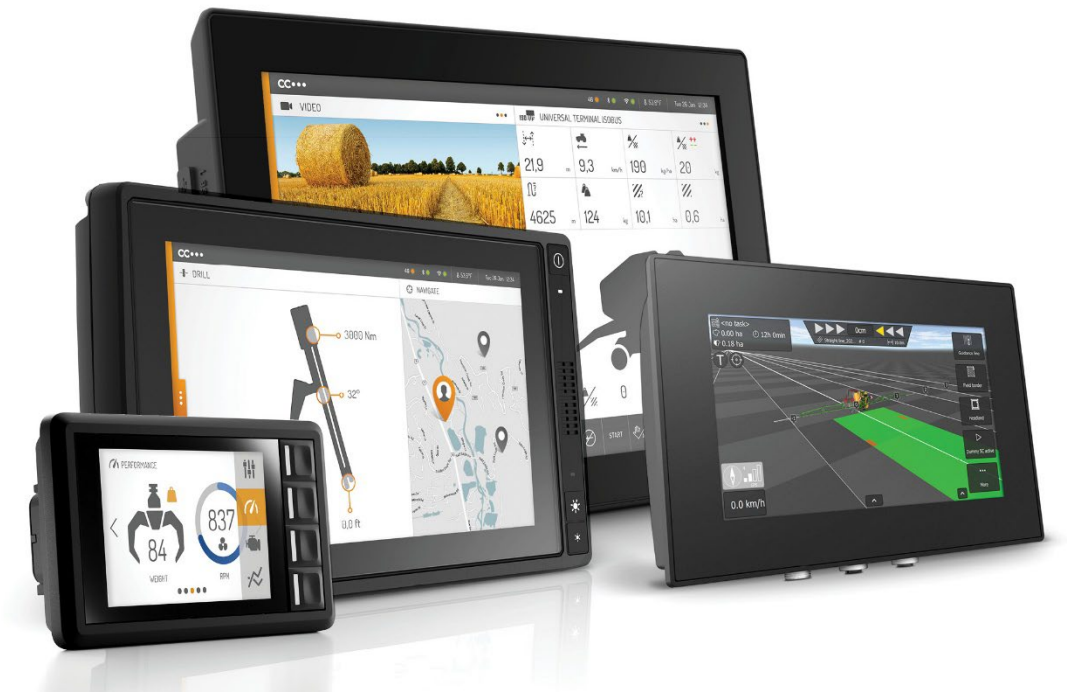
Revision: 2.0.8
2024-04-10

CC Linux

Software Guide



crosscontrol
www.crosscontrol.com

Contents

Revision history

Rev	Date	Comments
1.4.0	2018-11-19	Released. Document compatible with devices running CC Linux 1.4.x.x
1.4.1	2019-08-26	Added information about peripherals needing reconfiguration after a resume from suspend event. Document compatible with devices running CC Linux 1.4.x.x
1.4.2	2019-12-11	Added CCpilot X900 related information.
2.0.0	2021-01-18	Document compatible with devices running CC Linux 2.0.x.x
2.0.1	2021-10-18	Added V1000/V1200 related information.
2.0.2	2022-03-03	Added Yukon development board related information, table for CC Linux 2.0 supported devices and Display MCU firmware update description.
2.0.3	2022-04-27	Add a section for Triggers in Software configuration possibilities.
2.0.4	2022-08-26	Minor Revisions.
2.0.5	2023-03-03	Yukon now supports CfgIn and PWMOut
2.0.6	2023-04-14	Document compatible with devices running CC Linux 3.x.x.x
2.0.7	2023-10-30	Updated document with info about V510/V710
2.0.8	2024-04-10	Removed references to old platforms from the document

1. Introduction

The devices contained within the CC Linux platform are powerful display computers, communication devices and controllers with a rich set of integrated functions. Together with the CC Linux operating system, they form an open platform that facilitates easy implementation of reliable controls.

With the introduction of CC Linux 3.0 there have been several improvements to the system, which will break compatibility with previous releases of CC Linux.

- **Sysvinit replaced by systemd**: One of the biggest changes is the replacement of sysvinit with systemd. This has been a growing trend with major Linux distributions and has been seen as an advantage for CC Linux users also. The drawback is, that old sysvinit style scripts, as well as udev-rules do not directly migrate to the new system. There are several documents explaining the use of systemd in general that can be found on the internet.
- **Overlayfs introduced**: Previously the rw-part of the filesystem has been mounted as a separate directory and specific configuration files has been given symbolic links to this partition. To make the process easier for the user, overlayfs has been introduced. This overlays the root filesystem with another writeable filesystem that overlays edited files. Thus, files can now be directly modified and overwritten in the visible file system and restored if needed.

