CCpilot XM2 and CrossCore XM2

Programmer's Guide

maximatecc

www.maximatecc.com

Contents

Revision history

Rev	Date	Comments
0.1	2014-08-19	First draft
0.2	2014-10-08	Second draft

1. Introduction

1.1. Purpose

Developing applications for CCpilot XM2 and CrossCore XM2 is basically the same as developing any application under Windows or Linux.

This document contains device specific reference information describing application development and APIs used when developing applications for the XM2 device hardware.

These devices are available with both Windows and Linux operating system and this guide is applicable for both operating systems. Operating system specific information is pointed out in the text or addressed in the respective operating system sections.

Several functionalities are available using operating system or de-facto standard APIs. These may be briefly mentioned but not further described within this documentation.

A good prior understanding of Windows and/or Linux programming is needed to fully benefit from this documentation.

1.2. Conventions and defines

CCpilot XM2 and CrossCore XM2 are in most cases identical in functionality and usage. The following definition is used to separate unit specific details. The observe symbol is also used to highlight such difference.

Defines	Use
CCpilot XM2	Information that is specific for CCpilot XM2
CrossCore XM2	Information that is specific for CrossCore XM2
XM2 device	Information that applies to both CCpilot XM2 and CrossCore XM2

The observe symbol is used to highligt information in this document, such as differences between product CCpilot and CrossCore product models.

The exclamation symbol is used to highlight important information.

Text formats used in this document.

Format	Use
Italics	Paths, filenames, definitions.
Bold	Command names and important information

1.3. Identification

On the side of the XM2 device there is a label with version and serial numbers which identify your unique computer. Take note of them. During service and other contact with the supplier it is important to be able to provide these numbers.

1.4. References

For further information on XM2 device software and the available APIs see the following references.

[1] CCpilot XM2 and CrossCore XM2 – Software User Guide

[2] SocketCAN from the Linux kernel, http://lxr.linux.no/linux+v2.6.31.14/Documentation/networking/can.txt

[3] CC AUX API documentation (CC AUX 2.x.x.x, available in the CC AUX SDK)

[4] CAN Interface Description (available within the Windows SDK)

1.5. Include files and libraries

The programmers guide may contain references to include files needed when programming the XM2 device. Note that these files can be downloaded via the maximatecc support web site as development packages. Those packages may also include libraries to use for application development, or in other ways reference to library functions.

2. Interface overview

This section covers basic information on how to access the XM2 device hardware. Most of the hardware is accessed using the default Windows or Linux interfaces but some XM2 device specific interfaces, such as CAN and Digital In, require additional interfaces to be accessed.

The table below lists the API used to access each interface. These interfaces can be grouped into four categories. Standard libraries (Standard API), CC AUX library (CC AUX API), CGOS Library, and Other Libraries (Other API).

Depending on product model, all interfaces may not be present on your specific model. See also the operating system specific sections and additional documentation describing the software API.

Functionality	Standard API	CC AUX API	CGOS API	Other API	Comment
CAN	√			√	Depending on operating system.
Ethernet	√				
USB	√				
RS232	√				
Video In		√			
Audio In / Out	√				
Digital In		√			
Status LED		√			
Backlight		√			
Ambient Light sensor		√			
Buzzer		√			
Watchdog			√		
Power management		√			
S.M.A.R.T		√			
User EEPROM		√			

2.1. Standard Libraries

Most interfaces are accessed using standard libraries and access methods. Different access methods can be possible depending on development environment and additional installed frameworks, such as .Net or Qt.

The standard libraries used for Windows and Linux are described in their respective documentation sources, such as MAN pages or MSDN.

2.2. CC AUX library

The CC AUX API gives access to several hardware specific interfaces. The API is the same for both Windows and Linux, and in generally equal to the CC AUX API found in other devices from maximatecc. The API functions of this library are documented in the CC AUX API reference documentation, called CC AUX, listed as reference [3]. Below is a brief introduction on the API's found therein and their function.

Most API functions can be used from the CCsettings program as well.

Not all API functions are available in all product instances, and will in those cases return a defined error code.

2.2.1. About

Contains an API for reading hardware configuration, unit data etc.

2.2.2. Adc

Contains an API for reading built in ADC voltage information.

2.2.3. AuxVersion

Contains an API for reading firmware version information.

2.2.4. Backlight

Contains an API for controlling backlight settings as well as configuring automatic backlight functionality on CCpilot XM2.

2.2.5. Buzzer

Contains an API for controlling the built-in buzzer.

2.2.6. CanSetting

Contains an API for controlling CAN settings. Note that other or similar CAN-related settings are available through other API's and in the Windows registry.

2.2.7. Config

Contains an API for controlling internal and external power up and power down settings and time configurations, including power button and on/off signal configurations.

2.2.8. Diagnostic

Contains API's for getting run time information about the XM2 device.

2.2.9. DigIO

Contains an API to get the current status of the digital input signals.

2.2.10. FirmwareUpgrade

Contains an API that allows for updating of different firmware components of the XM2 device.

Consider careful usage for these functions, erroneous usage can result in a non-functional XM2 device.

2.2.11. FrontLED

Contains an API for overriding the default LED behaviour.

2.2.12. LightSensor

Contains an API for reading the light sensor values and getting raw and/or calculated values.

2.2.13. Power

Contains an API for reading power status and control functions for advanced shut down behaviour.

2.2.14. PowerMgr

Contains an API for delaying OS shutdown or suspend until the application is ready.

2.2.15. Smart

Contains an API for reading the remaining expected lifetime of the compact flash device and limited device information. The API utilizes the device S.M.A.R.T. data in the background.

2.2.16. TouchScreen

Contains an API for changing the touch screen profile between mouse or touch profile as well as other touch screen related settings on CCpilot XM2.

2.2.17. Video

Contains an API for controlling the analogue video streams in terms of location, size, scaling etc.

2.3. CC AUX API calling convention

The way to call different CC AUX API functions is exemplified here. Please adhere to this calling convention

```
/* Usage in CCaux API 2.x */
#include "Module.h"

MODULEHANDLE pModule = crosscontrol::GetModule();

eErr err = Module_function_1(pModule, arg, …);

Module_release(pModule);
```

2.4. CGOS Library

The CGOS Library/API is a third party library that gives access to CPU board specific functionality such as the watchdog, temperature and user EEPROM access.

2.5. Other Libraries

Custom or third party API may be required to access different interfaces. See the library overview table for information on the specific library.

3. Guidelines for CFast usage

A CFast flash memory is used for persistent storage in the XM2 device. CFast flash memories provide a small sized storage that is nearly insensitive for shocks and vibrations. But flash memories also have a limited number of write cycles that must be considered during application design. As with any file system improper shut down of the device may also lead to incomplete file operations and corrupt flash.

Here are some guidelines that can be used to better assure that the file systems are being kept consistent, and to prevent pre-mature aging of the CFast card.

3.1. Prevention for device shut down

- To prevent data loss due to sudden power disruption, large files shouldn't be written if there is a risk for abruption. Keep critical windows short, i.e. compress the files before writing to storage media if they are retained in RAM memory prior to the write, or divide the files into smaller pieces.
- The XM2 device doesn't deviate from the standard OS model in case when using the ON/OFF signaling from hardware. Hence, an application that is able to terminate properly using a normal operating system should be able to use the same solution on the XM2 device.
- In case of additional prevention, the applications can and should listen to operating system power management signals and/or power status signals via the CC AUX API. The shutdown process is a quick process and when shutdown signals occur the application shall terminate quickly, i.e. be able to quickly abrupt a file write in progress and do not write large files such as log files upon system shut down. A general design guide would be that an application shouldn't need more than a few hundred ms to make itself ready for shutdown
- The CC AUX PowerMgr API may be used by an application to delay OS shutdown and OS suspend operations in some use-case scenarios. The application can delay shutdown until it is ready with its operations.

3.2. Extend the CFast flash lifetime

- Application should not excessively write to the file system. Better approach is to use RAM-based log files and on regular intervals write files to permanent file system. Although use caution, RAM-files can be lost on sudden power off, so for mission critical data, another approach can be considered.
- Each write to permanent storage should be forced for synchronization, like (Linux perspective):

```
Command/script style: # sync
Programming style 1: res = fsycnc(fd);
Programming style 2: res = system("sync");
```

- It may be possible to add file system locks during startup as a startup script itself, to prevent unnecessary stress on the flash. This approach needs proper usage on several levels, making sure writes can be performed when they are supposed to be. File system locks can also be added as an extra security measure, possibly in combination with file system checks. But this may lead to longer startup times.

- For Linux based system the application should follow startup scripts guidelines to make sure that operating system signals are correctly passed to application, as found in the Software Guide document.
- The CC AUX Smart API can be used by applications to monitor the expected lifetime of the compact flash. It can be used to warn the user that a replacement is needed before the compact flash becomes corrupt.

4. Windows specifics programming guide

4.1. Set up the development environment.

Many development environments are available for Windows application development. This section describes setting up the environment for Microsoft Visual Studio, it is recommended to use version 2008 or higher. See the Microsoft documentation for Visual Studio installation procedure information.

4.1.1. SDK needed to start developing (on development computer)

The XM2 device specific SDK for development is divided into two parts in Windows, the *CC AUX API SDK* for hardware access functions and the *CC Win32 CAN API* for CAN functions. Both should be installed on the development computer for full access to all functions.

4.1.2. Drivers or packages needed to deploy on the XM2 device

No additional software needs to be installed on the XM2 device.

