



Exor Linux BSP Development User Manual

UM0013 (v1.11) – 30 Sep 2020

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.....	6
2.	Boot sequence.....	7
3.	System Settings.....	9
3.1	Sections overview.....	10
3.2	Management.....	11
3.2.1	Making a component backup	12
3.2.2	Updating a component	12
3.2	Services.....	13
3.2.1	SSH Service	13
3.2.2	Avahi Daemon.....	13
3.2.3	Autorun scripts from external storage	13
4	Application Launcher	14
4.1	Kiosk mode.....	15
4.2	Available application packages.....	15
4.3	Creating custom packages	15
5	BSP structure	17
6	Using the serial port.....	20
6.1	Serial port pinout	20
6.1.1	Pinout for eSMART and eTOP6xxL series.....	20
6.1.2	Pinout for eXWare and eX7xx series	21
6.2	Configuring the serial port.....	21
6.2.1	Changing the operating mode	21
6.2.2	Configuring the port.....	22
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.....	25
8.1	Using the pre-configured VirtualBox VM	25
8.1.1	Setup a guest-host shared folder.....	27
8.1.2	Using the ready development environment.....	27

8.2	Using QtCreator IDE	28
8.2.1	Build configuration	28
8.2.2	Application deploy	32
9	Using the Dbus HAL interface	35
9.1	Display.....	35
9.2	Devices control	39
9.3	BSP Management	40
9.4	Network.....	42
9.4.1	Network JSON object specification.....	43
9.5	Application management	45
9.5.1	Application JSON object specification	49
9.6	Other	50
10	Other useful information.....	53
10.1	Device discovery.....	53
10.2	Enabling the serial console	53
10.3	Enabling root SSH login.....	54
10.4	Recovering from a damaged MainOS	55
11	Source code	56
11.1	Building the Linux kernel.....	56
11.1.1	Source code and configuration	56
11.1.2	Deploy.....	56
11.2	Building the U-Boot bootloader	57
11.3	BSP source code	57

Version History

Version	Release date	Changes
1.0	22/09/2017	First Version
1.1	4/04/2018	Added 9 Using the EPAD Dbus interface Added 10 Other useful information, Enabling the serial console
1.2	5/04/2018	Added 8.2 Using QtCreator IDE Added 10.3 Enabling root SSH login Added 3.2.1 SSH Service Added 3.2.2 Services, Avahi Daemon
1.3	20/05/2018	Added 5 BSP structure Added 10.1 Device discovery
1.4	5/10/2018	Added 9.3 BSP Management
1.5	08/10/2018	Added 11 Source code
1.6	9/10/2018	Added 10.4 Recovering from a damaged MainOS Added 11.1.2 Deploy
1.7	24/01/2019	Updated device list in 6 Using the serial port 10.4 Recovering from a damaged MainOS revision
1.8	9/04/2020	9.3 BSP Management revision
1.9	22/06/2020	Added 4.1 Kiosk mode Added 9.4 Network Added 9.5 Application management
1.10	1/07/2020	Added 9.6 Other
1.11	30/09/2020	6.2.1 Changing the operating mode revision

1. Introduction

The goal of this document is to describe the relevant software components of the 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 series panels
- JSmart series panels
- eXware series
- PLCM07

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

<https://exorint.com/panel-bsp/>

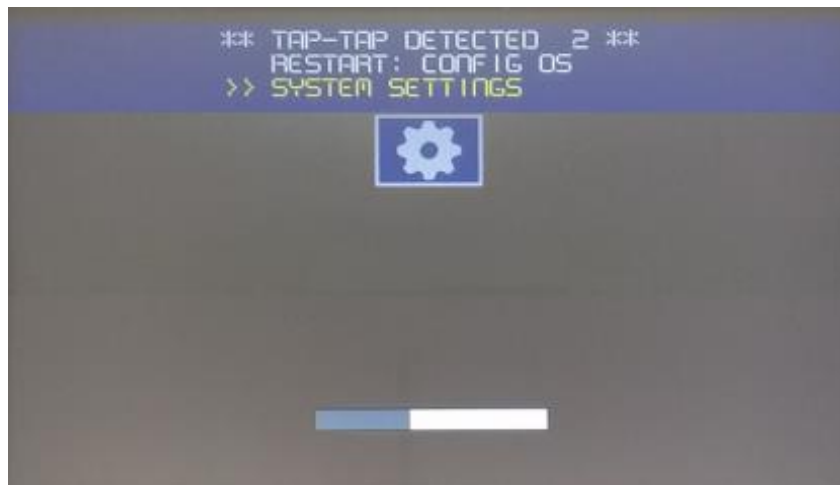
For datasheets, hardware specifications and other technical information, please refer to the hardware section on our website:

<https://exorint.com/product-category/products/#hardware-section>

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:



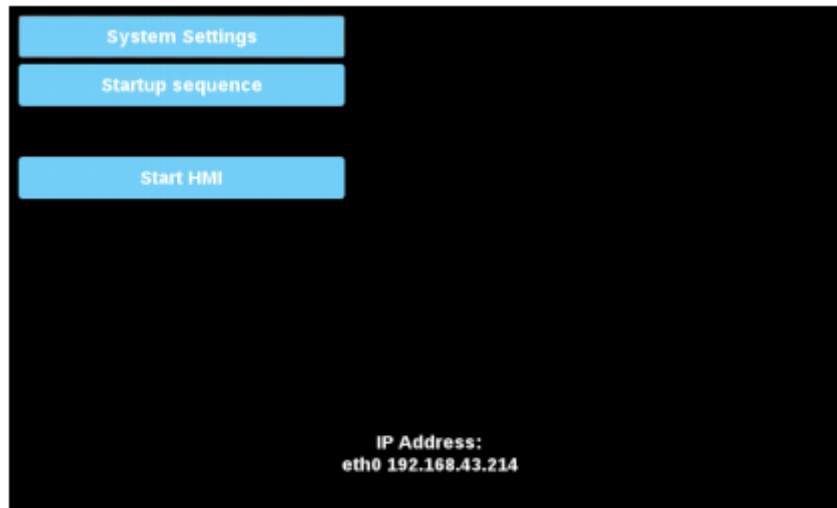
By keeping now the finger on the screen the "RESTART: CONFIG OS" option will remain selected and after 5 seconds the panel will reboot in the ConfigOS recovery mode (see).

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:

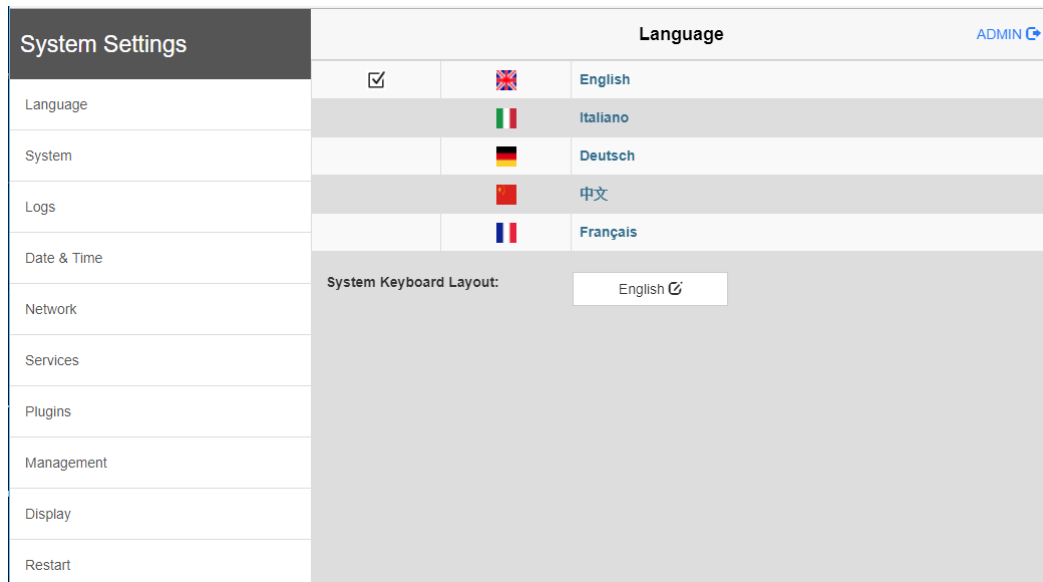
- **System Settings:** Access the panel settings. From here it's possible to read device information and configure and update the BSP (see [3. System Settings](#)).
- **Startup Sequence:** Install, uninstall, update and manage installed applications (see [4. Application Launcher](#))
- **Start HMI:** 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:

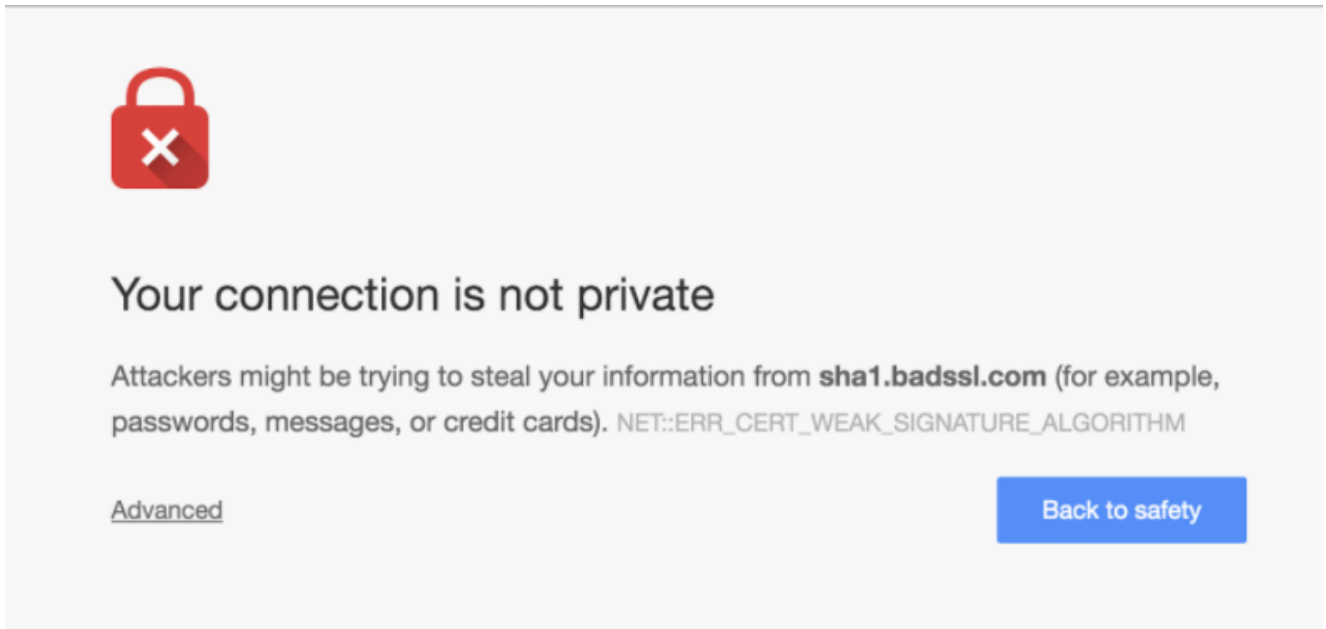
- Locally by interrupting the boot sequence at power on and then pressing the “System Settings” button.
- Remotely from a web browser.



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 interface is divided into several sections:

- Config OS:** A section for managing the configuration operating system.
- Main OS:** A section for managing the main operating system. It displays the following information:

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

 Below this information are three buttons: "Get" with a download icon, "Update" with a refresh icon, and "Check" with a gear icon.
- Settings:** A section for managing device settings.
- Data:** A section for managing device data.
- Splash image:** A section for managing the device splash image.
- FPGA:** A section for managing the device FPGA.

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

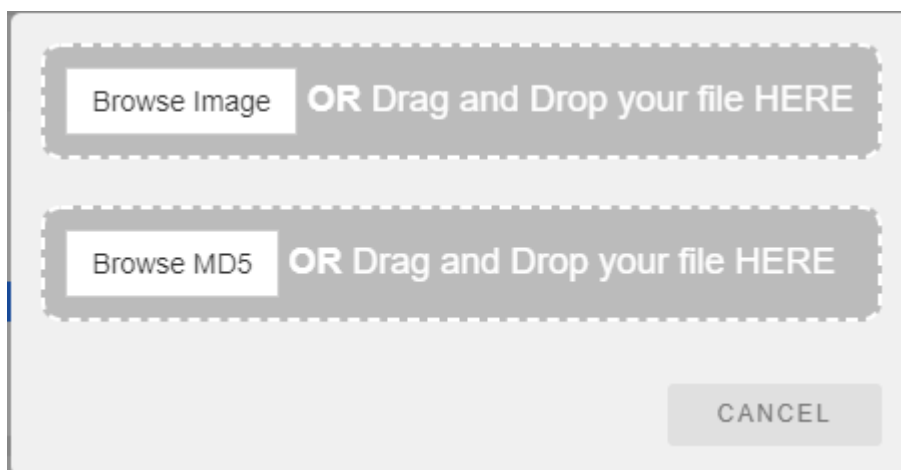
The “Update” action can be used to install a newer version of a BSP component or to restore it using an old backup.

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.

3.2 Services

3.2.1 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](#))

3.2.2 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](#))

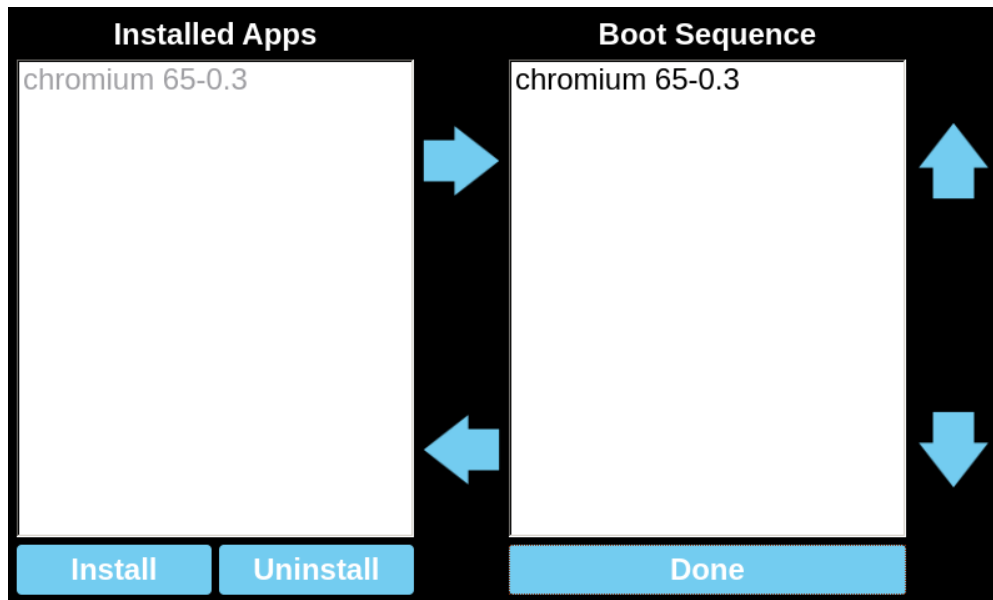
3.2.3 Autorun scripts from external storage

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 Available application packages

Some applications are available for download here:

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

4.3 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.

To get started it's suggested to download a demo application package named hmbrowser.zip which will install a simple web browser on the panel:

```
http://download.exoreembedded.net:8080/Public/ExorPanels/Other/JML_Demo/
```

The zip archive can be extracted and the files used as template, bash scripts have some additional comments that should help understanding what they do.

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. Check the demo files for the details.
- **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.

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/mmcblk1p1)

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/mmcblk1p2)

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/mmcblk1p3)

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/mmcblk1p5)

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 underlying MainOS /etc folder.

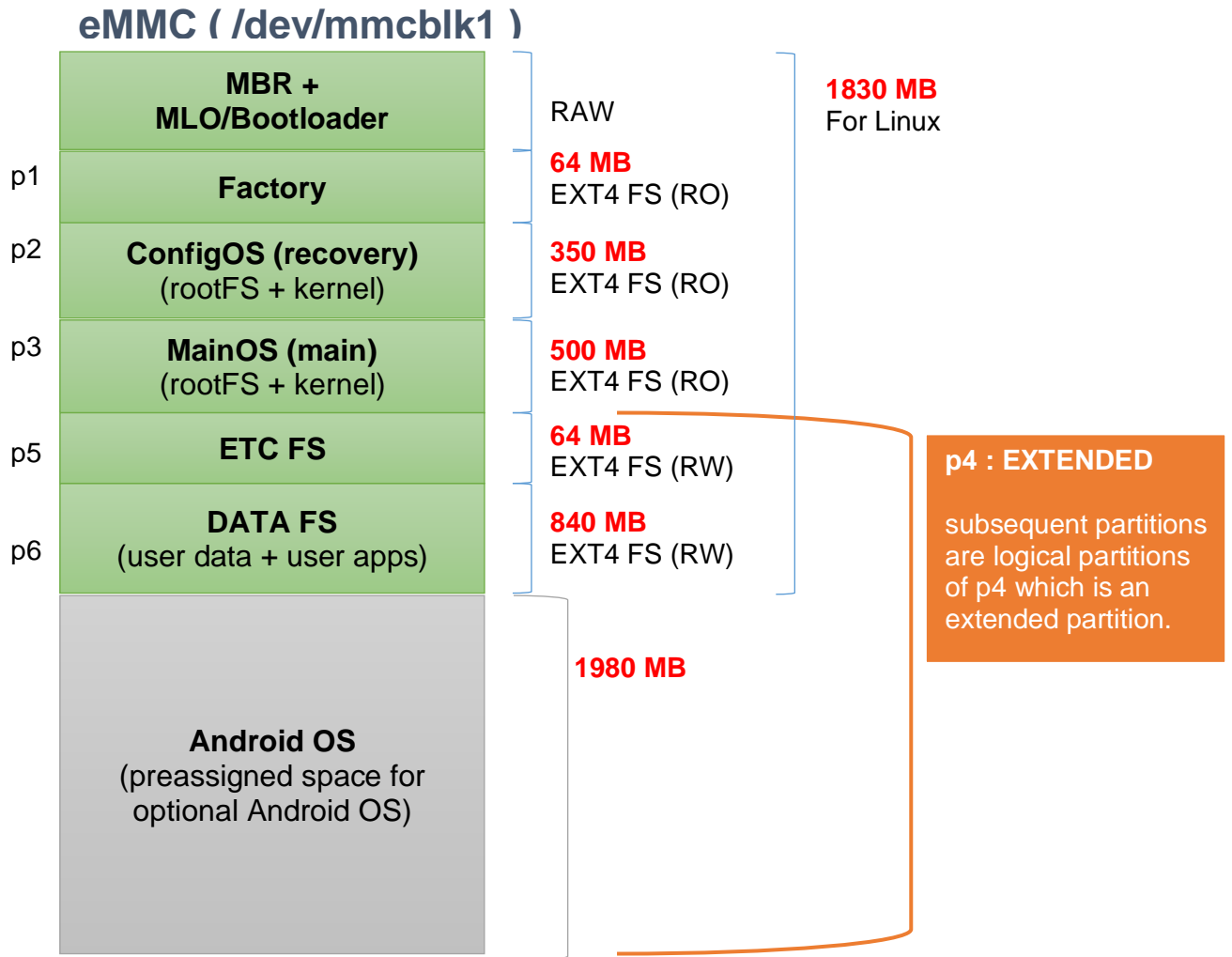
- **Data Partition** (/dev/mmcblk1p6)

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 `/dev/ttymx0`

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.