For specific details in CC Linux, please refer to the programmer's manual.

1.1. Scope

This document is intended for anyone handling a device running the CC Linux platform or developing software for such a device. This document is not intended as a complete reference documentation of the CC Linux devices, software, or software development tools. It is intended to introduce the development engineer to the hardware and software, by providing a top-down overview and summary of the device and a description of the intended use. It is also intended to introduce the reader to the software on the target and the host software development tools.

All devices included in the CC Linux platform (at the time of writing: CCpilot X900 and CCpilot V700, CCpilot V1000, CCpilot V1200, CCpilot V510, CCpilot V710, CCpilot V705 and the Yukon development board) are covered within this document. Depending on which CC Linux device model you are using, some features mentioned in this document might be unsupported. If so, it will be clearly stated using a greyed-out icon as shown below. If no icons are used, the feature is supported by all device models.

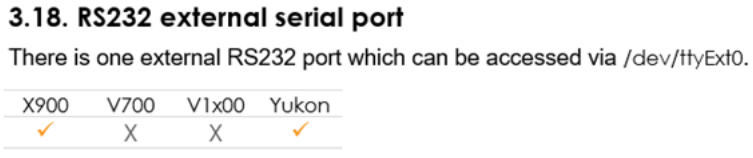


Figure 1: *Example of a feature supported by X900 and the Yukon development board and unsupported by CCpilot V700, CCpilot V1000 and CCpilot V1200.*

There may be slight differences in behavior of supported features depending on your device model. If so, there will be device specific chapters covering the details at the end of this document. Any such occurrence will be clearly stated in the text.

1.1.1. Currently supported devices on CC Linux 3.0

New products break new ground. This also means that some work is needed to make newer versions of the operating system support older products. This document, which is the Software Guide for CC Linux 3.0, contains information on what features, and functionalities are supported for each product as described in 1.1. Some older devices are not yet supported by the new version of the operating system, thus for those, the “supported devices” table on each occurrence throughout the document can be seen as a hint on what will be supported in CC Linux 3.0 on that particular device when support for it has been implemented. Please contact CrossControl for information about roadmaps of operating system versions and corresponding support for any device of interest.

CC Linux 3.0 supported devices:

X900	V700	V1x00	Vx10	Yukon
X	✓	✓	✓	✓

1.2. References

[1] CC Linux– Programmer’s Guide
[2] CCpilot VS– Technical Manual
[3] CCpilot VI – Technical Manual
[4] CCpilot X900 – Technical Manual
[5] CCpilot V700 – Technical Manual
[6] CCpilot V1000/1200 – Technical Manual
[7] CCpilot V510/V710 – Technical Manual
[8] CCAux API reference documentation
[9] http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html
[10] https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt
[11] https://github.com/systemd/systemd
[12] https://freedesktop.org/wiki/Software/systemd/
[13] https://crosscontrol.com/support/knowledge-base/solution-recipes/
[14] https://crosscontrol.com/software-solutions/application-modules/screen-sharing/#downloads/devicedownloads/ccpilot%20v700

2. Basic operation

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

2.1. Login and passwords

By default, the system will boot to a default console and no graphical user interface is launched. For access, the login credentials as stated in Table 1 are required. Additional users can be added using the adduser Linux program.

Table 1: Default login credentials

Username	Password	Access
root	suseroot	Full administrator
ccs	default	Full administrator using sudo

These passwords are publicly accessible and should be immediately changed on first boot in order to avoid security breaches. Issue the following commands using either SSH or on screen terminal.

Change password of root user:

```
$ sudo passwd
```

Change password of ccs user:

```
$ sudo passwd ccs
```

2.2. Using the graphical environment

CC Linux provides Weston/Wayland as the default graphical environment. However, it is up to the customer to choose the default bootup environment of the device so Weston does not start up by default on boot. Refer to chapter 6.2 on how to start and/or enable Weston on boot.

2.3. Using the touch screen

To use the touch screen, a graphical application or Weston will need to be enabled.