4.1.3. Setting up remote debugging

To enable efficient debugging, remote debugging can be set up where the application is executed on the XM2 device and debugged via the development computer. See the Microsoft Visual Studio documentation for set-up information.

4.2. Windows Interfaces specifics

This section covers windows specific details for programming with an XM2 device.

4.2.1. CAN

In Windows the CAN interfaces is accessed using the proprietary CAN API, see the CAN interface description documentation within the Win32CAN SDK, reference [4]. The CAN drivers are installed per default on the XM2 device but the SDK is needed in the development environment. Note that settings for CAN are stored in the Windows registry. Some, but not all of these are also available as settings in the CC AUX API.

5. Linux specific programming guide

This section is dedicated to useful tips and hints about Linux development hosts for application development and debugging purposes.

5.1. Retrieval of tool chain

The recommended tool chain is the x86/Linux cross compiler from Mentor Graphics called *Sourcery CodeBench,* which is a version of the standard gcc compiler. To get the *Sourcery CodeBench* cross compiler tool chain, visit the *Mentor Graphics Sourcery CodeBench* web site http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/overview. There are several alternatives for the cross-compiler, some with a price penalty but with better support, and one free version, called Lite. Register an account and download a suitable alternative for you. The recommended version at the time of writing this document is 2012.09-62 for **ia32 GNU/Linux**.

For support on tool chain issues, please refer to *Mentor Graphics CodeBench* support channels.

Use other compilers and/or toolchains at your own risk.

5.2. Installing tool chain

The installer binary requires the bash shell to be the default shell, and the downloadable file should have execution rights before the installation begins. If not enabled as default, make sure that */bin/sh* points to bash at least, or perform any other means of getting bash as the default shell. Make sure the installer binary has execution permission, and invoke it with root privileges:

```
# chmod +x ia32-2012.09-62-i686-pc-linux-gnu.bin
# sudo ia32-2012.09-62-i686-pc-linux-gnu.bin
```

The installer will start a Java-based installation GUI. The default settings should be applicable, but it is strongly recommended to install the cross compiler to */opt/codesourcery-x86/*, since it will integrate better with the target development files then.

5.3. Using tool chain

Once installed, the tool chain binaries should be found under */opt/codesourcery-x86/bin/*, all with the **i686-pc-linux-gnu-** prefix. The tool chain should be added to the path, i.e. add the directory mentioned to the *$PATH* environment variable.

```
user@host:~/$ i686-pc-linux-gnu-gcc --version
i686-pc-linux-gnu-gcc (Sourcery CodeBench Lite 2012.09-62)
4.7.2
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE.Now the cross compiler should be ready to
use.
```

5.4. Using correct development headers

The development package is released together with binary images. This package contains libraries available at the unit and all header files for utilizing them.

The development package is not installed to the device, it should be installed to the development host used for compiling software for the device. The development package can be extracted to almost any location at the development host, but it's preferred that it's extracted to */opt/XM2*, since there are default dependencies for Qt development binaries for that directory as the default directory.

When contents of this package change (not necessary at every release), the new version completely replaces the old release. Therefore, it is recommended that the package is extracted to a dedicated directory, which can be erased and recreated without losing anything else.

E.g.: extracting to */opt/XM2/*

```
# cd /opt
# tar xf ccpilot-xm2-devel-n.n.n.n.tgz
```

For the customer application to utilize the services on the device the cross-compiler has to find the correct development headers and libraries. Paths of those files must be added to the search path in project settings or the *Makefile*.

E.g.: if development package is extracted to */opt/XM2/*

```
# i686-pc-linux-gnu-gcc [other_commands] -I/opt/XM2/usr/include
-L/opt/XM2/lib -L/opt/XM2/usr/lib
```

5.5. Debugging remotely

To use gdb to debug an application running on the XM2 device, the application must have been compiled with the -g flag. Start **gdbserver** on the XM2 device:

```
~# gdbserver :10000 testApplication
```

Then start the host gdb and connect to the server:

```
# gdb testApplication
# (gdb) target remote Y.Y.Y.Y:10000
```

Above Y.Y.Y.Y is the IP address for the XM2 device. You can now debug the application normally, except that rather than to issue the run command one should use continue since the application is already running on the remote side.

Note that it is possible to fully debug the application but not the system calls made by the application. Such system calls include calls to the soft float library, like divide, add or multiply on floating point variables. It is therefore recommended to use next rather than step when such system calls are being made.

5.6. Linux Interfaces specifics

5.6.1. CAN

In Linux CAN is interfaced using SocketCAN. SocketCAN is a widely used CAN communication interface for Linux environments, and is a standard used in the Linux kernel.

Usage of SocketCAN requires knowledge of some system specific settings and details described herein, for additional SocketCAN information see the official SocketCAN documentation.

5.6.1.1. Configuration of the XM2 device interface

The XM2 device node files for the four CAN interfaces are *can0* to *can3*, which should be shown when listing all network interfaces with the **ifconfig** command. The XM2 device driver is implemented as loadable kernel modules, *can_dev.ko, xilinx.ko* and *xilinx__pci.ko*. In addition, there are at least one CAN protocol module providing access to the CAN protocol interface. A script handles the loading of the kernel modules upon start-up.

When XM2 device has finished its start-up, the CAN driver modules are loaded as a part of the kernel. This can be checked via terminal access using **lsmod** command:

```
# lsmod | egrep "can|xilinx"
can_raw         7552  0
can            23656  1 can_raw
xilinx_pci      3395  0
xilinx          6080  1 xilinx_platform
can_dev        15616  1 xilinx
```

Since the driver is compiled as modules, unnecessary protocols may be removed or new modules inserted according to user needs.

The CAN bus itself is not initialized during start-up, it only loads the drivers. Before any communications can be executed, user must set correct bus speed (as an example 250kbit/s) by first writing value into bitrate parameter:

```
# ccsettingsconsole --can=can0 --baudrate=250
```

and then setting interface up with **ifconfig**:

```
# sudo ifconfig can0 up
```

After this, **ifconfig** should show *can0* as a network interface:

```
# ifconfig
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:49
```

Same applies to the other CAN interfaces by changing *can0* to *can1, can2* or *can3*.

5.6.1.2. Bus recovery options

There are two options for implementing bus recovery after bus off has occurred: **manual** and **automatic**.

Manual recovery is initiated by writing a non-zero value to *can-restart* variable with ip - command:

```
sudo ip link set can0 type can restart-ms 10
```

Bus restart is then scheduled through kernel and implemented through can-core.

In automatic bus recovery, can-core detects state changes and re-initializes controller after the specified time period.

Automatic bus recovery from bus off state is by default turned off. It can be turned on via *sysfs* setting, where the wanted restart period in milliseconds is set into using the *can_restart_ms* variable. For example, a 100ms restart period for *can0* is set from command line like this:

```
sudo ifconfig can0 down
sudo ip link set can0 type can restart-ms 10
sudo ifconfig can0 up
```

Same commands apply for *can1* by replacing *can0* appropriately. Period is possible to set as needed from application perspective. Value zero turns automatic bus recovery off.

Warning: Enabling automatic bus recovery may disturb other nodes on bus, if CAN interface is incorrectly initialized.

5.6.1.3. Error interrupt options

Error interrupts are disabled by default. By enabling error interrupts user can receive error frames. Bus off errors will come through even if the error interrupts are not enabled.

Enable them by giving module parameter **errorirq=1** during module loading

```
# sudo modprobe xilinx errorirq=1
```

Or, by editing xilinx.conf under /etc/modprobe.d/ directory.

```
#xilinx.conf: Add new options to end of next line
options xilinx errorirq=1
```

Warning: Enabling error interrupts and sending frames when module is not connected to active bus may cause CAN acknowledge errors to overload CPU. User caution required. It is recommended to avoid sending until one frame is received.

5.6.1.4. SocketCAN Example

Below is an example of a SocketCAN application code. This example is not complete and may or may not compile as an application but it shows the nature of SocketCAN programming and the usage of standard socket programming.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>

#include <linux/can.h>
#include <linux/can/raw.h>
#include <string.h>

/* Define constants, if not defined in the headers */
#ifndef PF_CAN
#define PF_CAN 29
#endif

#ifndef AF_CAN
#define AF_CAN PF_CAN
#endif
```

```
/* ... */

/* Somewhere in your app */

   /* Create the socket */
   int skt = socket( PF_CAN, SOCK_RAW, CAN_RAW );
   const int loopback = 0;

   /* Locate the interface you wish to use */
   struct ifreq ifr;
   strcpy(ifr.ifr_name, "can0");
   ioctl(skt, SIOCGIFINDEX, &ifr); /* ifr.ifr_ifindex gets filled
                                    * with that XM2 device's index */

/* Select that CAN interface, and bind the socket to it. */
   struct sockaddr_can addr;
   addr.can_family = AF_CAN;
   addr.can_ifindex = ifr.ifr_ifindex;
   bind( skt, (struct sockaddr*)&addr, sizeof(addr) );

/* Disable filters and loopback feature. */
   setsockopt(skt, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0);
   setsockopt(skt, SOL_CAN_RAW, CAN_RAW_LOOPBACK,
              &loopback, sizeof(loopback));

/* Send a message to the CAN bus */
   struct can_frame frame;
   frame.can_id = 0x123;
   strcpy( &frame.data, "foo" );
   frame.can_dlc = strlen( &frame.data );
   int bytes_sent = write( skt, &frame, sizeof(frame) );