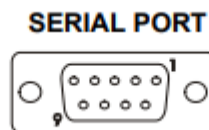
During development the serial port can also be used to get a console (see [10.1 Enabling the serial console](#)).

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	



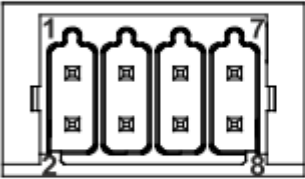
RS-422, RS-485

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

SERIAL PORT



RS-232

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

RS-422, RS-485

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

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, TIOCSRS485, &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 Yocto SDK containing the cross-compiler and all the necessary libraries is available for download here:

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

The current version of the SDK is based on Yocto Dora 1.5.3, uses GCC 4.9 and comes with Qt 4.8 libraries. To use it a Linux machine is required, installation is as simple as executing the SDK installer script:

```
$ chmod +x un60-hsxx-sdk-1.0.4.sh
$ sudo ./un60-hsxx-sdk-1.0.4.sh
```

For further support on how to use the SDK it's also possible to refer to the official Yocto Developer's Guide, in particular chapter 2, "Using the Standard SDK" :

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

8.1 Using the pre-configured VirtualBox VM

You can download the Exor's VirtualBox development VM from here:

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

The virtual machine comes in the OVA (Open Virtualization Archive) format. To import it on VirtualBox got to "File" -> "Import Appliance...", select the downloaded .ova file and then click "Import". At this point VirtualBox will give you the opportunity to customize the VM, double-click on entries to edit them.

You will notice there are two network adapters, one is set to work in NAT mode while the second one works in bridged mode, the virtual machine will always use the bridged interface if possible and fall back to the other only if necessary. Adjust both adapters to work with the real network interface you use to have access to internet. Note that if the bridged adapter is not correctly configured you won't be able to resolve the device hostname, its IP address has to be used in this case.

Appliance settings

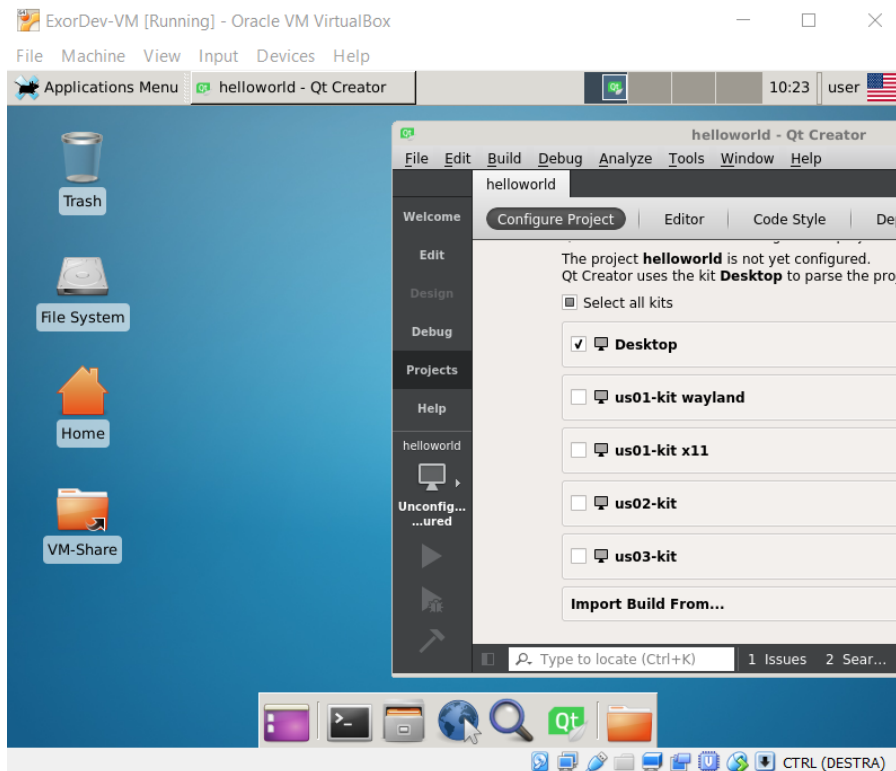
These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Description	Configuration
Virtual System 1	
Name	ExorDev-VM
Guest OS Type	Ubuntu (64-bit)
CPU	2
RAM	2048 MB
USB Controller	<input checked="" type="checkbox"/>
Network Adapter	<input checked="" type="checkbox"/> Intel PRO/1000 MT Desktop (82540EM)
Network Adapter	<input checked="" type="checkbox"/> Intel PRO/1000 MT Desktop (82540EM)
Storage Controller (SATA)	AHCI
Virtual Disk Image	C:\VirtualBox VMs\ExorDev-VM\ExorDev-VM-...

Reinitialize the MAC address of all network cards

Restore Defaults Import Cancel

The default amount of RAM is set to 2GB but if possible we suggest to increase it to at least 4GB, adjusting the number of CPU cores is also a good idea. When done click on “Import”. After importing the box it will be possible to change VM settings again.

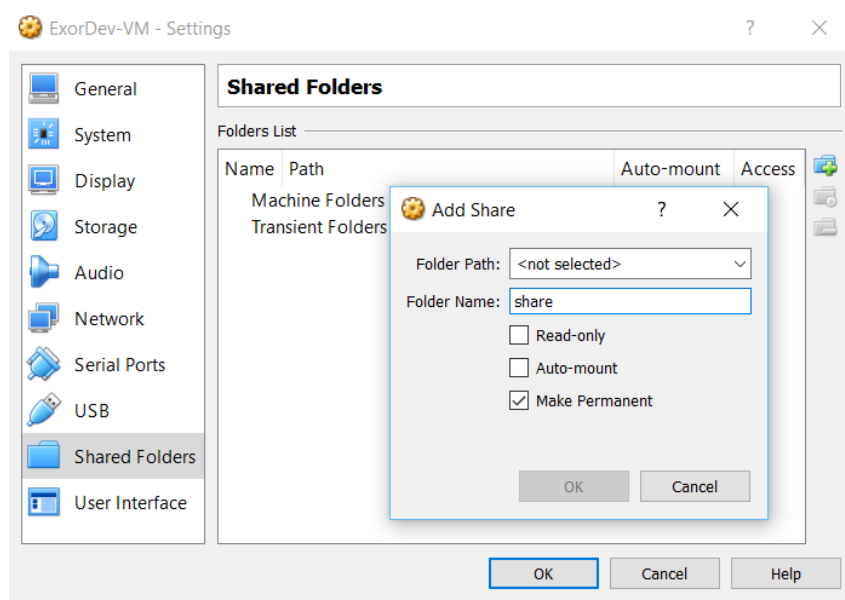


8.1.1 Setup a guest-host shared folder

We recommend configuring a shared folder between host and guest, it's the easiest way to move files from and to the VM. From VirtualBox right-click on Exor's VM and select "Settings...". Now go to "Shared Folders" and click on the add button to the right. Configure as follow:

- **Folder Path:** choose the host folder to share with the virtual machine
- **Folder Name:** must be `share`.
- **Read-only:** leave unchecked.
- **Auto mount:** leave unchecked.
- **Make Permanent:** set checked.

The chosen folder will be available inside the virtual machine from `/home/user/VM-Share`, a link to this location can be also found on the VM's desktop.



8.1.2 Using the ready development environment

If you choose to use our VirtualBox here are some information on how to use the environment. The Linux system used is based on Ubuntu 14.04, the default user is:

- username: **user**
- password: **password**

To run a command with root privileges you can use `sudo`, entering the password is not required.

8.2 Using QtCreator IDE

The provided Virtual Box VM already comes with the QtCreator IDE ready to use. The only configuration step required in this case is to adjust the target device IP address for remote application deployment ([8.2.1 Build configuration](#), step 5).