2.3.1. Weston and included demo

To test the touch, Weston includes a demo application Weston-smoke. Launch a console by touching the console icon on the top left of the screen in Weston. Then, launch Weston-smoke by typing:

```
$ weston-smoke
```

2.3.2. Crosscontrol demo applications

To install and use the demo applications, see chapter 5.

Navigate the start-up screen application using the touch screen with a stylus or finger.

- Tap the screen to perform an action equivalent to a left mouse click.
- Two simultaneous touches can be recognized to implement pan, zoom and pinch, see chapter 5.4 for an example.

X900	V700	V1x00	Vx10	Yukon
✓	✓	X	✓	X

2.4. Print screen

It is possible to take screenshots using the weston-screenshooter included in the window manager. To take a screenshot, connect an external keyboard to the device and press Super+S (on Windows keyboards, the Super key is the Windows key). The resulting image will end up in the / directory.

2.5. Software deployment

There are several methods to add your own software to the device. The standard methods to transfer software to the device are to either copy files using a network connection or to use USB storage devices, with manual or automatic copy functions. 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.

2.5.1. Device start-up behavior

At power on, the device has an internal microcontroller that monitors the power supply and performs start-up configurations and power settings, the System Supervisor (SS). The SS then supplies the main processor and peripherals with power; from there the device execution begins, starting with the BIOS and/or boot loader, depending on your device model.

Most of these options can be configured from CCSettingsConsole. See chapter 4.1 for details. Additionally, the CCAux API can be used to configure these settings from a user application.

2.6. BIOS

The BIOS is a 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 start-up 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 start-up. 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.

X900	V700	V1x00	Vx10	Yukon
✓	X	X	X	X

2.7. Boot loader

The boot loader is the first software block that executes in the main processor. It serves the purpose of setting up the main processor peripherals and timings and will then load the Linux kernel into RAM memory. Once finished, the operating system execution takes over.

It is possible to connect to the boot loader through a debug serial port with special equipment attached to the device. The reason for connecting to the boot loader would be for debug purposes

only, where specific settings are needed. Detailed information about the boot loader access can be requested from CrossControl if needed.

2.8. Linux system start-up specifics

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

- Main system (normal, default):

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 system (backup):

This is only for device updates and recovery use. It contains only basic maintenance tools. Updating the main system should be done from rescue system. Most of the hardware is not accessible in the rescue system.

Both modes are completely separate Linux operating systems, each of which is a custom-built Linux version 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 systemd [9] 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 targets. 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.

2.9. Status indication (LED and Buzzer)

The buzzer and/or status LEDs (or backlit soft keys, depending on your device model) will be used to indicate the different running states of the device. For a detailed description, refer to the device specific chapters 10.1 (X900), 11.1 (V700) and 12.1 (V1x00/Vx10 and Yukon).

For error indication, all device models behave the same, as described in 2.9.1

2.9.1. Error indication

If an error occurs, the device will indicate the type of error by blinking the status LEDs (or backlit soft keys) or beeping the buzzer in different patterns. The device may be restarted by a button or ignition signal and, depending on whether the error is severe, the device may start normally or go back to error indication.

The reason for fatal error situations could be voltage levels out of range, temperature related problems or internal hardware errors. First steps should be to let the device cool off, and verify it has a correct power supply attached, before starting the device again.

Some devices will beep the buzzer in the same pattern as the LEDs in order to indicate the error.

See the *Technical Manual* of your device model for a more complete description of the error indication and a list of possible error codes.

3. Accessing and using the interfaces

This section covers basic usage and access of the device hardware. Most of the hardware is accessed using the default Linux interfaces but some device specific interfaces may require additional software and/or interfaces to be accessed. See the *CC Linux – Programmer's Guide* documentation for general information regarding software development using the device interfaces.

There may also be additional methods to access the device interfaces than the ones described herein, depending on additional installed software or connected hardware.

3.1. Storage and file system

Depending on your device model, the device uses an eMMC or CFast based storage. The storage is partitioned into protected operating system parts and writable user parts. The filesystem abstracting the storage is overlayFS on top of ext4. OverlayFS provides a great way to merge filesystems such that one of the filesystems (called the "lower" one) never gets written to, but all changes are made to the "upper" one.

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

There is however a trade-off that an application needs to make here if the data to be saved is mission critical or not. An application should not cache files in the eMMC/CFast file system, since in case of a sudden power loss, the eMMC's/CFast's writable partition is made write-protected to protect the files from being corrupted. An application needs to be careful when writing large files, as it can cause pro-longed write-protect sequences, which is a potential hazard to the file system and eMMC/CFast.