   /* Read a message back from the CAN bus */
   int bytes_read = read( skt, &frame, sizeof(frame) );
```

5.6.2. Serial Number Broadcast interface

The XM2 device has Serial Number Broadcast service. *SNB* does not have programming interface at the XM2 device end, but the broadcasted data output can be handled elsewhere, even in another XM2 device if required.

The message sent is a multicast UDP datagram to address 224.0.0.27. The message contains a char array with three values separated by tabs; Serial number, Firmware version and XM2 Device type. The sender's IP address is available in datagram headers.

Example data contents (without quotes):

```
"PR01<tab>0.3.0<tab>0"
```

An example implementation of the data listener is available in development package in *example_src/snb/snb_reader.c*

Technical support

Contact your reseller or supplier for help with possible problems with your XM2 device. In order to get the best help, you should have access to your XM2 device and be prepared with the following information before you contact support.

- The part number and serial number of the XM2 device, which you find on the brand label
- Date of purchase, which is found on the invoice
- The conditions and circumstances under which the problem arises
- LED indicator flash patterns.
- The XM2 Device log files (if possible)
- Prepare a system report on the XM2 device, from within *CCsettings* (if possible).
- Description of external equipment which is connected to the XM2 device.

Trademark, etc.

All trademarks sighted in this document are the property of their respective owners.

CCpilot is a trademark which is the property of maximatecc AB.

Intel is a registered trademark which is the property of Intel Corporation in the USA and/or other countries. Linux is a registered trademark of Linus Torvalds. Microsoft and Windows are registered trademarks which belong to Microsoft Corporation in the USA and/or other countries.

maximatecc AB is not responsible for editing errors, technical errors or for material which has been omitted in this document. maximatecc is not responsible for unintentional damage or for damage which occurs as a result of supplying, handling or using of this material including the devices and software referred to herein. The information in this handbook is supplied without any guarantees and can change without prior notification.

maximatecc respects the intellectual property of others, and we ask our users to do the same. Where software based on maximatecc software or products is distributed, the software may only be distributed in accordance with the terms and conditions provided by the reproduced licensors.

For end-user license agreements (EULAs), copyright notices, conditions, and disclaimers, regarding certain third-party components used in the XM2 device, refer to the copyright notices documentation.

maximatecc

maximatecc AB
P.O. Box 83 • SE-822 22 Alfta • Sweden
Phone: +46 271 75 76 00• info@maximatecc • www.maximatecc.com

CCpilot XM2 and CrossCore XM2

Software guide

maximatecc

www.maximatecc.com

Contents

Revision history

Rev	Date	Comments
0.1	2014-08-14	First draft
0.2	2014-08-19	Second draft.
0.3	2014-10-08	Third draft.

1. Introduction

This software guide contains information on how to start using the CCpilot XM2 and CrossCore XM2 products and how to access Service functionality.

XM2 devices are available with both Windows and Linux operating system. This guide is applicable for both operating systems. Operating system specific information is pointed out in the text or addressed in the respective operating system sections. Windows refers to Windows Embedded Standard 7.

The reader should be familiar with the respective operating system and computer usage to fully benefit from this material.

1.1. Conventions and defines

CCpilot XM2 and CrossCore XM2 are in most cases identical in functionality and usage. The following definition is used to separate unit specific details. The observe symbol is also used to highlight such difference.

Defines	Use
CCpilot XM2	Information that is specific for CCpilot XM2
CrossCore XM2	Information that is specific for CrossCore XM2
XM2 device	Information that applies to both CCpilot XM2 and CrossCore XM2

The observe symbol is used to highligt information in this document, such as differences between product CCpilot and CrossCore product models.

The exclamation symbol is used to highlight important information.

Text formats used in this document.

Format	Use
Italics	Paths, filenames, definitions.
Bold	Command names and important information

1.2. Identification

On the side of the XM2 device there is a label with version and serial numbers which identify your unique computer. Take note of them. During service and other contact with the supplier it is important to be able to provide these numbers.

1.3. References

For further information on the XM2 device and the available APIs see the following references.

[1] CCpilot XM2 and CrossCore XM2 – Programmers Guide

[2] SocketCAN in Linux kernel, http://lxr.free-electrons.com/source/Documentation/networking/can.txt?v=3.10

[3] CCAux API documentation (CC AUX 2.x.x.x, available in the CC AUX SDK)

[4] CAN Interface Description (available within the Windows SDK)

[5] CCpilot XM2 and CrossCore XM2 – Technical Manual

[6] Embedded Write filer documentation (MSDN), http://msdn.microsoft.com/en-us/library/bb521449%28v=winembedded.51%29.aspx

[7] File based write filter (FBWF) documentation, (MSDN), http://msdn.microsoft.com/en-us/library/ee832762.aspx

[8] Hibernate once resume many (HORM) documentation (MSDN), http://msdn.microsoft.com/en-us/library/bb521614%28v=WinEmbedded.51%29.aspx

[9] CCpilot XM2 and CrossCore XM2 – Performing Operating System Recovery

[10] CCpilot XM2 and CrossCore XM2 – Performing Firmware Updates

2. Basic operation

This section provides an overall description on basic usage of XM2 device.

2.1. Login and passwords

By default auto login is enabled and no passwords are required to start using XM2 device.

The root/administrators passwords are as follows

- Under Linux there are two users per default. The user is **ccs** and the password is **default**. This user has administrator rights by using the sudo approach. There is also the administrator user **root** with password **suseroot** as default.
- Under Windows there are two users per default. The Windows default administrator user **Administrator** with the password **CCpilotXM** and the user **Operator** which does not have any password. The Operator user has administrator rights and automatic login enabled by default.
- Custom users can be added with the tools from respective operating system.

2.1.1. Serial console login (Linux)

By default, the serial port can be used as a login terminal for headless access to the XM2 device. This service can however be disabled, if a peripheral connection on serial port is desired. This is specifically covered in chapter 7.4 Changing serial port settings.

Login service is using the same credentials as mentioned above for serial port terminal access.

2.2. Using the touch screen

Navigate the desktop or any application using the touch screen with a stylus or finger.

- Tap the screen to perform the equality of a mouse click.
- Double click is performed similar to using an external pointing device. Tap the screen twice in the same place.
- Tap and hold the on the touch screen to perform the equivalent to a right click.
- There are two available USB profiles for the touch screen; mouse profile and touch profile. The profile can be changed in *CCsettings* or through the CCAux API.

Besides the touch screen, keyboard and mouse connected via USB can also be used.

Touch screen functionality is only available in CCpilot XM2 product versions.

2.3. Keyboard

In Windows, there is a virtual keyboard available to input letters and symbols using the touch screen. The virtual keyboard can be used either via the mouse or the touch screen.

The icon to enable the virtual keyboard looks like this:

 in Windows

For Linux systems, no virtual keyboard is supported.

2.4. Calibrating the touch screen

To calibrate the touch screen, open *CCsettings*. Select *Touch* and then *Calibrate Now*. Then follow the step by step sequence to calibrate the display.

It is also possible to launch the calibration application explicitly from any custom application by calling the *TouchCalibrator* application (naming convention applicable per operating system). This application also take additional arguments to configure the calibration method, such as number of calibration points (5, 9 or 13 points) and edge distance settings of the calibration points. The 5 point calibration is the default method. The 13 point calibration can be used if the results of the 5 point calibration are not satisfactory. The available command line options can be seen with the `--h` or `--help` option flags.

Touch screen functionality is only available in CCpilot XM2 product versions.

2.5. Software deployment

There are several methods to add your own software to the device. The standard methods to transfer software to XM2 device are to either copy files using the network connection or to use USB storage devices. To install the software, follow the instructions for the respective software.

Additionally, software can be deployed with remote access functions, see the operating system specific parts in this document for more information.

2.6. Recovery access – log in via serial port (Linux only)

There are different ways to perform recovery of a device that has become corrupt in some way. One method is to log in via the external serial port interface. This login method uses the same credentials as other log in points, but the advantage by using the serial port login is that the log in via serial port is always enabled, unless specifically instructed to. The connection configuration is normally 115200-8-N-1 when using a serial port login.

See section 7.4 for additional details and how to disable the recovery login via external serial port method. Once logged in the user should have access to the device and should be able to perform necessary recovery actions.

3. Device start-up behaviour

At power on, the XM2 device has an internal microcontroller that monitors the power supply and performs start-up configurations and power settings. It then supplies the main computer board with power; from there the device execution begins, starting with the BIOS.

3.1. BIOS

The BIOS is XM2 device specific BIOS, which performs the initial setup of the main processor and its on-board peripherals. Once finished, the operating system execution takes over.

During BIOS operation different startup options can be accessed, the BIOS can for instance be used to occasionally boot from USB attached storage media by pressing the key **F11**. If other non-standard BIOS settings are desired, access the BIOS Settings menu by pressing the **DEL** key during BIOS startup. BIOS settings changes should normally not be needed and can be affecting specific functions within the system, so any changes must be careful considered before applying.

3.2. Windows system start-up specifics

Once the Windows operating system is started, normal operating system steps are performed. The Windows operating system is fully available as standard, with installed device drivers for all hardware peripherals available.

3.3. Linux system start-up specifics

The Linux operating system can be started in two different modes:

- Normal or Main system:
    - This is the main operation mode which includes all drivers to available hardware, system libraries, graphical applications and tools described in this and other documents.
- Rescue or Backup system
    - This is only for device updates and recovery use. It contains only basic maintenance tools. Updating the main system should be done from backup system. Most of the hardware is not available while on backup system.

Each of these modes is a completely separate a Linux operating system, each of which is a custom built x86 Yocto based Linux system consisting of a kernel and a root file system with system binaries and configuration files. Such a system is started with the Linux kernel execution, which turns over the execution process to the init system in the root file system. That in turn loads drivers and programs according to the configuration files, and eventually loads the user applications. The startup time of the system is normally defined as the time from power on until the user application can begin to execute.

Depending on the level of functionality needed by the application, it can be started at different startup levels. A very fast startup level means that some of the hardware might not yet have been initialized properly, and thus the application needs to handle that properly. On the other hand, a slower startup level guarantees that the required functionality is available upon application initialization.

Note that there is no need for rebooting into rescue mode to perform permanent settings.

3.4. Status LED indication

This section describes the basic default status LED behavior. The LED is located in different locations depending on the device type, see *Technical Manual* for details. Note that the status LED behavior can be disabled totally and/or overridden by user software through the CC AUX API

3.4.1. Start-up sequence

During startup, the LED indications are as follows:

1. Flashing YELLOW at 1 Hz. Pre-heating is activated and in effect, start-up is delayed while the device is heated.
2. Flashing YELLOW at 2 Hz. Device start-up phase begins.
3. Operating system is then started, and specific service software begins to execute.
4. Static GREEN. System is operational.

3.4.2. Shutdown sequence

Once shutdown or reboot is initiated:

1. Static YELLOW. Shutting down, rebooting or entering suspended mode.
2. LED OFF. Device is off

3.4.3. Suspend/resume sequence

Once suspend is initiated:

1. Flashing YELLOW at 0.2 Hz. In suspend mode
2. Flashing YELLOW at 2 Hz. Resuming operation
3. Static GREEN. System is operational.

3.4.4. Error indication

There are two different types of error indications available from the status LED:

- Flashing RED – An occasional three RED blinks during startup indicates an internal error that is recoverable, i.e. the system can continue its execution and normal operation. Error code can be retrieved trough the CCAux API, since the status LED will follow its normal status indication scheme.
- Flashing BLUE – The device becomes non-operational, and seems to be dead, despite powered on. A number of BLUE LED blinks should be displayed. Record the number of blinks, as this can be useful in an eventual support contact.

Reason for BLUE LED blinking could be voltage levels out of range, temperature related problems or internal hardware errors. First steps should be to let device cool off, and verify it has correct power supply attached, before starting the device again.

3.5. Default startup applications (Linux)

On the CCpilot XM2 Linux versions, a default graphical application is started when the device is started. This application supports the user with a GUI for simple tasks and information until the

startup application can be replaced with the proper user defined application. The replacement of this application is described in section 7.5.

It's also possible to connect to the device via a VNC Viewer application by default; it will however require a known IP address of the device. Further information about that is found in section 7.6.5.

4. Accessing and using the interfaces of XM2 device

This section covers basic usage and access of the XM2 device hardware. Most of the hardware is accessed using the default Windows or Linux interfaces but some XM2 device specific interfaces, such as CAN and Digital In, require additional software and/or interfaces to be accessed. See the *CCpilot XM2 and CrossCore XM2 – Programmers guide* documentation for general information regarding software development using XM2 device interfaces.

Depending on product model, all interfaces may not be present on your specific model. There may also be additional methods to access XM2 device than the ones described herein, depending on operating system and additional installed software.

4.1. CFast/HDD

XM2 device uses a CFast card (flash) based storage. The storage is identified to the system as an IDE-device, i.e. a normal hard disk drive.

The CFast module is industrial grade classified and has both static and dynamic wear levelling to prevent a premature aging and to ensure the longest lifetime, still CFast drives have a limited number of write cycles. It is recommended that the amounts of writing to storage are limited within the application. Rather keep information in RAM memory and write larger blocks at one time instead of frequently writing smaller pieces.

Additional details about CFast usage recommendations can be found in *CCpilot XM2 and CrossCore XM2 – Programmers guide*.

4.1.1. Windows

In the maximatecc Windows images, measures have been taken to reduce the number of accesses during normal operation. These measures include disabling of file access disabling of prefetching services. Note that the page file is disabled by default; this is to life time of the CFast card and reduce the system reserved storage media space. Further configuration to reduce flash write cycles can be set up by the customer: See chapter 0

Windows system specifics, for information about the EWF, FBWF and HORM technologies.

4.1.2. Linux

In Linux, the CFast card is partitioned into three different partitions, where two of them hold the write protected main system and rescue system respectively, and the last partition holds the user file system area, which is write enabled by default. The latter area is the preferred location for user software installations. See chapter 7.1 Installing new drivers, applications and system packages

4.2. CAN

XM2 device has four CAN interfaces with user configurable baud rate and frame type accessible from the CCsettings application or from registry and/or standard file system settings.

The CAN interfaces can also be accessed in Windows using the proprietary CAN API, further described in the *CAN Interface Description*, or with the Linux operating system standard API *SocketCAN*. More information can be found in *CCpilot XM2 and CrossCore XM2 – Programmers Guide*.

4.3. Ethernet

Two Ethernet ports are available on XM2 device, thus enabling simultaneously connection with two physical network infrastructures. XM2 device is per default set up to use DHCP for IP address retrieval. The network connection settings can be changed within the operating system settings.

Be aware that connecting XM2 device to a network environment can impose a security threat if not taking the required security measures.

4.4. USB

Via the USB ports, a multitude of devices can be connected to XM2 device. For some devices, drivers compatible with the operating system must be installed in order for the device to function. Follow the instruction from the device supplier for the respective operating system.

4.5. RS232 Serial access

Select *COM1* in Windows to access the serial port of XM2 device.

In Linux the serial port is accessed from *dev/ttyS0* and when communicating with the targets, the settings for serial port to use are *115200-8-N-1*. By default the serial port is set up as a console, but this can be disabled to allow connection of peripheral accessories. covered in chapter 7.4 Changing serial port settings.

login
This is specifically

The Serial port in XM2 device follows RS232 signalling levels, but with a limited set of signals. For more information see the Technical manual.

Note that the serial port should not be used by any software when performing upgrades of internal microcontroller firmware. The internals of the device needs to have undisturbed access to the serial port during this operation.

4.6. Video in

The video in signals can be viewed by the *CCvideo* application, further described in chapter CCvideo. The video in signals can also be accessed and controlled using the CCAux API.

4.7. Audio In / Out

Audio in and audio out is controlled using the normal operating system functionality. The audio device is identified as *Cirrus Logic HD Audio*. Note that the audio input is muted per default.

4.7.1. Linux audio settings

Audio in and out is controlled using the normal operating system functionality from the ALSA libraries, including sample applications.

4.7.2. Windows audio settings

The sound settings can be managed using normal operating system procedures accessible via the control panel.

4.8. Digital Input

The digital input signals are available for software developers using the CCAux API. The digital input status can be viewed within CCsettings for test purposes.

4.9. Status LED

The status LED in the XM2 device is used by the computer itself and by the operating system to indicate different states or to present warnings, as described in 3.4 Status LED indication. The status LED can also be controlled by the applications running on XM2 device using the CCAux API. The blinking frequency and colour of the LED can be controlled. The latest status LED instruction is always used. The status LED can also be disabled which means that only hardware error codes and user application behaviour is enabled. Yellow blinking at start-up and static green in operational mode are turned off.

4.10. Backlight

The XM2 device has an adjustable backlight intensity level. In addition to the backlight control buttons on the CCpilot XM2 front, the backlight functionality can also be controlled from *CCsettings* and via software using the CCAux API. The most recent setting is always used. The backlight buttons are dedicated for backlight adjustment only.

Together with the ambient light sensor it is possible to make a custom, fully automatic, backlight control. Such an automatic backlight control function is included in CCpilot XM2, but it is not enabled by default. It can be set up in *CCsettings* or through the CCAux API.

For CrossCore XM2, the backlight and ambient light sensor is not available.

4.11. Ambient light sensor

The ambient light sensor measures light levels in front of CCpilot XM2; for example it's used for automatic backlight control. The ambient light sensor is accessed through the CCAux API. It can also be accessed for diagnostic through *CCsettings*.

4.12. Buzzer

In addition to the audio In/Out signals, CCpilot XM2 is also equipped with a buzzer that can play tones in various frequency and intensity levels. The buzzer is accessed through the CCAux API. It can also be accessed for diagnostic through *CCsettings*.

For CrossCore XM2, the buzzer is not available.

4.13. EEPROM

A user accessible EEPROM is available in the device, if such storage is required. It's not normally needed since a device of this type can store persistent settings in ordinary files in the file system instead. This functionality is available using the CCAux API.