We are now however going through the complete process of configuring the same environment from scratch. This guide takes version 4.5.x 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:

```
$ source /opt/exorintos/1.5.3/environment-setup-cortexa8hf-vfp-neon-poky-linux-gnueabi
$ qtcreator
```

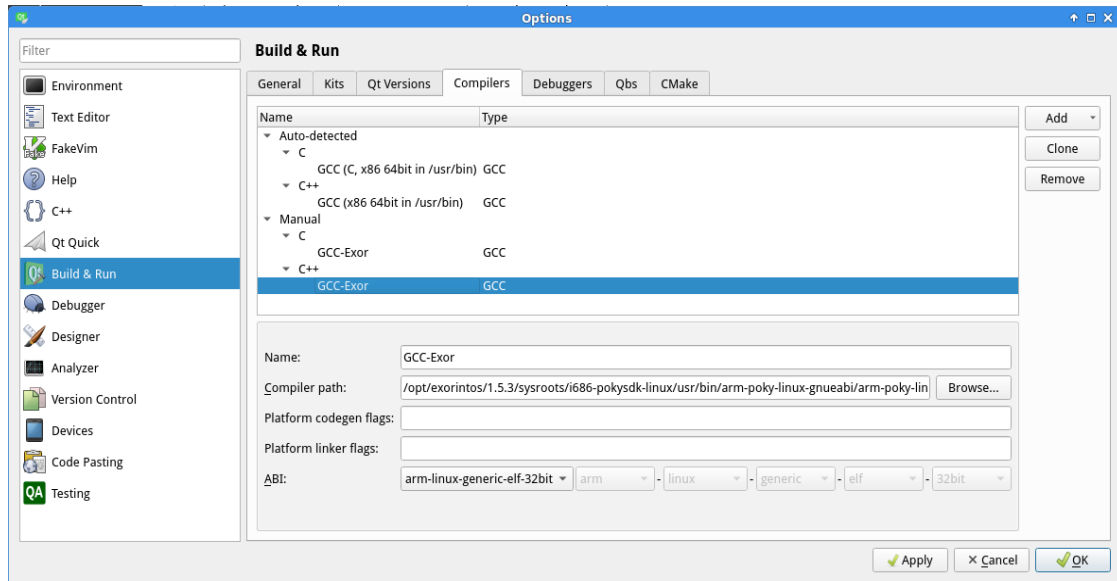
As an alternative it's also possible to create a bash script containing the above two lines to start the IDE more easily as it has been done on the VM.

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

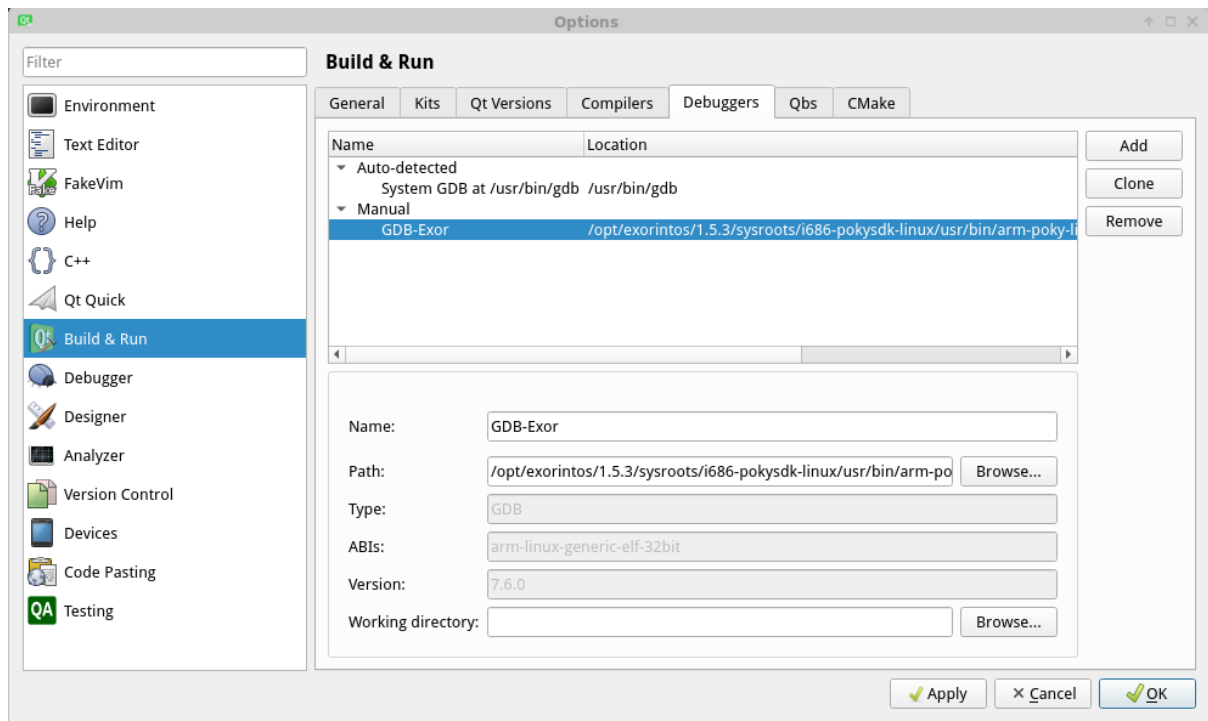
8.2.1 Build configuration

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

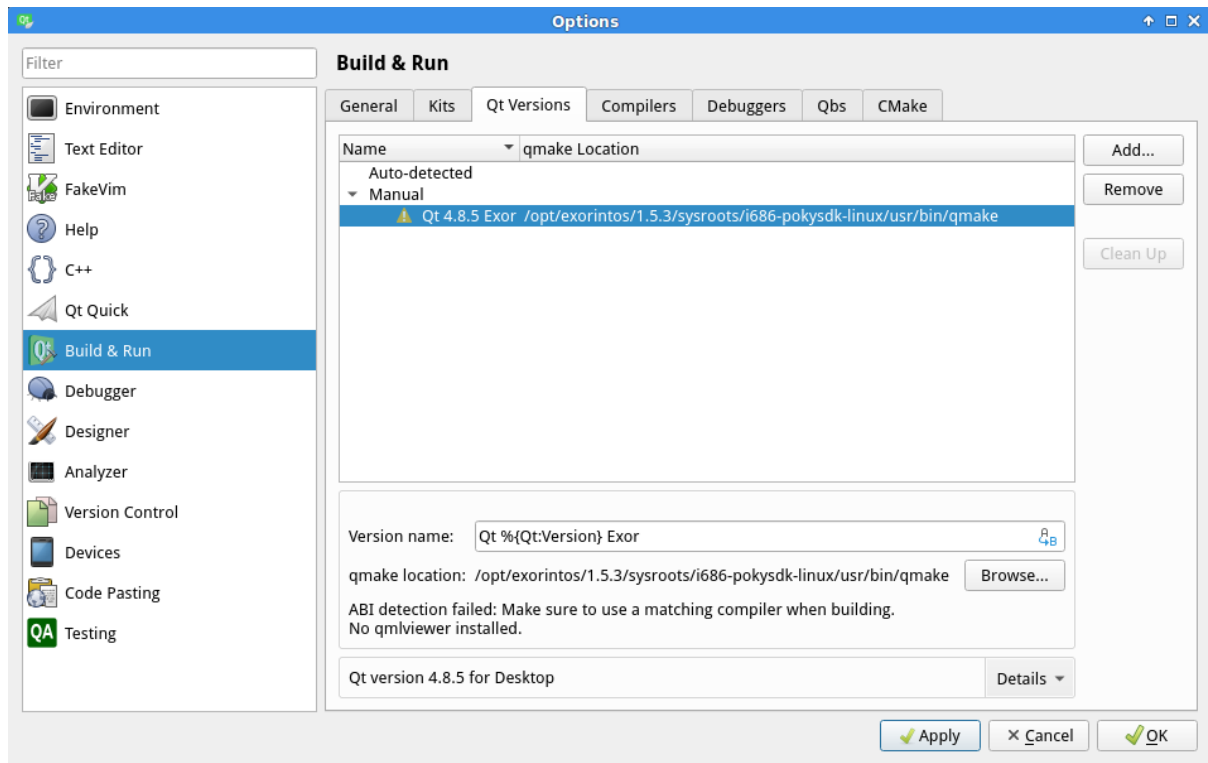
- 1) In the "Compilers" tab click on "Add" -> "GCC" -> "C" and select the cross compiler picking it from the SDK installation folder. If the SDK has been installed in the default location the correct path is: `/opt/exorintos/1.5.3/sysroots/i686-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gcc`.
Optionally edit "Name" to give a more meaningful name for the entry, select "arm-linux-generic-elf-32bit" as ABI and finally click "Apply".
- 2) From the same tab now click "Add" -> "GCC" -> "C++" and select `/opt/exorintos/1.5.3/sysroots/i686-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++` instead. Again, select "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: `/opt/exorintos/1.5.3/sysroots/i686-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb`. Optionally edit “Name”, then click “Apply”.