The eMMC/CFast is partitioned into two root file systems, which are write protected, and one user file system area, which is write enabled by default. Table 2 and Table 3 show the file system layouts for the main and rescue systems respectively.

In both main and rescue mode, any attached USB memory is automatically mounted once inserted. Supported formats for USB-memory include FAT types which is the default format for USB-memories. USB-memory devices are never automatically formatted so if the file system is unsupported, the device will remain unmounted.

Table 2: File system layout for main system

Mount point	Filesystem	Mount status	
/	Overlay	rw	Virtual merge of lower and upper dirs
/mnt/.rootfs/ro	Ext4	ro	Lower dir for overlay (original root filesystem)
/mnt/.rootfs/rw	Ext4	rw	Upper dir for overlay (work directory for overlay)
/media/usbsda1	Vfat	rw	First USB memory, if available
/media/usbsdb1	Vfat	rw	Second USB memory, if available
/tmp	tmpfs	rw	RAM memory, for storing temporary files
/var/volatile	tmpfs	rw	RAM memory for storing logs etc. during runtime

Table 3: File system layout for rescue system

Mount point	Filesystem	Mount status	
/	Overlay	rw	Virtual merge of lower and upper dirs
/rescue/rootfs/ro	Ext4	ro	Lower dir for overlay (original rescue root filesystem)
/rescue/rootfs/rw	Ext4	rw	Upper dir for overlay (work directory for overlay)
/media/usbsda1	Vfat/ext4	rw	First USB memory, if available
/media/usbsdb1	Vfat/ext4	rw	Second USB memory, if available
/tmp	tmpfs	rw	RAM memory, for storing temporary files
/var/volatile	tmpfs	rw	RAM memory for storing logs etc. during runtime

3.1.1. User file storage

In previous versions of CC Linux the system has forced users to save files under /opt. This is no longer necessary but can still be used if it is logical to the user.

On the other hand, CC Linux provides the OPKG packaging system which should be used to install and remove user applications and thus will provide proper locations to the files. Refer to the Programmer's Guide for more information.

3.2. CAN

The device has up to four CAN interfaces with user configurable baud rate and frame type accessible from the CCSettingsConsole application. The CAN interfaces can also be accessed with the Linux operating system standard API SocketCAN. More information can be found in the *CC Linux – Programmer's Guide*.

3.3. CAN-FD

In order to speed up the transfer of larger data, CAN with flexible data rate (CAN-FD) can be used. For integration with existing CAN networks, each CAN-FD port can also be used as a regular CAN port. The CAN-FD interfaces are accessed via SocketCAN similarly as regular CAN.

X900	V700	V1x00	Vx10	Yukon
✓	X	✓	✓	✓

3.4. Ethernet

The device is set up to use DHCP for IP address retrieval. The network connection settings can be changed within the operating system settings, i.e. by using the network interfaces file as described in chapter 6.5.1.

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

3.5. Wireless LAN

The network connection settings can be changed within the operating system settings, i.e. by using the network interfaces file as described in chapter 6.6.

X900	V700	V1x00	Vx10	Yukon
X	X	✓	✓	X

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

3.6. Bluetooth

The connection settings can be changed within the operating system settings, as described in chapter 6.7.

X900	V700	V1x00	Vx10	Yukon
X	X	✓	✓	X

3.7. USB

A multitude of peripherals can be connected to the device via USB. For some peripherals, drivers compatible with the operating system must be installed in order to enable correct function. For such installations, please contact CrossControl for support in adding a suitable driver or follow the instruction from the device manufacturers.

3.8. Video in

The video-in signal is not available in any of the current products.

3.9. Configurable inputs

The configurable input channels are available for software developers using the CCAux API. Parts of the functionality available can be viewed or set within CCsettingsConsole for test purposes.

For additional technical details, please see the *Technical Manual* of your device model and the *CCAux API Reference Documentation*.

X900	V700	V1x00	Vx10	Yukon
X	X	X	✓	✓

3.10. Analog inputs

The analog input channels are not available in any of the current products.

3.11. Digital Outputs

The user-settable digital output signals are not available in any of the current products.

3.12. PWM Outputs

The user-settable PWM output signals are available for software developers using the CCAux API. The output settings can be viewed or set within CCSettingsConsole for test purposes.

Please note that the PWM outputs are turned off upon suspend and need to be reconfigured when resuming. The CCAux API function **PowerMgr_hasResumed** can be called from within the user application in order to detect a resume from suspend event.

