to use.

For example, if the correct device is `/dev/ttymx0`, the line should be changed from:

```
# 00:12345:respawn:/sbin/getty 115200 tty00
```

To:

```
00:12345:respawn:/sbin/getty 115200 ttymx0
```

To complete the operation save the file, sync the changes to disk and reboot:

```
$ sync
$ reboot -f
```

Serial ports, by default, operate in RS232 115200n8.

After a BSP update the modified file will be overwritten and above changes have to be manually applied again. Also it's important to remember to disable the console before trying to use the serial port for any other purpose.

### 9.3 Enabling root SSH login

SSH login as root user is disabled. One possibility to gain root privileges is to login as admin user and then use sudo or become root with:

```
$ sudo su
```

However to write in some filesystem locations with SFTP or to execute an application as root remotely from an IDE, direct root login can be useful.

Enabling it is not a supported configuration and should be done only for development purposes. It can be done from a shell:

- 1) Open the `/etc/ssh/sshd_config` file with an editor:

```
$ sudo su
$ nano /etc/ssh/sshd_config
```

Look for this line:

```
PermitRootLogin no
```

And change it to:

```
PermitRootLogin yes
```

- 2) Choose a password for the root user:

```
$ passwd root
```

- 3) Replace the shadow file in `/mnt/factory` and reboot

```
$ mount -o remount,rw /mnt/factory
$ cp /etc/shadow /mnt/factory
$ chown root:shadow /mnt/factory/shadow
$ sync
$ reboot -f
```

It should be now possible to login as root with the chosen password. Please note that, after a BSP update or settings restore, changes to the `/etc/ssh/sshd_config` file will be lost and have to be manually applied again.

## 9.4 Recovering from a damaged MainOS

During development it's always possible to get things wrong to the point that the system may not boot properly any more. In these cases it's still possible to boot in recovery mode and have the MainOS restored.

If the device's boot sequence gets to the point where the loading bar is shown it should be possible to reboot in ConfigOS from the tap-tap menu ( see [2. Boot sequence](#) ). If this can not be done there are other two possibilities to get to recovery mode:

1. The bootloader will automatically boot in ConfigOS after 3 failed attempts of booting the MainOS. Power on/off the device waiting few seconds between reboots until the panel boots in recovery mode.
2. The u-boot prompt can be accessed from the serial port by keeping pressed `ctrl+c` on the console at early device power on. From there it is possible to ask the bootloader to load the ConfigOS with this command:

```
# run altbootcmd
```

Once in ConfigOS there are two ways the MainOS can be restored:

1. Update the MainOS from System Settings with an official package.
2. If the cause of the boot failing is known it's possible to fix it from a SSH shell. When in ConfigOS SSH is always disabled and needs to be enabled again from System Settings. The data partition can be found in `/mnt/data` as always while the MainOS rootfs can be found mounted RO in `/mnt/mainos` and can be remounted RW if needed.

## 10 Source code

BSP open source code can be obtained under request. Sources for the Linux kernel and U-Boot bootloader for all our platforms are also publicly available on our GitHub account:

<https://github.com/ExorEmbedded>

### 10.1 Building the Linux kernel

#### 10.1.1 Source code and configuration

Git source repository, branch and configuration vary depending on the specific platform:

	Repository	Branch	defconfig	dtb
eSMART	<a href="https://github.com/ExorEmbedded/linux-us01">https://github.com/ExorEmbedded/linux-us01</a>	ti-linux-3.12.y	am33xxusom	usom_eco.dtb
PLCM07	<a href="https://github.com/ExorEmbedded/linux-us01">https://github.com/ExorEmbedded/linux-us01</a>	ti-linux-3.12.y	am33xxusom	usom_plcm07.dtb
eTop6xxL	<a href="https://github.com/ExorEmbedded/linux-us02">https://github.com/ExorEmbedded/linux-us02</a>	4.1-LTS_us02_etop	socfpga-usom	usom_etop6xx.dtb
eX705, eXWare	<a href="https://github.com/ExorEmbedded/linux-us01">https://github.com/ExorEmbedded/linux-us01</a>	ti-linux-3.12.y	am33xxusom	usom_etop705.dtb
eX707 / 710	<a href="https://github.com/ExorEmbedded/linux-us03">https://github.com/ExorEmbedded/linux-us03</a>	master	imx6usom	usom_etop7xx.dtb
eX715 / 21	<a href="https://github.com/ExorEmbedded/linux-us03">https://github.com/ExorEmbedded/linux-us03</a>	master	lmx6usom	usom_etop7xxq.dtb
JSmart05 / 07 / 10	<a href="https://github.com/ExorEmbedded/linux-us03">https://github.com/ExorEmbedded/linux-us03</a>	master	imx6usom	usom_jsmart.dtb
JSmart15 / 21	<a href="https://github.com/ExorEmbedded/linux-us03">https://github.com/ExorEmbedded/linux-us03</a>	master	imx6usom	usom_jsmartq.dtb

#### 10.1.2 Deploy

A built kernel can be deployed for testing by simply placing the files in the /boot folder. We highly recommend to make a backup of the zImage and dtb files that would be otherwise overwritten. The

root folder is by default mounted read-only and needs to be remounted read-write before any file can be replaced:

```
$ sudo su
$ mount -o remount,rw /
```

If something goes wrong with the new kernel image chances are that the device will not boot at all. In this case remember that it's still possible to boot in ConfigOS to restore these files (see [10.4 Recovering from a damaged MainOS](#)).

## 10.2 Building the U-Boot bootloader

Git source repository, branch and configuration vary depending on the specific platform:

	Repository	Branch	config
eSMART, PLCM07, eX705, eXWare	<a href="https://github.com/ExorEmbedded/uboot-us01">https://github.com/ExorEmbedded/uboot-us01</a>	uboot2014.04_uS01	am335x_usom_config
eTop6xxL	<a href="https://github.com/ExorEmbedded/uboot-us02">https://github.com/ExorEmbedded/uboot-us02</a>	us02_etop	us02_etop_config
eX707 / 710, JSmart05 / 07 / 10	<a href="https://github.com/ExorEmbedded/uboot-us01">https://github.com/ExorEmbedded/uboot-us01</a>	uboot2014.04_uS01	mx6dl_usom_config
eX715 / 21, JSmart15 / 21	<a href="https://github.com/ExorEmbedded/uboot-us01">https://github.com/ExorEmbedded/uboot-us01</a>	uboot2014.04_uS01	mx6q_usom_config

## 10.3 BSP source code

To get the full BSP related open source code and licensing information please refer to the following web page:

```
http://oss.exorint.net
```