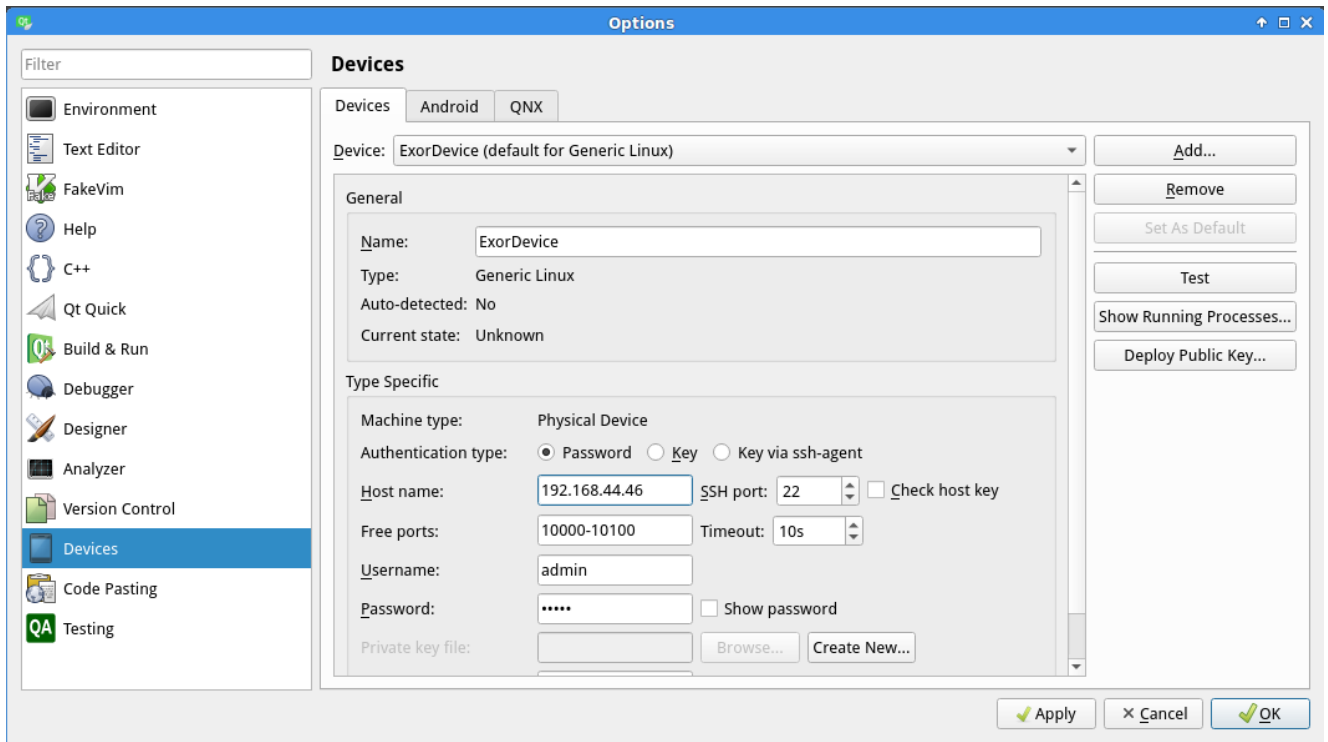


- 4) From “Qt Versions” tab, press “Add..”. The default path to select is: `/opt/exorintos/1.5.3/sysroots/i686-pokysdk-linux/usr/bin/qmake`. QtCreator should automatically recognize the Qt version selected. Press “Apply”.



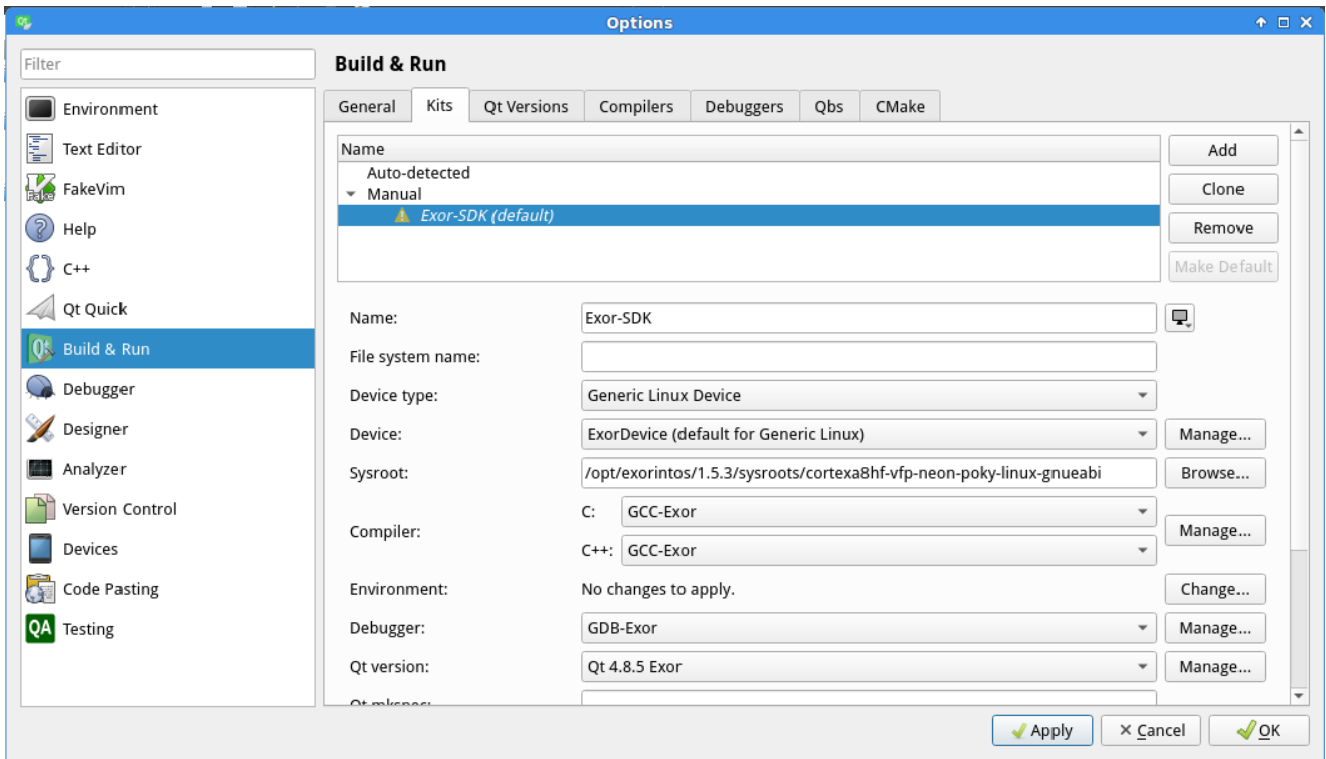
- 5) This step is required for configuring automatic application deployment on the target. Move to the “Devices” section. Here VM users will only need to adjust the IP address and the password if it has been changed from the default. To define a new devices from scratch instead, click on “Add..”, select “Generic Linux Device” and press “Start Wizard”. Fill in these informations:
- **Name:** the device name, for example, `Exor Device`.
 - **Host name:** enter the device IP address.
 - **Username:** `admin`.
 - **Authentication type:** set to “Password”.
 - **Password:** admin’s default password is `admin`.

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:

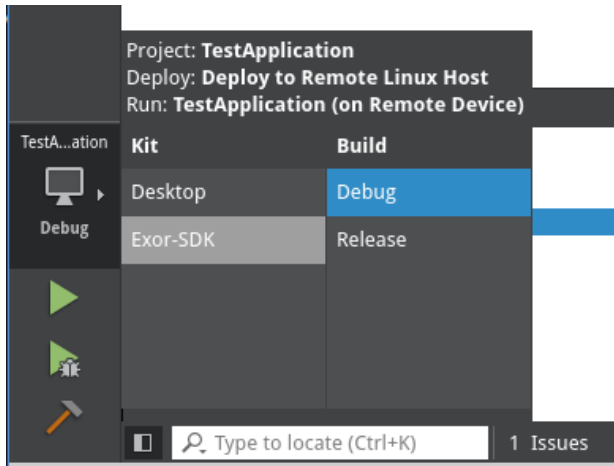
- **Name:** choose a name for the kit.
- **Device Type:** select “Generic Linux Device”.
- **Device:** select the device configured in 5).
- **Sysroot:** if the SDK is installed in the default location, the path to select is:
/opt/exorintos/1.5.3/sysroots/i686-pokysdk-linux/usr/bin/qmake
- **Compiler:** 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).