For additional technical details, please see the *Technical Manual* of your device model and the *CCAux API Reference Documentation*.

X900	V700	V1x00	Vx10	Yukon

X	X	X	✓	✓

3.13. Backlight

Users can adjust the screen backlight intensity level. The backlight functionality can be controlled from CCSettingsConsole, via software using the CCAux API, and via backlight buttons.

The most recent setting is always used, it is saved between restarts.

3.14. Ambient light sensor

The ambient light sensor measures light levels in front of the device. The ambient light sensor is accessed through the CCAux API. It can also be accessed for diagnostic through CCSettingsConsole.

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

X900	V700	V1x00	Vx10	Yukon
✓	✓	✓	✓	✓

3.15. Buzzer

The device is equipped with a buzzer which 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 CCSettingsConsole.

Please note that the buzzer is turned off upon suspend and needs to be reconfigured when resuming. The CCAux API function **PowerMgr_hasResumed** can be called from within the user application in order to detect a resume from suspend event.

On some device models, the volume can be considered very loud when standing close to the device. Increment the volume gradually and use appropriate ear protection.

3.16. Temperature sensors

Several temperature sensors are placed internally in the device. It is possible for an application to retrieve temperature information from the temperature sensors through the CCAux API.

3.17. Buttons

Depending on your device model there are up to 10 buttons with configurable functionality. The configuration of each button can be set using either CCAux API or CCSettingsConsole.

Table 4 lists all available configurations for each device model. For additional details, see your device model's *Technical Manual*. *Note that the Vx10 and Yukon uses a new button API where the term "action trigger" is not used.*

Table 4: Available button configurations. Configuration can be set individually for each button.

Configuration	Description	Supported on device				
		X900	V700	V1x00	Vx10	Yukon

Only MP action	Button is not handled in SS. The button can be used in user software via the input event API. Button events can be read from the device node /dev/input/event2	No	No	No	No	No
Start-up trigger	Button is used by SS to trigger a start-up action; powers up the system or wakes up from suspend state (if supported).	Yes	No	No	Yes	Yes [2]
Action trigger	Button is used as an action trigger (available actions listed in Table 5.) It is possible to choose one action for short presses and a different action for long presses.	Yes	No	No	Yes	Yes [2]
Start-up & Action trigger	Button is used as both Start-up and Action trigger.	Yes	No	No	Yes	Yes [2]
BL decrease	Button decreases display backlight intensity.	No[1]	No	No	No	No
BL increase	Button increases display backlight intensity.	No[1]	No	No	No	No
BL decrease & start up	Button decreases display backlight intensity and acts as a start-up trigger.	No[1]	No	No	No	No
BL increase & start up	Button increases display backlight intensity and acts as a start-up trigger.	No[1]	No	No	No	No

Table 5: Available actions for short/long presses on a button configured as an 'action trigger'. Actions are set globally for all buttons.

Action	Supported on device				
	X900	V700	V1x00	Vx10	Yukon
No action	Yes	No	No	Yes	Yes [2]
Shutdown	Yes	No	No	Yes	Yes [2]
Suspend	Yes	No	No	Yes	Yes [2]

[1]X900 has dedicated buttons for BL increase and decrease.

[2] Yukon has a dedicated power button.

X900	V700	V1x00	Vx10	Yukon
✓	X	X	✓	✓

3.18. RS232 external serial port

There is one external RS232 port which can be accessed via /dev/ttyExt0.

X900	V700	V1x00	Vx10	Yukon
✓	X	X	X	✓

3.19. RS485 external serial port

An external RS485 is not available in any of the current devices.

3.20. Speaker

The speaker is controlled using the normal operating system functionality from the ALSA libraries, including sample applications. The audio device is identified as Cirrus Logic HD Audio.

X900	V700	V1x00	Vx10	Yukon
✓	X	X	X	X

3.21. Audio Out

The audio out is controlled using the normal operating system functionality from the ALSA libraries, including sample applications. The audio device is identified as Cirrus Logic HD Audio.

X900	V700	V1x00	Vx10	Yukon
✓	X	X	X	X

4. Pre-installed console applications

In addition to the standard Linux utilities the device comes pre-installed with a few CrossControl developed console applications. The purpose of the pre-installed applications is to provide a quick and easy overview of the device's function and enable easy device control.