4.14. Hardware watchdog functionality

There is a watchdog available making it possible for the hardware to monitor an application so it does not stall the system. The exact behaviour is up to the application software to handle and configure. This functionality is available using the CGOS interface.

4.15. Temperature sensors

There are several temperature sensors placed internally in the device. Thus it is possible for an application to retrieve temperature information from the temperatures sensors through the CCAux API.

5. CCsettings

CCsettings is used for viewing and adjusting XM2 device specific features and information. Through the *CCsettings* main window, the different settings pages are reached, each containing different areas of settings and information.

All functionality and settings accessed using *CCsettings* is also available through the CCAux API to enable integration of settings and functionality within any additional user application, or from the command line with the tool *ccsettingsconsole*. For details about this, please see the API documentation or *ccsettingsconsole --help*.

Start *CCsettings* from its menu item or optional desktop shortcut. In Windows, *CCsettings* can also be started from the Control Panel.

5.1. Main menu

The main screen of *CCsettings* show the available settings pages.

Click on the respective icons to enter its settings page.

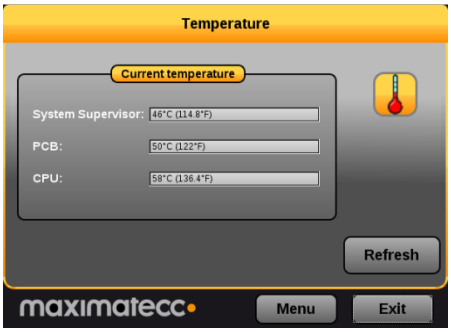
It is always possible to revert to the main menu by pressing the *Menu* button. If a keyboard is in use, the Escape key can also be used to revert to the main menu.

5.2. Version

The version page displays the XM2 device internal software versions, as well as application interface version and the version of *CCsettings* itself.

5.3. Temperature

XM2 device's internal temperature sensors can be viewed from the temperature page.



5.4. Display

Use the display page to alter the XM2 device backlight functionality.

Adjust the backlight of the display by dragging the *Backlight level* slider.

Enable automatic backlight using the *Enable* checkbox. The display brightness is then adjusted using CCpilot XM2's light sensor. *Soft transitions* enable smoother adjustment of the backlight.

The *Sensitivity* slider adjusts the level and phase of backlight adjustment depending on the surrounding light.

The Status LED can be automatically dimmed according to the backlight setting if LED dimming is enabled. Note that this may not work if a custom color is used or if a blink sequence is set up by user applications.

For CrossCore XM2, the Display option is not available in CCSettings.

5.5. Power

The power page is used to enable or disable the built in power controls in XM2 device.

Enable power by checking the respective check box.

Using the *Power status* drop down box it is possible to adjust the current setup or the default setup during system start-up.

5.6. Heater

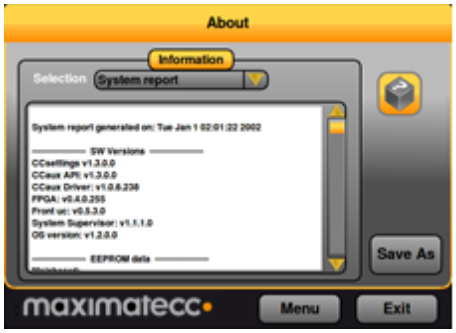
The heater page is used to adjust the starting temperature of the internal heater.

Adjust the temperature and activate the setting by pressing the *Apply* button

5.7. About

The about page displays device specific information and status information. Select what type of information to display using the *Selection* drop down box.

Select *System report* drop down to generate a full device information report, which can be saved to file. This report can be very useful for service and support questions.



5.8. LED

The LED page is used to view the possible LED indication types and to adjust the LED behavior during startup.

Enable or disable the status LED indication during startup of XM2 device using the *Enable* checkbox.

5.9. Buzzer

The Buzzer page is used to analyze the possible audio output from the Buzzer.

Adjust the frequency and volume of the built in buzzer using the sliders.

Test the buzzer sound by pressing the *On* button. Press it again to turn off the buzzer.

For CrossCore XM2, the Buzzer is not available in CCSettings.

5.10. CAN

CAN is normally used and configured directly from the controlling software. But from *CCsettings* it is possible to adjust the default *Baud rate* and CAN *Frame type*.

The *Device* drop down box is used to select the CAN device to adjust.

5.11. Digital Input

The digital input page displays the status of the digital input ports on XM2 device.

5.12. Touch screen

The touch screen page is used to select whether the touch display shall be identified to the operating system as a touch screen or as a mouse.

Select *Use touch profile* to identify the touch screen as a touch screen, this is the default and preferred setting.

Select *Use mouse profile* to identify the touch screen as a mouse, this setting is required for Windows XP-based installations without the tablet PC component.

If the touch screen if the pointing precision seems low it is possible to make a calibration. Start the calibration sequence by selecting *Calibrate Now* and follow the step by step sequence. Different calibration methods can be chosen to perform more precise calibration.

In the *Advanced* tab, additional touch and calibration settings can be made. For instance, by entering a time at *Right click time* and pressing *Apply* it is possible to adjust the time needed until pressing on the touch screen will result in a right click action. See the CCAux API documentation for further information on the advanced settings.

For CrossCore XM2, the Touch Screen option is not available in CCSettings.

5.13. Light sensor

The light sensor page is used for Light sensor evaluation. This access is for evaluation, no settings are stored when leaving the light sensor page.

For CrossCore XM2, the Light Sensor option is not available in CCSettings.

5.14. On/Off

The On/Off page is used to adjust the startup and shut down settings of XM2 device.

Any altered setting must be activated using the *Apply* button.

Short press - ON/OFF button - adjust the unit behavior when making a quick press and release of the on/off button.

Long press - ON/OFF button - adjust the unit behavior when pressing and holding the On/Off button.

ON/OFF signal - adjust the behavior of the On/off signal through the power connector. The time settings adjust how long the signal must stay low before XM2 device reacts to the signal.

Unit start-up makes it possible to select if the unit shall respond to start-up from both the on/off button and/or the on/off signal in the power connector, or if it shall be configured to start up directly when supply voltage is applied.

Suspend mode is used for activating automatic transition from suspend mode to shut down of the computer. The time defines the amount of minutes XM2 device shall remain in suspend mode before shutting down. Setting the time value to *0* disables the shut-down feature; XM2 device will stay in suspended mode indefinitely, until started or until power is removed.

5.15. Advanced

The Advanced page is used for loading new firmware into XM2 device and to restore settings to factory defaults.

To update the respective firmware, make sure the firmware file is located on the device. Browse for the appropriate file, then press the corresponding upgrade button and wait for the firmware update to complete.

Note that no other software can use the serial port of the device while upgrading the *SS* or *Front* controller. This includes the serial console usage under Linux, and thus an upgrade must disable this prior to starting the upgrade.

For detailed information about how to upgrade the firmware, please see [10] CCpilot XM2 and CrossCore XM2 – Performing Firmware Updates.

6. CCvideo

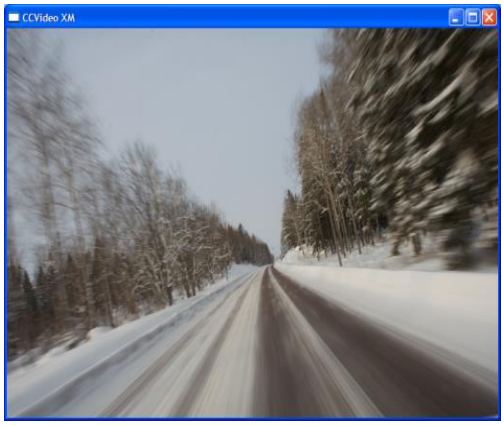
CCvideo is a pre-loaded application for viewing the CCpilot XM2 video in signals. Two video signals can be simultaneously displayed, using any of the four input channels.

All functionality and settings accessed using *CCvideo* is also available through the CCAux API to enable integration of settings and functionality within any additional user application.

Video functionality viewing is not applicable for the CrossCore XM2.

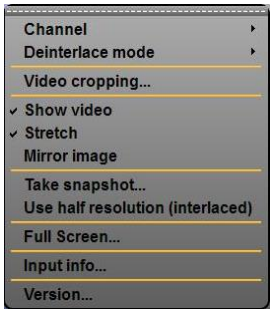
6.1. CCvideo main window

When starting *CCvideo* the main window will be shown. This window displays the currently selected video signal.



6.2. Selection menu

The selection menu is activated by right clicking within the *CCvideo* window.



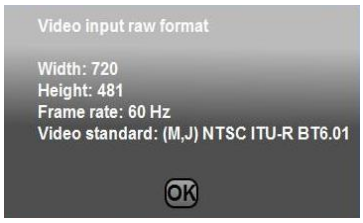
- ---- - Lock the menu visible.
- *Channel* - Select the video in signal to display.
- *Deinterlace mode* - Select deinterlace mode for video generation.
- *Video cropping* - Opens the Video cropping dialog.
- *Show video* - Enables/disables the video in feed to be displayed.
- *Stretch* - Enables/disables stretching of the video image to fit the size of the *CCvideo* application window.
- *Mirror image* - Enables/disables video image mirroring.
- *Take snapshot* – Take a snapshot of the video image. The image will be saved to disk and the user will be asked if he wants to view the image in the OS default image viewer.This is currently not supported in Linux.
- *Use half resolution (interlaced)* – Setting for the snapshot function. If enabled, the snapshot image will only use half of the original video resolution. This function is intended to be used by custom applications if disk/memory usage and performance of capturing images are critical. This is currently not supported for Linux.
- *Full Screen* – View video in full screen mode.
- *Input info* - Displays input information for the currently viewed video in feed.

- *Version* – Display the software version of *CCvideo*.

6.3. Input info

The input info dialog displays information for the currently used video in feed.

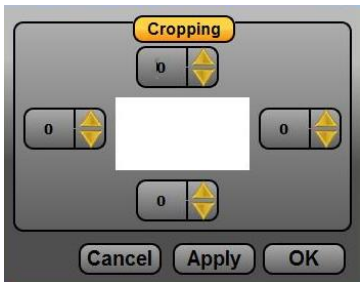
- *With* - Native with of the signal in pixels.
- *Height* - Native height of the signal in pixels.
- *Frame rate* - Current frame rate of the video signal.
- *Video standard* – The detected video standard of the video signal.



6.4. Video cropping

Use the video format settings to adjust the video being shown.

- *Cropping* - Removes black or unwanted parts of the video signal. The video feed is cropped by the number of lines set for each part of the picture.



7. Linux specifics

This section describes specific details for systems running Linux. The Linux system is a custom built version, using the Yocto project and BSP files for the current CPU module, as well as customized recipes to form a slimmed system based on sample applications in Qt and the X Window system graphics framework.

This section describes specific details for the configurability of the software components in the system, such as default configuration files, startup scripts and networking settings.

7.1. Installing new drivers, applications and system packages

All additional software or user files should normally be installed and stored under the user partition */opt/ (or /usr/local)*, which is mounted read-write. Under */opt*, a limited set of standard directories are found which are always writable, those directories are */opt/etc*, */opt/bin*, */opt/lib*, and */opt/sbin*.

7.1.1. Remounting file system in read-write mode

In very rare cases, editing write-protected files may be required. It is possible to temporarily mount the file system writeable, to allow edit of protected files, using the following commands.