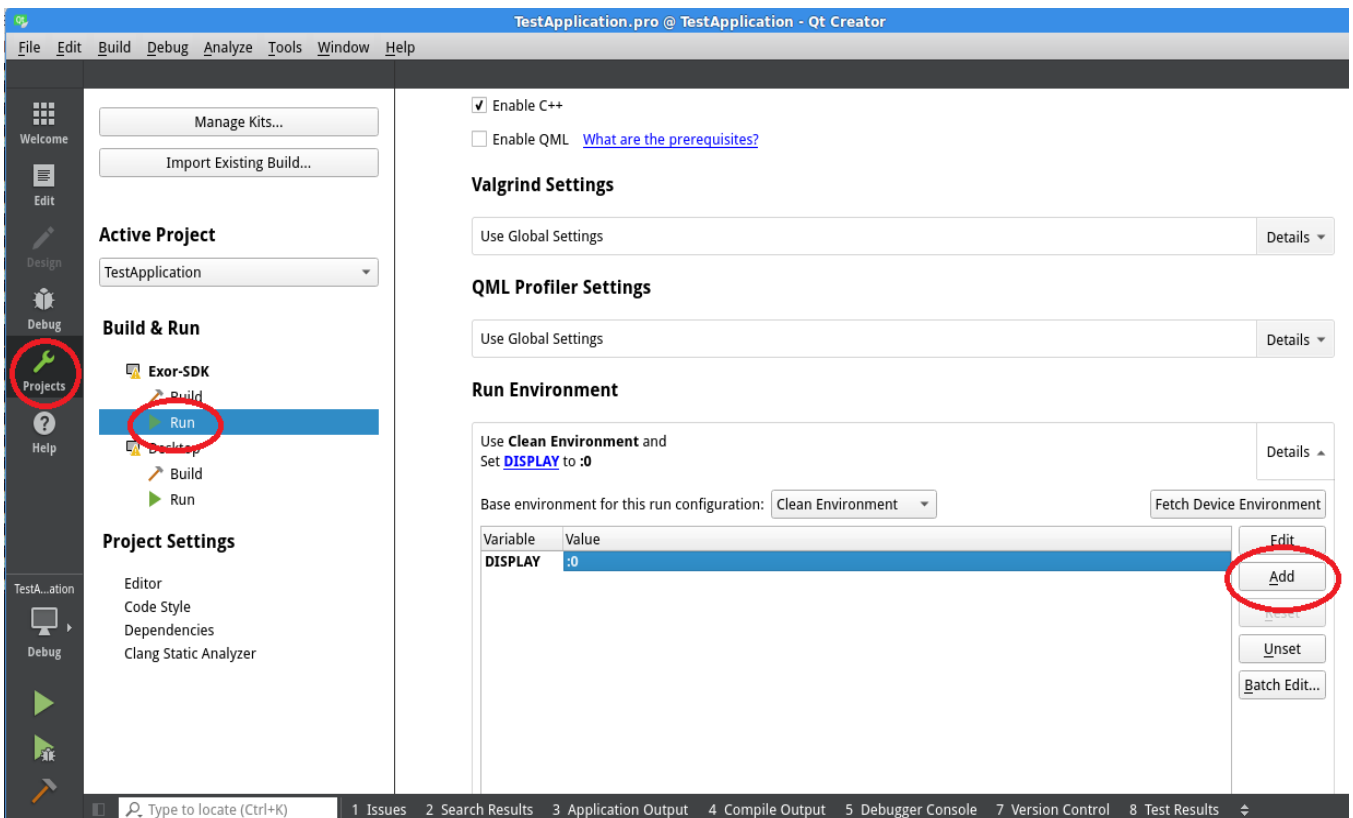
8.2.2 Application deploy

Before starting here, make sure a 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:



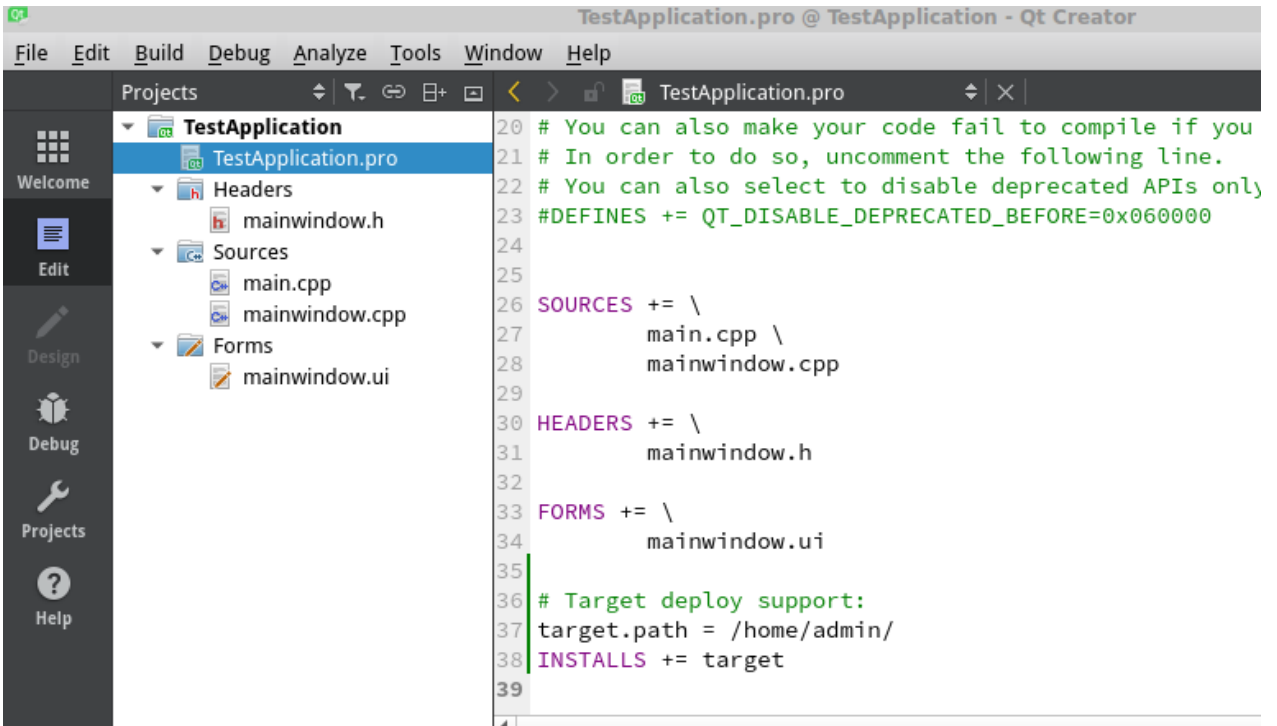
- 3) Click on “Projects” in the menu on the left and then “Run” as shown in the screenshot below. Scroll down to the “Run Environment” options, press “Add” and define a variable `DISPLAY` with value `:0`.



- 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)



- 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 Using the DBus HAL interface

For device configuration, hardware peripherals access and application management the com.exor.EPAD and com.exor.JMLauncher dbus interfaces can be used. Custom applications running on the target can use these for a better integration with the device.

Dbus requests can also be sent using the dbus-send tool. For each method listed below a command line that can be used to call it is also given.

9.1 Display

Object Path: /Backlight

Method: com.exor.EPAD.Backlight.setBacklightTimeout

Input: STR displayName Can be set to empty string "" to use the default
INT32 value Backlight timeout in minutes, 0 = always on [>=0]

Output: INT32 returnCode 0 = success

Description: Sets the inactivity timeout in minutes after which the backlight is turned off. The backlight is automatically turned on again on user interaction.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/Backlight"
com.exor.EPAD.Backlight.setBacklightTimeout string:"" int32:<value>
```

Object Path: /Backlight

Method: com.exor.EPAD.Backlight.backlightTimeout

Input: STR displayName Can be set to empty string "" to use the default

Output: INT32 value Backlight timeout in minutes

Description: Get the currently set inactivity timeout in minutes.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/Backlight"
com.exor.EPAD.Backlight.backlightTimeout string:""
```

Object Path: /Backlight

Method: com.exor.EPAD.Backlight.saveBrightness

Input: INT32 value Brightness level [0-255]
Output: INT32 returnCode 0 = success.
Description: Set backlight brightness, 0 will set it to the minimum level without turning it off.
Dbus-send:

```
dbus-send --print-reply --system --dest=com.exor.EPAD "/Backlight"
com.exor.EPAD.Backlight.saveBrightness int32:<value>
```

Object Path: /Backlight
Method: com.exor.EPAD.Backlight.getBrightness
Output: INT32 value Brightness level [0-255], -1 if backlight is off
Description: Get the current backlight brightness.
Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/Backlight"
com.exor.EPAD.Backlight.getBrightness
```

Object Path: /Backlight
Method: com.exor.EPAD.Backlight.offBrightness
Output: INT32 returnCode 0 = success
Description: Turns off the backlight. Use saveBrightness to turn it on again. After a reboot the backlight will be on again at the last set brightness level.
Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/Backlight"
com.exor.EPAD.Backlight.offBrightness
```

Object Path: /
Method: com.exor.EPAD.Backlight.setScreenOrientation
Input: INT32 value Screen orientation [0, 90, 180, 270]
Output: INT32 returnCode 0 = success.
Description: Set the screen orientation. Requires a reboot for changes to take effect.
Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.setScreenOrientation
int32:<value>
```

Object Path: /

Method: com.exor.EPAD.Backlight.getScreenOrientation

Output: INT32 value Screen orientation [0, 90, 180, 270]

Description: Get the current screen orientation.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.getScreenOrientation
```

Object Path: /

Method: com.exor.EPAD.Backlight.setScreenSaturation

Input: INT32 red Red saturation [0-100]
 INT32 green Green Saturation [0-100]
 INT32 blu Blue Saturation [0-100]

Output: INT32 returnCode 0 = success.

Description: Set screen color saturation. Default values are 100,100,100.
 Supported only on ex707/710/715/721 and JSmart.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.setScreenSaturation
int32:<red> int32:<green> int32:<blu>
```

Object Path: /

Method: com.exor.EPAD.Backlight.getScreenSaturation

Returns: INT32 red Red saturation [0-100]
 INT32 green Green Saturation [0-100]
 INT32 blu Blue Saturation [0-100]

Description: Get current screen color saturation.
 Supported only on ex707/710/715/721 and JSmart.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.getScreenSaturation
```

Object Path: /

Method: com.exor.EPAD.Backlight.setScreenWhiteBalance

Input: INT32 value Screen white balance [-100 - 100]

Output: INT32 returnCode 0 = success.

Description: Set screen white balance. Default value is 0.
Supported only ex707/710/715/721 and JSmart.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/"  
com.exor.EPAD.setScreenWhiteBalance int32:<value>
```

Object Path: /

Method: com.exor.EPAD.Backlight.getScreenWhiteBalance

Output: INT32 value Screen white balance [-100 - 100]

Description: Get current screen white balance.
Supported only on ex707/710/715/721 and JSmart.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/"  
com.exor.EPAD.getScreenWhiteBalance
```

Object Path: /

Method: com.exor.EPAD.Backlight.setScreenHue

Input: INT32 value Screen hue [-100 - 100]

Output: INT32 returnCode 0 = success.