4.1. CCSettingsConsole

CCSettingsConsole is a console application for controlling the device settings (LED and buzzer settings, on/off behavior, etc.) using the CCAux API.

CCSettingsConsole can be used to alter the way the device starts up or shuts down, update firmware and change important settings. Only change these settings if you know what you are doing.

CCSettingsConsole can be run either remotely over ssh or locally by connecting a keyboard to the device and opening a terminal from CCLauncher or Weston, depending on your device settings.

To get a description of how to use the application, run the following command:

```
$ sudo ccsettingsconsole --help
```

Settings are grouped into categories like LED, buzzer and Diagnostics, to name a few. The following command is helpful to get a complete list of available categories their possible settings:

```
$ sudo ccsettingsconsole --list
```

For *each category* it is possible to:

- read all current settings at once:

```
$ sudo ccsettingsconsole --category
```

- read one setting:

```
$ sudo ccsettingsconsole --category --setting
```

- set one setting:

```
$ sudo ccsettingsconsole –category --setting=value
```

4.1.1. Buzzer settings example

As an example, run the following command to get the current buzzer settings:

```
$ sudo ccsettingsconsole --buzzer
Buzzer Frequency: 2600
Buzzer Volume: 400
Buzzer status:  Disabled.
```

And the following to change the buzzer settings to 1000 Hz frequency, volume 200, enabled:

```
$ ccsettingsconsole --buzzer --frequency=1000
Buzzer frequency set to: 1000 Hz.
$ ccsettingsconsole --buzzer --volume=200
Buzzer volume set to: 200
$ ccsettingsconsole --buzzer --status=Enable
Buzzer status set to enable.
```

4.2. CCSystemReport

CCSystemReport is a utility program to print out various system diagnostics on the console. To execute the program, run the following command:

```
$ ccsystemreport
```

The diagnostics information contains following information depending on the device:

- Installed software versions
- EEPROM data
- System diagnostics
- Analog to Digital Converter (ADC) values
- CPU board information
- Settings and log files

4.3. CCAux daemon

The CCAux daemon (ccauxd) is a background service designed to:

- Read status and event requests from the SS and initiate system restart and shutdown sequences, for example events triggered by button presses.
- Set the button status LED (or backlit soft keys) according to a predefined set of rules.
- Manage automatic backlight levels.
- Implement the PowerMgr feature set, which is configurable from the CCAux API.

The daemon can be removed if needed functionality is implemented in user software.

```
$ sudo systemctl disable ccauxd.service
```

5. Additional optional applications (graphical)

In addition to the console applications, some other applications are provided to ease access and control of the device. These applications can be separately downloaded for the specific device from the support page [11].

5.1. CCVNCServer

VNC server package can be downloaded the from the support site [14]. The package includes installation instructions, documentation, and archive for installation on target.

X900	V700	V1x00	Vx10	Yukon
X	✓	✓	✓	✓

6. Software configuration possibilities

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.

6.1. Installing new drivers, applications and system packages

With the introduction of overlayfs, applications can be installed to any location just as other general Linux-based systems. To see the modified files, the rw-partition can be viewed in the location referred to in chapter 3.1.

6.1.1. Mounting user partition in rescue system

Normally, the eMMC user partition is not mounted in the rescue system. If access to the user partition is needed from the rescue system, it can be mounted to /main_rootfs manually:

```
$ sudo mount-main-rootfs.sh
```

When changes are done, unmount using:

```
$ sudo umount-main-rootfs.sh
```

6.1.2. 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 <LOWER_DIR_FOR_OVERLAY>
```

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 <LOWER_DIR_FOR_OVERLAY>
```

6.1.3. User libraries

The recommended way of installing user libraries, is to install them with the application and either use RDPATH at linking time to specify the library location or set the environment variable LD_LIBRARY_PATH before executing the application.

Libraries that are used by multiple application in the system can generally be installed in their default locations (such as /usr/lib).

Note, that this approach may overwrite specific system libraries with ones provided with the user application.

For legacy compatibility and flexible configuration, the library locations can also be defined. To do this, copy the files to their respective location (for example /opt/lib/. Then edit /etc/ld.so.conf to include the required directory. Finally, update the used library cache file by executing the following command:

```
$ sudo ldconfig -C /etc/ld.so.cache
```

If additional library file locations are needed, the paths of these can be added to above command as parameters.

The 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.

6.1.4. User binaries

User binaries can be installed in any location in the system. It is advisable to use the provided package manager to manage these binaries. Refer to the programmer's manual for more information.