```
# sudo mount -o remount,rw /
```

Important, remember to use the following command to remount the file system as write protected again, before shutting down or restarting the device. Note that any changes to the write-protected file system will be overwritten when performing an operating system upgrade.

```
# sudo mount -o remount,ro /
```

7.1.2. User libraries

To install additional shared libraries, install the library files into */opt/lib/*. Then, update the used library cache file by executing the following command:

```
~# sudo ldconfig -C /opt/etc/ld.so.cache
```

If additional library file locations are needed, the paths of these can be added to above command as parameters. The environment variable *$LD_LIBRARY_PATH* can also be used for finding library files not in the cache.

Library cache file is never automatically updated. But if the file does not exist at system start-up, it is re-created with default version containing information only about the standard libraries.

7.1.3. User binaries

Additional binaries such as customer application software or additional open-source solutions are preferably installed to */opt/bin/* or */opt/sbin/*. These directories are available in the standard path, adding binaries to these locations does not require an update to the *$PATH* environment variable.

If additional levels of binaries are required, the *$PATH* environment variable must be updated through a start-up script.

7.1.4. Start-up scripts

The user has the possibility to start applications and scripts by modifying or adding start-up scripts. When the kernel is started, the start-up script *rc* located in */etc/init.d/* is executed. Normally, this script reads start-up scripts under */etc/rcX.d/,* depending on the actual run level *X*.

The default run level is 3, so applications should at least have startup scripts for this run level. The *rc* script has been modified to parse additionally start-up scripts found in */opt/etc/rcX.d/* as well as standard system scripts. The parsing is done in a temporary directory, so scripts from each source location are interleaved depending on their respective names as described below.

To start applications in run level 3, create a script located in */opt/etc/rc3.d/* that starts the desired applications. This is the default run level. Each script must be named *SXXname*, where XX is two digits and corresponds to the order of the script execution.

Note that these scripts are usually sourced, so no exit should be performed within these scripts, nor should any application lock the scripts by not performing proper spawning or forking.

The scripts should follow the correct format for the start-stop system to work correctly. For more information on how the scripts should be created, see standard reference documentation for rc scripting.

Run level 0 is dedicated for shutdown. When the device is shutting down the scripts from */etc/rc0.d/* and */opt/etc/rc0.d/* are executed to perform last clean-up actions. Required naming and execution order rules comply with start-up level 3, but kill scripts are normally named *KXXname* instead.

Additionally, there's a specific shutdown script available for user editing. It is called /opt/shutdown, and is by default an empty script. Any user can add specific shutdown related tasks in that script, if desired.

7.2. Text editor

The console text editor *nano* is available per default for text editing, as well as the *vi* editor.

7.3. IP address configuration

There are several ways of setting the IP address of a device. The default method is DHCP, but a static IP address can also be used. This can be done through the network interfaces configuration file.

7.3.1. File method for IP address configuration

The network interfaces file is located in writable storage, but the edit process has been made transparent to that fact. This method requires knowledge about the interfaces file format, but a sample is given below.

```
# sudo nano /etc/network/interfaces
```

Sample of interfaces file setting a static address.

```
auto lo
iface lo inet loopback

iface eth0 inet static
address 192.168.2.185
```

```
netmask 255.255.255.0
gateway 192.168.2.254
```

Once the file has been edited, it is recommended to either reboot the device, or to bring the network interfaces down and up again, for the IP address configuration to take effect.

```
# sudo ifdown eth0
# sudo ifup eth0
```

7.4. Changing serial port settings

By default the serial port is set up as a console, but this can be disabled to allow connection of peripheral accessories, or to allow firmware components upgrade. The serial port is accessed from *dev/ttyS0* and when used as a serial console, the settings for serial port are *115200-8-N-1*.
To disable the serial console do the following:

```
# sudo disable-serial-console.sh
```

Use the following command to enable it again

```
# sudo enable-serial-console.sh
```

When the serial port is not used as a serial console, i.e. disabled, the settings can be changed to fit your connection requirements with standard serial port operations.

Earlier disable/enable – commands do not work over reboot. To permanently disable serial-console:

```
# sudo touch /opt/etc/disable-serial-console
```

To permanently re-enable serial-console:

```
# sudo rm -f /opt/etc/disable-serial-console
```

7.5. Default startup application

As mentioned in 3.5 Default startup applications (Linux), CCpilot versions have a GUI application that is started by default. The default application looks like:

A user can and should replace or simply edit the default startup application script, i.e. the init script found at */opt/etc/init.d/StartupGui*. This init script is automatically run when entering run level 3 (during boot) or run level 0 (during shutdown).

Note: This script is part of the rc script solution mention in section 7.1.4. Make sure the correct script behavior is met; otherwise it can break the XM2 device startup procedure.

An example of the default startup script is given here for illustration purposes:

```
#
# StartupLauncher init-script
#

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/b
in

# Count the number of Xorgs running.
XORGS=`ps | grep Xorg | wc -l`
RUNNING=2 # If less than 2 Xorg processes then Xorg is not
running

case "$1" in
    start)
    # Wait until XORG is running
    while [ "$XORGS" -lt "$RUNNING" ] ; do
        sleep 1 # Sleep to prevent busylooping
        XORGS=`eval ps | grep Xorg | wc -l`
    done
```

```
    DISPLAY=":0.0" StartupLauncherGui &
    ;;


    stop)
    killall StartupLauncherGui
    ;;

    restart)
    killall StartupLauncherGui
    DISPLAY=":0.0" StartupLauncherGui &
    ;;

    force-reload)
    B=3
    ;;

    force-stop)
    B=4
    ;;

    *)
    echo "Usage: S50startuplauncher {start|stop|restart}" >&2
    exit 1
    ;;
esac

exit 0
```

If the user wants no startup application to be run during boot, an empty file must be stored in place of the script. If removed completely, the missing script will be replaced by the default on next boot. Note that it is still possible to start applications in parallel or with other run level scripts, as mentioned in 7.1.4 Start-up scripts.

7.6. Remote access

The methods described in this chapter require an IP address being assigned to the XM2 device.

7.6.1. SSH

To connect to the XM2 device, issue the following command (and give password when asked):

```
# ssh ccs@X.X.X.X
```

To connect to a host from the XM2 device, issue the following command:

```
~# ssh Username@X.X.X.X
```

Above X.X.X.X is known as an SSH server IP address, with username *Username*. A password might be necessary.

7.6.2. SCP

To copy a file to target use the following command (and give password when asked):

```
# scp File1 ccs@X.X.X.X:/opt/File
```

To copy a file from target use the following command (and give password when asked):

```
# scp ccs@X.X.X.X:/opt/File File
```

To copy a file from a host while on a XM2 device to a host, use the following command:

```
~# scp File Username@X.X.X.X:File
```

To copy a file from the XM2 device to a host, use the following command:

```
~# scp Username@X.X.X.X:File File
```

Above X.X.X.X is known as an SSH server IP address, username *Username*. A password might be necessary.

7.6.3. Password-free login for SSH and SCP

Even though the *ccs* user has password, SSH-connections can be configured to connect without password, using identity files. This method is mainly useful for remotely executed scripts or alike.

On connecting host (not the target XM2 device), execute the command below and enter an empty passphrase when prompted.

```
~$ ssh-keygen -t rsa -f xm1_rsa
```

Copy (or append) the created *xm1_rsa.pub* file into target XM2 device as a */etc/ssh/authorized_keys* -file. Note, this method needs to be done in recovery mode.

Move the *xm1_rsa* file to a usable location (e.g. *~/.ssh/*)

Either configure the id-file into use in *ssh_config* or assign it when executing **ssh** or **scp**.

```
~$ ssh -i ~/.ssh/xm1_rsa ccs@xm-linux
```

7.6.4. Remote command execution

After password-free login is enabled, any commands can be started remotely without login.

```
~$ ssh ccs@xm-linux "ls -al /opt/"
```

If starting services or background tasks, append "&" to command between quotes.

7.6.5. VNC

The XM2 device Linux version also has a VNC server enabled, so that the GUI desktop can be accessed remotely, for instance when using CrossCore XM2 headless devices.

To access the device via VNC, you will need to use a VCN viewer application and enter the IP address of the XM2 device. By default, no password is needed.

7.7. Serial Number Broadcast configuration

The XM2 device Linux version can identify itself over the IP network by sending out its serial number as a broadcast IP packet. The Serial Number Broadcast (SNB) service is started by default at device boot-up and will broadcast a specific identification message to the local network every fifth second. The message send frequency can be modified and the service can be completely

disabled using the configuration file */opt/etc/ccsnb.conf*. This file does not exist by default. Default values are used if any value is unset or the file does not exist.

```
# Serial Number Broadcast - configuration.
# Lines beginning with '#' are comments.  Unnecessary options
can be omitted.