Description: Set screen hue. Default value is 0.
Supported only on ex707/710/715/721 and JSmart.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.setScreenHue  
int32:<value>
```

Object Path: /

Method: com.exor.EPAD.Backlight.getScreenHue

Output: INT32 value Screen hue [-100 - 100]

Description: Get current screen hue. Default value is 0.
Supported only on ex707/710/715/721 and JSmart.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.getScreenHue
```

9.2 Devices control

Object Path: /Buzzer

Method: com.exor.EPAD.Buzzer.beep

Input: INT32 freq Buzzer frequency in Hz
INT32 time Duration in ms

Output: INT32 returnCode 0 = success.

Description: Turns on the buzzer for *time* ms at *freq* frequency.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.Buzzer.beep
int32:<freq> int32:<time>
```

Object Path: /Sensors

Method: com.exor.EPAD.Sensors.accelerometerAxis

Output: DOUBLE AccelX X axis in g units
DOUBLE AccelY Y axis in g units
DOUBLE AccelZ Z axis in g units

Description: Get the 3 axes values from the accelerometer if available.
Supported only on JSmart.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/Sensors"
com.exor.EPAD.Sensors.accelerometerAxis
```

Object Path: /Sensors

Method: com.exor.EPAD.Sensors.temperature
Input: INT32 id Id of the temperature sensor [≥ 0]
Output: INT32 temp Temperature in Celsius degrees /10
Description: Read temperature form the specified sensor.
 Supported only on JSmart.
Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/Sensors"
com.exor.EPAD.Sensors.temperature
```

9.3 BSP Management

These APIs can be used to manage BSP components. All the operations below are performed asynchronously.

Object Path: /
Method: com.exor.EPAD.downloadImage
Input: STR imgType BSP component to update
 STR pkgFile Path of the update package file (*.tar.gz)
 STR md5File Path of the md5 file (*.tar.gz.md5)
 BOOL md5Check Whether to verify the package md5 using *md5File*
 STR user Username to force an update
 STR paswd Password to force an update
 BOOL reboot Whether the device can reboot during the update
Description: Start a BSP component update. Valid *imgType* values are:
mainos, configos, user (data partition), *etc* (settings partition), *bootloader,*
xloader, fpga and splash

md5File is a text file containing the package md5 checksum. If *md5Check* is set to false, *md5File* can be set to an empty string.

If known, *user* and *paswd* can be set to force update operations that normally would not be allowed. This includes BSP components downgrades and updates to versions detected to be not compatible with the hardware.

On all the other cases these two values can be set to an empty string.

reboot can be set to true if the system is allowed to reboot the device to complete the update. Some update operations require this parameter to be true otherwise the process will fail immediately:

- MainOS update if the system is currently in MainOS
- ConfigOS update if the system in currently in ConfigOS

Settings (etc) update if the system is currently in MainOS
 Data (user) update if the system is currently in MainOS

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.downloadImage
string:"<imgType>" string:"<pkgFile>" string:"<md5File>" boolean:<md5Check> string:"<user>"
string:"<paswd>" boolean:<reboot>
```

Object Path: /

Method: com.exor.EPAD.uploadImage

Input:

STR	imgType	BSP component to update
STR	outFile	Path for the output file
BOOL	md5Check	Currently unused

Description: Creates a backup of a BSP component. Valid *imgType* values are:
mainos, configos, data, settings, bootloader, xloader, fpga and splash

The *md5Check* parameter is currently unused. To maintain the current behavior on future BSP versions this should be set to false for now.

Settings and data backup can be performed only while the system is in ConfigOS.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.uploadImage
string:"<imgType>" string:"<outFile>" boolean:<md5Check>
```

Object Path: /

Method: com.exor.EPAD.formatImage

Input:

STR	imgType	BSP component to update
BOOL	reboot	Whether the device can reboot during the update

Description: Clears or restores to defaults a BSP component. Valid *imgType* values are:
data, settings, splash

If called for the settings *imgType* the device configuration is restored to factory defaults. Settings restore and data clear require a device reboot if the system is currently in MainOS, in these cases *reboot* needs to be set to true.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD "/" com.exor.EPAD.formatImage
string:"<imgType>" boolean:<reboot>
```

9.4 Network

Object Path: /NetworkManager

Method: com.exor.EPAD.NetworkManager.getConfigJSON

Output: STR config Network configuration in JSON format

Description: Returns all device network related information available in JSON format. See [9.4.1 Network JSON object specification](#)

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD.NetworkManager "/NetworkManager"
com.exor.EPAD.NetworkManager.getConfigJSON
```

Object Path: /NetworkManager

Method: com.exor.EPAD.NetworkManager.setConfigJSON

Input: STR config Network configuration in JSON format

Description: Change network parameters as specified in the *config* JSON object string. Only writable parameters can be specified in the object, see [9.3.1 Network JSON object specification](#). The configuration would take effect only after a reboot, to apply immediately call `applyNewConfiguration`.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD.NetworkManager "/NetworkManager"
com.exor.EPAD.NetworkManager.getConfigJSON string:'<config>'
```

Examples of valid <config> JSON strings:

Set eth0 interface in DHCP mode:

```
'{ "interfaces" : [ { "name" : "eth0", "dhcp" : true } ] }'
```

Set eth0 static and eth1 in DHCP mode:

```
'{ "interfaces" : [ { "name" : "eth0", "dhcp" : false, "netmask" : "255.255.0.0", "ip_address" : "192.168.20.50"
}, { "name" : "eth1", "dhcp" : true } ] }'
```

Set device hostname:

```
'{ "hostname" : "HMI-Panel" }'
```

Enable bridge between eth0 and eth1 and set bridged interface in DHCP mode:

```
'{ "bridge" : { "enabled" : true, "list" : [ "eth0", "eth1" ] }, "interfaces" : [ { "name" : "br0", "dhcp" : true } ] }'
```

Disable bridge:

```
'{ "bridge" : { "enabled" : false } }'
```

Object Path: /NetworkManager

Method: com.exor.EPAD.NetworkManager.applyNewConfiguration

Input: BOOL async Execute operation asynchronously

Description: Reconfigure network parameters, applies a new configuration if changed. Method returns immediately if *async* is true

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.EPAD.NetworkManager "/NetworkManager"
com.exor.EPAD.NetworkManager.getConfigJSON boolean:<async>
```

9.4.1 Network JSON object specification

Following is the full JSON object structure as returned by the getJSONConfig method. The same structure needs to be followed when the object is passed to the setJSONConfig but in this case only writable parameters can be specified.

```
{
  "version" : int,
  "hostname" : "string",
  "dns" : {
    "servers" : [array/string],
    "search" : [array/string]
  },
  "bridge" : {
    "enabled" : boolean,
    "hw" : boolean,
    "interfaces" : [array/string],
    "list" : [array/string]
  },
}
```

```

    "wifi" : {
        "interfaces" : [array/interfaceObj]
    },
    "interfaces" : [array/interfaceObj]
}

```

Parameter	Type	Writeable	Description
version	String	No	API version. Currently set to 0
hostname	string	Yes	Device hostname
dns.servers	array/string	Yes	DNS server IPv4 addresses. Ex: ["192.168.2.200", "8.8.8.8"]
dns.search	array/string	Yes	Search domains
bridge.enabled	Boolean	Yes	When enabled the bridge interface "br0" will be available for configuration
bridge.hw	Boolean	No	Whether the device has hardware support to work as a network switch
bridge.interfaces	array/string	No	List of interfaces supporting bridge
bridge.list	array/string	Yes	List of bridged interfaces if bridge is enabled
wifi.interfaces	array/interfaceObj	Yes	List of wifi interfaces
interfaces	array/interfaceObj	Yes	List of ethernet interfaces

Finally the *interfaceObj* object structure:

```

{
    name" : "string",
    "mac_address" : "string",
    "ip_address" : "string",
    "netmask" : "string",
    "dhcp" : boolean,
    "configured" : boolean,
    "readonly" : boolean,
    "virtual" : boolean,
    "hidden" : boolean,
    "actual_netmask" : "string",
    "actual_ip_address" : "string"
}

```

Parameter	Type	Writeable	Description
name	string	Yes	Interface name. Ex. "eth0". In setConfigJSON this is specified as a selector for the interface to configure

mac_address	string	No	MAC address. Ex. ""
ip_address	string	Yes	Configured IPv4 address. Ex. "192.168.2.200"
netmask	string	Yes	Configured netmask. Ex. "255.255.0.0"
dhcp	boolean	Yes	Whether DHCP is used
configured	boolean	No	Whether
Readonly	boolean	No	Whether this interface configuration can be changed
Virtual	boolean	No	Whether this is a physical interface
Hidden	boolean	No	Whether this is can be managed from system settings network configuration GUI
actual_netmask	string	No	Current netmask used by the interface
actual_ip_address	string	No	Current IPv4 address used by the interface

9.5 Application management

The application launcher also provides a dbus interface in order to allow to manage software installed on the device.

Object Path: /

Method: com.exor.JMLauncher.infoJSON

Output: STR info Application information in JSON format

Description: Returns all application related information available in JSON format. See launcher info structure in [9.5.1 Application JSON object specification](#)

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/" com.exor.JMLauncher.infoJSON
```

Object Path: /

Method: com.exor.JMLauncher.install

Input: STR pkgPath Absolute path to the package file

Optional:

BOOL async Execute operation asynchronously (default: false)

INT position Application position after installing (default: -2)

Output: INT32 returnCode 0 = success.