If additional levels of binaries are required, the *$PATH* environment variable must be updated under /etc/profile

6.2. Systemd Management

Systemd is a system and service manager for Linux operating systems. When run as first process on boot (as PID 1), it acts as init system that brings up and maintains userspace services. Separate instances are started for logged-in users to start their services.

Systemd is usually not invoked directly by the user, but is installed as the /sbin/init symlink and started during early boot. The user manager instances are started automatically through the user@.service service.

The basic object that ***systemd*** manages and acts upon is a "unit". Units can be of many types, but the most common type is a "service" (indicated by a unit file ending in ***.service***). To manage services on a ***systemd*** enabled device, our main tool is the ***systemctl*** command.

The user has the possibility to start applications and scripts by modifying or adding systemd services.

We can start the service by typing:

```
$ sudo systemctl start weston.service
```

To stop it again, type:

```
$ sudo systemctl stop weston.service
```

To restart the service, type:

```
$ sudo systemctl restart weston.service
```

To attempt to reload the service without interrupting normal functionality, type:

```
$ sudo systemctl reload weston.service
```

Most systemd service files are not started automatically at boot. To configure this functionality, you need to enable the service. This hooks it up to a certain boot target, causing it to be triggered when that target is started.

To enable a service to start automatically at boot, type:

```
$ sudo systemctl enable weston.service
```

Disable the service, type:

```
$ sudo systemctl disable weston.service
```

There is a great deal of information one can pull from systemd to get an overview of the system state.

To list all the units ***systemd*** has loaded or has attempted to load into memory, including those that are not currently active, type:

```
$ sudo systemctl list-units --all
```

A systemd component called ***journald*** collects and manages journal entries from all parts of the system. This is basically log information from applications and the kernel.

To see all log entries from current boot, type:

```
$ sudo journalctl -b
```

For more information on ***journalctl***, see:

```
$ journalctl --help
```

To see an overview of the current state of a service, you can use the *status* option with the ***systemctl*** command.

```
$ systemctl status weston.service
```

6.2.1. Systemd targets

In systemd, targets are basically synchronization points which the device can use to bring the device into a specific state. Service and other unit files can be tied to a target and multiple targets can be active at the same time. See Table 6 for a short description of systemd targets.

Note, that with sytstemd, **runlevels are no longer used**. A note from the manual:

```
"Runlevels" are an obsolete way to start and stop groups of services used in
SysV init. systemd provides a compatibility layer that maps runlevels to targets,
and associated binaries like runlevel. Nevertheless, only one runlevel can be
"active" at a given time, while systemd can activate multiple targets
concurrently, so the mapping to runlevels is confusing and only approximate.
```

```
Runlevels should not be used in new code, and are mostly useful as a shorthand
way to refer the matching systemd targets in kernel boot parameters.
```

Note, that not all SysV-runlevels do have separate targets on systemd.

Table 6: Systemd targets (with reference to SysV)

Mode (SysV)	Systemd target	Description
To halt the system (0)	poweroff.target, systemctl halt	Shutdown
Single user mode (1)	*rescue.target*	*Administrative tasks*
Multi User (2)	multi-user.target	Does not configure network interfaces nor export network services
Multi User with Network (3)	multi-user.target	Start the system normally
Experimental (No User) (4)	multi-user.target	Not used/User-definable
Multi-user with Graphical & Network (5)	graphical.target	Same as multi-user.target + display manager
To reboot a system (6)	reboot.target, systemctl reboot	Reboots the system
Emergency shell	emergency.target	Emergency shell

All targets available on the device, type:

```
$ systemctl list-unit-files --type=target
```

Default target:

```
$ systemctl get-default
```

6.2.2. Example service

The default target is *multi-user.target*, so applications should at least have a startup service file for this target.

To start applications in *multi-user.target*, create a new service file:

```
$ sudo systemctl edit --full --force mycustomapp.service
```

Example *mycustomapp.service*:

```
[Unit]
Description=mycystomapp description

[Service]
ExecStart=/usr/sbin/mycustomapp start
ExecStop=/usr/sbin/mycustomapp stop

[Install]
WantedBy=multi-user.target
```

After modifying a unit file, you should reload the ***systemd*** process itself to pick up your changes:

```
$ sudo systemctl daemon-reload
```

Start new service:

```
$ sudo systemctl start mycustomapp.service
```

Enable new service on boot:

```
$ sudo systemctl enable mycustomapp.service
```

For more information on ***systemd*** and how the service files should be created, see[10]

6.2.3. Power off or reboot the system

To power off the device, type:

```
$ sudo systemctl poweroff
```

Reboot the device, type:

```
$ sudo systemctl reboot
```

6.3. Text editor

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

6.4. Terminal

It is possible to access a local device by connecting an external keyboard to the device and opening an on screen terminal by pressing down ctrl+alt+F4 keys. The login credentials are stated in chapter 2.1.