# Message send interval in seconds
INTERVAL=10

# Service disable switch (DISABLE|OFF|0)
#ACTIVE=DISABLE

# Advanced features only. Use with discretion.
#
# Firmware-field is auto discovered, but can be overwritten.
String value
#FIRMWARE=1.0.0
#
# UnitType-field is TBD, if unset '0' is used. String value
#UNITTYPE=XM
```

7.8. USB memory installer

If a USB memory is inserted before start up, it is possible to activate a run time hook to enable automatic software execution. For instance, this function is suitable for production time installers, or automated SW updates.

If you add a script (executable file) that's called **cc-auto.sh** (only that) in the root of a USB memory, it will be executed during device startup. The USB memory can be a FAT-formatted one or with other suitable file system, so there is no specific requirement there, but remember that some Windows based editors will leave Windows EOL characters in edited files, including scripts. Such characters may or may not affect the execution of this type of script.

The *cc-auto.sh* script is then executed automatically at insertion of USB memory or during start up. By placing commands which copy new applications and perform updates and installations in that script, this feature can be used for creating auto-update of user and system software. There are no specific limits on what user can do with *cc-auto.sh file*, but it's recommended that all desired commands are applied within the *cc-auto.sh* script process.

7.8.1. Example of automatic software installation with USB memory

This is an example that enables operating system update to be automated, but it can be used as an illustration on what can be done with the installer script as such. This script solution is to be used for the backup system solution only, that's not needed in the normal user software update case.

1. *cc-auto.sh* creates folder */usr/local/my_application/*.
2. *cc-auto.sh* copies all necessary files to that folder or to other folders from USB memory.
3. *cc-auto.sh* performs additional application setup actions.
4. *cc-auto.sh* may run *reboot* but it is not necessary. If reboot is written, remember to remove memory when restarting, otherwise the device will enter a reboot loop since the script is also run at startup.

At this point the USB memory can and should be removed, and device should reboot itself and start the installed applications. A small script example is given below:

```
#/bin/sh
# Automatic installation script example

mkdir -p /opt/my_application
tar -C /opt/my_application -xzf
/media/usb/my_application.tar.gz

ldconfig -C /opt/etc/ld.so.cache

sleep 10

reboot
```

7.9. Media files playback

Video and audio playback is supported by gstreamer multimedia framework, which supports multiple video (MPEG2, H.264/AVC, DivX, Xvid) and audio codecs (wav, MP3, AAC) and container formats (MPEG, AVI, mkv). Note that the CCpilot XM2 currently supports only codecs for WAV audio playback. Video player is not included in the system. Audio file can be played with the command:

```
# aplay <audiofile>
```

This command launches a player and plays the file automatically so it can also be launched from a script. For custom applications, gstreamer-0.10 C API can be used (see documentation at http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html).

There is also a Qt frontend system available for gstreamer as well, which is called Phonon.

7.10. USB memory folders

If a USB memory is inserted, it is normally mounted automatically and mapped to /media/usbsdb1. Some USB memories are however not mounted there due to specific USB memory brand issues, they are then mounted in a more ad-hoc way. Such mounts can be detected using Linux standard tools and script solutions, if it needs to be detected automatically within user software.

7.11. Removal of user data and files, i.e. factory reset

The XM2 device normally stores all user data, applications and settings under /usr/local, or /opt which points to the same storage place. It is possible to remove all of the settings and files under that location, and have the XM2 device generate the default contents back upon a restart of the device.

Note: This can potentially also remove some applications and files that are part of the factory installation by maximatecc and delivered to you as such. If that is the case, please avoid this method of user data and file removal unless specific knowledge about this has been gathered.

The restoration is done via the command

```
# reboot-rescue.sh clear
```

which will reboot the device twice and reformat the entire user partition, as well as restoring the default files that actually are there.

8. Windows system specifics

This section describes specific details for system running Windows. The XM2 Windows images are adapted with additional drivers and software to access XM2 device specific hardware and functionality.

8.1. Windows system content

To preserve file system size the Windows images have a limited set of components. For example, only the English language support is pre-installed. Customer specific images may be created if specific software or component configuration is needed.

8.1.1. Windows drivers

Dated 2014-10-06, the Windows driver content is as follows, i.e. this specifies the 3rd party software that is used in XM2 device:

Usage	Provider	Driver
Audio	Cirrus Logic	CS4207_WinVista_Win7_32-64-bit_6-6001-1-30
Chipset	Intel	Intel Chipset Device Software 10.0.13 (10.0.13.0)
Ethernet	Intel	Intel PROSet 18.6
Graphic	Intel	Intel EMGD 36.15.0.1073
USB	Intel	Intel USB 3.0 eXtensible Host Controller driver 3.0.0.19
CGOS	Congatec	Congatec API
Virtual COM	Future Technology Devices International	CDM20824_WHQL_Certified
CCaux	maximatecc	CCaux API

8.2. File system layout

The default file system is NTFS and the compact flash is set up with one partition *C:*.

8.3. Installing new drivers, applications and system packages

When installing third party software, use the installation guidelines provided by the software provider.

8.4. Remote access

There are a couple of remote access software programs installed in the XM2 device with Windows. Those are SSH server, VNC server and serial number broadcaster service.

8.4.1. SSH

It is possible to connect to the XM2 device with SSH access even when its operating system is Windows. This provides a means to remote copy files to the XM2 device over the network in analogy to how it's done with the Linux system. It also provides terminal access to the Windows system from a remote location.

8.4.2. VNC

The XM2 device also has a VNC server enabled, so that the GUI desktop can be accessed remotely, for instance when using CrossCore XM2 headless devices. The VNC server requires a password, and it's set to "default".

To access the device via VNC, you will need to use a VCN viewer application and enter the IP address of the XM2 device, along with the password when prompted for.

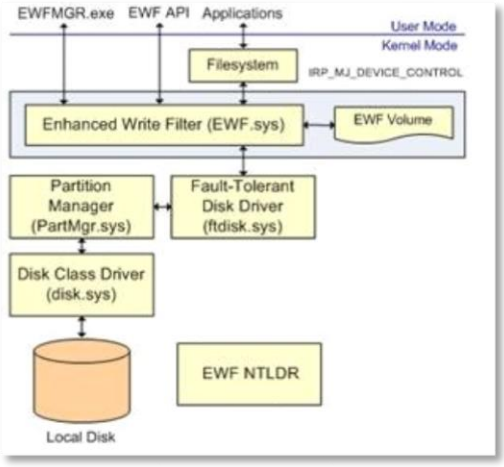
8.4.3. Serial Number Broadcaster

The XM2 device Windows version can identify itself over the IP network by sending out its serial number as a broadcast IP packet. The **CCSnbService** service is started by default at device boot-up and will broadcast a specific identification message to the local network every fifth second. This service is part of the CC AUX installation package, and can be disables as any normal operating system service.

8.5. Embedded write filter

The maximatecc Windows Embedded 7 (WES7) image uses the Embedded Write Filter (EWF) to reduce wear on the compact flash disk.

EWF is primarily intended to write protect a run-time image. For Compact flash based devices it can also be used to extend the lifespan of the media by limiting the number of disk writes. In WES7, EWF can work only in RAM mode. This means that EWF works as a RAM overlay enabling the OS to act as normal, but no changes are committed to disk until the user triggers a commit. EWF commits includes all changes to the partition.



Follow these steps to enable EWF RAM reg mode on a partition:

```
1. Enable EWF on partition:
> ewfmgr <partition> -enable
      e.g. > ewfmgr C: -enable
2. Reboot for EWF settings to take effect.
```

Please refer to the Microsoft EWF documentation, [6], for more information on how to set up and use EWF.

8.6. File based write filter

In addition to the Embedded Write Filter, maximatecc Windows Embedded 7 (WES7) image also includes File Based Write Filter (FBWF).

One disadvantage with EWF is that it only works on a sector level and is unaware of the file structure which makes it hard to implement single file commits. Therefore a file based write filter is available which operates on a file level. Using FBWF you can set up a list of write through files which are persistent and keep the rest in the RAM overlay reducing the number of writes to disk.

Using FBWF it is also possible to perform file level commits. The normal usage scenario for EWF and FBWF is that EWF is used to protect the partition with the run-time image and FBWF is used on a data partition to enable file operations.

Follow these steps to enable FBWF on a WES7 image:

```
1. Enable FBWF
>fbwfmgr.exe /enable
2. Add a volume to protect
>fbwmgr.exe /addvolume C:
3. Add a directory to exclude (add to write-through list)
>fbwmgr.exe /addexclusion C:\Personal
4. Check the status
>fbwmgr.exe
5. Reboot for changes to occur.
```

Please refer to the Microsoft FBWF documentation, [7], on how to set up and use FBWF.

8.7. Hibernate once resume many

Hibernate once resume many (HORM) is a feature available in the maximatecc Windows Embedded 7 (WES7) image.

HORM enables a robust boot scenario using a one-time created hibernation file from which the system always boots. This enables the creation of very fast booting as well as robust devices. Even after power failures these systems come up quick and without damages to the file system. HORM requires RAM or RAM (Reg) overlay and can only be configured using EWF. HORM can be activated and deactivated using a command line tool. HORM has a requirement that all volumes must either be protected with EWF or be in unmounted state when the Hibernate Once occurs. This is to prevent state synchronization problems. Each Resume from hibernation expects the entire system to be in exactly the same state as when the Hibernate Once occurred.

Follow these steps to enable HORM on a WES7 image:

```
1. Enable hibernation
> powercfg.exe /h ON
2. Disable false bootstat warnings
> bcdedit.exe /set {current} bootstatuspolicy ignoreallfailures
3. Enable EWF on all partitions
> ewfmgr.exe /all /enable
4. Restart to have the command take effect
> shutdown.exe /r /t 0
5. Activate HORM
> ewfmgr.exe C: /activatehorm
6. Capture the HORM state by hibernating the machine once
> shutdown.exe /h
7. Resume the machine and start using HORM. At this point each
restart should result in a resume from the state captured in
the previous step
8. To change the hibernation file state you simply set the
computer in the state you want (start applications etc.), press
WIN+R and execute command "shutdown -h" to create a new
hibernation file.
```

Please refer to the Microsoft HORM documentation, [8], on how to set up and use HORM.

9. Software update and recovery

9.1. Restore firmware settings

CCsettings can be used to reset the firmware settings to the factory default settings, if needed. Start *CCsettings* and press the *Factory defaults* button from the *Advanced* page.

9.2. Updating firmware components

The XM2 device has three different firmware components; The System Supervisor is a microcontroller that controls the power configuration within the device. The Front microcontroller controls the functionality available in the front, including LED, Buzzer and touch screen handling. The FPGA implements different communication controller interfaces internally in the device.

Each of these firmware components can be updated, either individually or batched. The preferred method to perform the update is by using the *CCSettings* program, as briefly mention in section 5.15. It is also possible to use the command line based application *ccsettingsconsole*, or from your own program using the CC AUX API functions.

For specific details on how to perform firmware component updates, please see [10] CCpilot XM2 and CrossCore XM2 – Performing Firmware Updates.

9.3. Updating system components

Updated system components can be installed in XM2 device to enable new features or provide enhancement on existing functionalities.

- XM2 device specific drivers, interfaces and applications are updated using updated software packages available from maximatecc or your sales contact. These packages contain installation instructions or such instructions are available separately.
- Firmware for XM2 device internal functionality is mainly updated through *CCsettings*.
- General operating system updates are added using the operating system update methods, such as installers and software packages in Windows. For Linux, the next section covers the operating system update.

9.4. Linux OS update

9.4.1. Updating operating system

The Linux system on an XM2 device can be updated by an administrator user via operating system binary images. The update process can also be used for resetting the device to the default state.

Warning: Errors during an update can set the module in an unrecoverable state. In such case, the module must be then shipped to factory for repair, or have internal parts replaced through service interfaces.

9.4.2. Released image files

New versions of the operating system for the device are released as a set of binary format image files as well as additional configuration files and scripts required for update.

The complete system consists of four main parts in the CFast card: bootloader, main system root file system, backup system root file system and user defined area. Bootloader part also contains main and backup system kernels. Normally, software updates concern only main kernel and root file system, occasionally also the other parts.

9.4.3. Update automation

9.5. This example shows how to automate remote update so that user Similar automation can also be saved to USB memory; so that inserting the working device will execute an update without interaction. This method was introduced in section 7.8 USB memory installer

If a USB memory is inserted before start up, it is possible to activate a run time hook to enable automatic software execution. For instance, this function is suitable for production time installers, or automated SW updates.

If you add a script (executable file) that's called **cc-auto.sh** (only that) in the root of a USB memory, it will be executed during device startup. The USB memory can be a FAT-formatted one or with other suitable file system, so there is no specific requirement there, but remember that some Windows based editors will leave Windows EOL characters in edited files, including scripts. Such characters may or may not affect the execution of this type of script.

The *cc-auto.sh* script is then executed automatically at insertion of USB memory or during start up. By placing commands which copy new applications and perform updates and installations in that script, this feature can be used for creating auto-update of user and system software. There are no specific limits on what user can do with *cc-auto.sh file*, but it's recommended that all desired commands are applied within the *cc-auto.sh* script process.

9.5.1. Example of automatic software installation with USB memory

This is an example that enables operating system update to be automated, but it can be used as an illustration on what can be done with the installer script as such. This script solution is to be used for the backup system solution only, that's not needed in the normal user software update case.

5. *cc-auto.sh* creates folder */usr/local/my_application/*.
6. *cc-auto.sh* copies all necessary files to that folder or to other folders from USB memory.
7. *cc-auto.sh* performs additional application setup actions.
8. *cc-auto.sh* may run *reboot* but it is not necessary. If reboot is written, remember to remove memory when restarting, otherwise the device will enter a reboot loop since the script is also run at startup.

At this point the USB memory can and should be removed, and device should reboot itself and start the installed applications. A small script example is given below:

```
#/bin/sh
# Automatic installation script example

mkdir -p /opt/my_application
```

```
tar -C /opt/my_application -xzf
/media/usb/my_application.tar.gz

ldconfig -C /opt/etc/ld.so.cache

sleep 10

reboot
```

9.6. Media files playback

Video and audio playback is supported by gstreamer multimedia framework, which supports multiple video (MPEG2, H.264/AVC, DivX, Xvid) and audio codecs (wav, MP3, AAC) and container formats (MPEG, AVI, mkv). Note that the CCpilot XM2 currently supports only codecs for WAV audio playback. Video player is not included in the system. Audio file can be played with the command:

```
# aplay <audiofile>
```

This command launches a player and plays the file automatically so it can also be launched from a script. For custom applications, gstreamer-0.10 C API can be used (see documentation at http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html).

There is also a Qt frontend system available for gstreamer as well, which is called Phonon.

.

9.6.1.1. Updating automatically

Power-up the device, make sure the login less SSH access is set up properly, or perform that step manually. Copy the update image files, *fullup.sh* script to the device into */usr/local/fw_update/* -folder. USB memory or NFS mount can all be used for copying. Then execute **reboot-rescue.sh**

For example:

```
scp -r fw_update -i xm_rsa root@X.X.X.X:/usr/local/
ssh -i xm_rsa root@X.X.X.X "/usr/bin/reboot-rescue.sh"
```

Unit will reboot into secondary system, and then update the main system using the files copied above, and then reboot back into normal system.

9.6.1.2. Example of automatic system software update with USB memory

This is an example that enables operating system update to be automated, but it can be used as an illustration on what can be done with the installer script as such. This script solution is to be used for the backup system solution only, that's not needed in the normal user software update case.

9. *cc-auto.sh* creates folder */usr/local/fw_update/*.
10. *cc-auto.sh* copies all necessary files to that folder, including file called *fullup.sh*.
11. *cc-auto.sh* runs *reboot-rescue.sh*.

    At this point the USB memory can and should be removed.

12. Device reboots to backup system.
13. Backup system executes */usr/local/fw_updata/fullup.sh*.

14. */usr/local/fw_updata/fullup.sh* does whatever it wants and then deletes the */usr/local/fw_updata/* folder to prevent these actions from executing twice.
15. The last thing the script does is to reboot the device again, into normal operating mode.

9.6.2. Updating backup system

Note: Updating only backup system does not involve or affect main system.

When the backup system requires update, it is updated from the main system side. If the main side is non-operational, then update the main side first and then continue here.

9.6.2.1. Preparing update within Linux

Copy the backup system update images as listed:

- ccpilot-*xm2-bs_kernel.bin,*
- *ccpilot-xm2-bs_rootfs.bin*
- *ccpilot-xm2-bs.md5sum*
- *fullup.sh*

to unit */tmp/ folder*. Copy method choice is free; **scp**, USB memory or NFS mount can all be used.

Access the Linux console as **root** user, either over SSH connection from other host or serial console terminal access to Linux.

9.6.2.2. Updating unit within Linux

Prerequisite: New file images are at the devices */tmp/* -folder. Update tool command **fullup.sh** is already present, or it can be copied to target explicitly.

Warning: All excess processes that might interfere or interrupt the update process should be terminated.

9.6.2.2.1. Upgrading whole secondary system

```
/tmp # ./fullup.sh -s
```

Note the -s flag, without it the normal side is updated!

If the -s flag is not recognized (first output line reads: "*Reflashing system with new images of kernel and root-fs.*") the version of *fullup.sh* is too old, locate a newer version and use it.

9.7. Reinstalling the operating system

If the operating system needs to be reinstalled it is advised to use the custom XM2 device operating system images, available through maximatecc's on-line support pages or your support contact. These operating system images have the same content as the pre-installed operating system. There is a separate operating system recovery manual that can be followed, see [9] CCpilot XM2 and CrossCore XM2 – Performing Operating System Recovery for a more thorough description.

When re-installing XM2 device, all stored material such as user saved data and computer configuration will be lost. Backup all necessary information before re-installing the operating system. It will also remove additional software packages installed.

Operating system licensing

OS	OS Version	Licensing
Linux	Yocto version 1.5.3	Linux is distributed under the GNU General Public License (GPL) licence. http://www.yoctoproject.org/
Windows	Microsoft Windows Embedded Standard 7	maximatecc's standard WES7 image for XM2 device requires the WS7P SKU (licence); the tablet PC package included in the image requires this license. www.microsoft.com/windowsembedded

For specific versions of 3rd party software, please see each operating systems system contents chapter above.

Technical support

Contact your reseller or supplier for help with possible problems with your XM2 device. In order to get the best help, you should have access to your XM2 device and be prepared with the following information before you contact support.

- The part number and serial number of the device, which you find on the brand label.
- Date of purchase, which is found on the invoice.
- The conditions and circumstances under which the problem arises.
- LED indicator flash patterns.
- Prepare a system report on the device, from within *CCsettings* (if possible).
- Detailed description of all external equipment connected to the unit (when relevant to the problem).

Trade Mark, etc.