Description: Install the application package found in *pkgPath*, if *async* is true the method returns immediately. *position* can have following values:
= 0 Don't add the application to startup sequence

- =-1 Add at the bottom of the startup sequence
- =-2 Auto. Adds to startup sequence only if it's empty
- > 0 Add to startup sequence at the specified position

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/"
com.exor.JMLauncher.install string:"<pkgPath>" boolean:<async> int32:<position>
```

Signals:

During package installation dbus signals will be generated on the com.exor.JMLauncher interface to give updates on the status of the operation:

installationSuccess	Installation was successfully completed
installationFailed	Installation failed
installationCanceled	Installation canceled from local GUI by user
installationStatus(INT)	Installation progress in percentage
licenseAgreementRequest	The application is requiring the user to accept the license terms, installation will resume after receiving the input

Object Path: /

Method: com.exor.JMLauncher.uninstall

Input: STR appName Application name

Output: INT32 returnCode 0 = success.

Description: Uninstall application *appName*

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/"
com.exor.JMLauncher.uninstall string:"<appName>"
```

Object Path: /

Method: com.exor.JMLauncher.add

Input: STR appName Application name

Optional:

INT32 position Application position (default = -1)

Output: INT32 returnCode 0 = success.

Description: Add application *appName* into startup sequence at position *position*:
 = -1 Add at the bottom of the startup sequence

>0 Add to startup sequence at the specified position

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/" com.exor.JMLauncher.add
string:"<appName>" int32:<position>
```

Object Path: /

Method: com.exor.JMLauncher.remove

Input: STR appName Application name

Output: INT32 returnCode 0 = success.

Description: Remove application *appName* from startup sequence

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/" com.exor.JMLauncher.remove
string:"<appName>"
```

Object path /

Method: com.exor.JMLauncher.isKiosk

Output: BOOL kiosk Whether the device is in kiosk mode

Description: Returns true if the device is in “kiosk” mode (see [4.1 kiosk mode](#))

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/" com.exor.JMLauncher.isKiosk
```

Object path /

Method: com.exor.JMLauncher.exitKiosk

Input: BOOL save Whether to apply to next boot

Description: Exit kiosk mode (see [4.1 kiosk mode](#)). If *save* is true the very next boot the device will start with kiosk mode disabled.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/"
com.exor.JMLauncher.exitKiosk boolean:<save>
```

Object path /
Method: com.exor.JMLauncher.enterKiosk
Input: BOOL save Whether to apply to next boot
Description: Enter kiosk mode (see [4.1 kiosk mode](#)). If *save* is true the very next boot the device will start with kiosk mode enabled (can undo a `exitKiosk` call).
Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/"
com.exor.JMLauncher.enterKiosk boolean:<save>
```

Object path /
Method: com.exor.JMLauncher.startApp
Input: STR appName Application name
Description: Start application *appName* if not already executing. The application is not added in the startup sequence
Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/" com.exor.JMLauncher.startApp
string:"<appName>"
```

Object path /
Method: com.exor.JMLauncher.quitApp
Input: STR appName Application name
Description: Stop application *appName* if not already executing. The application is not removed from startup sequence. Remember that if the device is in kiosk mode and all the running applications are stopped the device will reboot (see [4.1 kiosk mode](#)).
Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/" com.exor.JMLauncher.quitApp
string:"<appName>"
```

Object path /

Method: com.exor.JMLauncher.restartApp

Input: STR appName Application name

Description: Restart application *appName*.

Dbus-send:

```
$ dbus-send --print-reply --system --dest=com.exor.JMLauncher "/" com.exor.JMLauncher.quitApp
string:"<appName>"
```

9.5.1 Application JSON object specification

Following is the full JSON object structure as returned by the JMLauncher's infoJSON method:

```
{
  "kiosk" : bool,
  "status": int.
  "apps" : [array/applicationObj]
}
```

Parameter	Type	Description
kiosk	bool	True if the device is in kiosk mode
status	int	Status code
Apps	array/applicationObj	List of installed applications

Status can have one of following values:

Name	Value	Description
MAINWINDOW	-1	The configuration menu is shown
LOADING	0	Applications are being launched
LOADED	1	Startup sequence completed
DOWNLOAD	4	A package is being downloaded on the device
DOWNLOAD_FINISHED	5	Package downloaded
DOWNLOAD_FAILED	6	Package download failed
INSTALL	7	An application is being installed
INSTALL_FINISHED	8	Install completed successfully
INSTALL_FAILED	9	Install failed
INSTALL_FAILED_NO_SPACE	10	Install failed, not enough disk space
INSTALL_CANCELING	11	User requested to cancel installation
INSTALL_CANCELED	12	Install canceled

UNINSTALLING	13	An application is being uninstalled
UNINSTALLING_PREVIOUS	14	Removing old application before update
UNINSTALLING_FINISHED	15	Application removed
INCOMPLETE_INSTALL	16	Cleaning an incomplete
UPDATE	17	An application is begin updated
UPDATE_FINISHED	18	Application updated
UPDATE_FAILED	19	Update failed
UPDATE_FAILED_NO_SPACE	20	Update failed, not enough disk space
INSTALL_LICENSE	22	User is required to accept license terms
RESTARTING	23	Device needs to be rebooted

ApplicationObj JSON structure:

```
{
    "name" : string,
    "version" : string,
    "enabled" : bool,
    "folder" : string,
    "order" : int,
    "running" : bool
}
```

Parameter	Type	Description
name	string	Application name
version	string	Application version
enabled	bool	True if application is included in the startup sequence
folder	string	Installation folder name. The application is installed in /mnt/data/hmi/<folder>
order	int	Position in the startup sequence (0 if disabled)
running	bool	True if the application is running on the device

9.6 Other

Object path	/		
Method:	com.exor.FileBrowser.getOpenFileName		
Input:	STR	dir	Directory to show (default "/")
	ARR_STR	fileFilters	List of file types filters
	ARR_STR	nameFilters	List of file name filters
	STR	message	Dialog message

ARR_STR	btnLabels	Labels for buttons (default “Ok” and “Cancel”)
STR	stylesheet	CSS stylesheet

Description: Opens a fullscreen filebrowser dialog showing files in the *dir* directory, the *message* string is shown on top. The user is allowed to browse the filesystem, press the “Cancel” button to close the dialog or select a file and press “Ok”. Button labels can be changed specifying the two strings in the *btnLabels* array. It is possible to restrict files the user is able to select:

- *fileFilters* can list any value found in Qt “QDir::Filter” enum (<https://doc.qt.io/qt-5/qdir.html#Filter-enum>), “QDir:” should be omitted (es. “AllDirs”, “Files”).
- *nameFilters* can contain any string that the name of the file needs to match, use of the wildcard * is allowed (es “*.zip” selects files with name ending with “.zip”)

stylesheet can be used to change how dialog looks by using Qt stylesheet syntax (<https://doc.qt.io/Qt-5/stylesheet-syntax.html>). To separately target the “Ok” and “Cancel” buttons it’s possible to use “#okButton” and “#cancelButon” selectors.

Dbus-send:

```
$ dbus-send --print-reply --session --dest=com.exor.FileBrowser "/"
com.exor.FileBrowser.getOpenFileName string:"<dir>" array:string:<fileFilters>
array:string:<nameFilters> string:"<message>" array:string:<btnLabels>
string:"<stylesheet>"
```

Examples:

Open the filebrowser in “/mnt/data” and allow the user to choose an archive file to extract. Buttons background color is changed using CSS.

```
$ dbus-send --print-reply --session --dest=com.exor.FileBrowser "/"
com.exor.FileBrowser.getOpenFileName \
string:"/mnt/data" \
array:string:"AllDirs","Files","NoDot" \
array:string:"*.zip","*.rar","*.tar.gz","*.7z" \
string:"Please select an archive file to extract:" \
array:string:"Extract","Cancel" \
string:"#cancelButton,#okButton { background: #73CCF0; }"
```

Signals:

When the dialog is closed a signal is sent on the com.exor.FileBrowser interface:

done(String) Returns path of the file selected or empty string if operation in canceled

Object path	/		
Method:	com.exor.FileBrowser.getSaveFileName		
Input:	STR	dir	Directory to show (default "/")
	STR	fileName	Default file name
	BOOL	warnOnExist	Warn the user if files already exists
	STR	message	Dialog message
	ARR_STR	btnLabels	Labels for buttons (default "Ok" and "Cancel")
	STR	stylesheet	CSS stylesheet

Description: Opens a fullscreen filebrowser dialog the *dir* directory, the *message* string is shown on top. The user is allowed to press the "Cancel" button to close the dialog or browse to the destination directory, choose a file name (proposed name will be *fileName*), and press "Ok". If *warnOnExist* is true and the selected file already exists the user will be asked if it's ok to overwrite it.
Button labels can be changed specifying the two strings in the *btnLabels* array.
stylesheet can be used to change how dialog looks by using Qt stylesheet syntax (<https://doc.qt.io/Qt-5/stylesheet-syntax.html>).

Dbus-send:

```
$ dbus-send --print-reply --session --dest=com.exor.FileBrowser "/"
com.exor.FileBrowser.getSaveFileName string:"<dir>" string:"<fileName>"
boolean:<warnOnExist> string:"<message>" array:string:<btnLabels> string:"<stylesheet>"
```

Signals:

When the dialog is closed a signal is sent on the com.exor.FileBrowser interface:

done(String) Returns path of the file to save or empty string if operation in canceled

10 Other useful information

10.1 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
```

10.2 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.

10.3 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.

10.4 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.

11 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>

11.1 Building the Linux kernel

11.1.1 Source code and configuration

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

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

11.1.2 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](#)).

11.2 Building the U-Boot bootloader

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

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

11.3 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
```