6.5. 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.

6.5.1. File method for IP address configuration

This method requires knowledge about the interfaces file format, but a sample is given below.

```
$ sudo nano /etc/systemd/network/eth0.network
```

Sample of network file setting dynamic IP address:

```
[Match]
Name=eth0

[Network]
DHCP=ipv4
```

Sample of network file setting static IP address:

```
[Match]
Name=eth0

[Network]
Address=192.168.1.20/24
Gateway=192.168.1.1
```

```
DNS=192.168.1.1
```

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 systemctl restart systemd-networkd
```

6.5.2. Firewall

The system uses iptables for the firewall. The following systemd units are responsible for the IPv4 and IPv6 firewall:

```
# IPv4
$ systemctl status iptables
# IPv6
$ systemctl status ip6tables
```

You can see the current active firewall rules with the following commands:

```
# IPv4
$ sudo iptables-save
# IPv6
$ sudo ip6tables-save
```

To configure the firewall, the following steps can be used.

Add the new firewall rule:

```
$ sudo iptables -A INPUT -p tcp -s 0/0 -d 0/0 --dport 80 -j DROP
```

Update the firewall configuration:

```
# IPv4
$ sudo iptables-save > /etc/iptables/iptables.rules

# IPv6
$ sudo ip6tables-save > /etc/iptables/ip6tables.rules
```

Reboot and check that the firewall configuration works as desired.

6.6. Wireless LAN

The needed radio firmware is loaded at boot. Use the NetworkManager to connect to a Wi-Fi access point:

```
nmcli device wifi connect SSID password SSID_PASSWORD
```

6.7. Bluetooth

To use Bluetooth, you need to run the following commands as the root user.

```
systemctl enable bluetooth-attach
systemctl start bluetooth-attach
```

Then use bluetoothctl to power on the module and scan:

```
** root@v1200:/usr/bin# bluetoothctl
Agent registered>

** [bluetooth]# power on
Changing power on succeeded
[CHG] Controller 00:16:A4:4A:09:37 Powered: yes

** [bluetooth]# scan on
Discovery started
[CHG] Controller 00:16:A4:4A:09:37 Discovering: yes
[NEW] Device 11:75:58:E5:C3:99 TimeBox-mini-light
[NEW] Device 70:26:05:0D:49:46 LE_WH-1000XM2
[bluetooth]#
```

6.8. Remote access

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

6.8.1. SSH

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

```
$ ssh ccs@X.X.X.X
```

To connect to a host from the 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.

Please note that root access over SSH is disabled.

Access can be re-enabled using the SUDO command by editing /etc/ssh/sshd_config and the line to:

```
PermitRootLogin yes
```

The default login and password can be found in chapter 2.

6.8.2. SCP

To copy a file to the device (while on the host) use the following command (and give password when asked):

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

To copy a file from the device (while on the host) use the following command (and give password when asked):

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

To copy a file to the host (while on the device), use the following command:

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

To copy a file from the host (while on the device), 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.

6.8.3. Password-free login for SSH and SCP

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

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

```
~$ ssh-keygen -t rsa -f v700_rsa
```

Copy (or append) the created *v700_rsa.pub* file into **target** device as a */etc/ssh/authorized_keys* -file.

On the **host**, move the *v700_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/v700_rsa ccs@X.X.X.X
```

6.8.4. Remote command execution

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

```
~$ ssh ccs@X.X.X:X "ls -al /opt/"
```

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

6.9. Weston Graphics

The graphics framework uses the Wayland protocol reference implementation Weston for graphics operations. Wayland is fast and efficient, and is used by most major Linux desktop systems, giving it vast standard support in the Linux user space world.

For instance, Qt has a plugin that enables the Qt libraries to be built for Weston. For Qt applications the impact for that means that it simply needs to be started with a specific flag, -platform wayland, and built with the correct development libraries.

Weston includes a windowing system, enabling several applications to be run overlapped.

By default, Weston is not enabled on start up. This can be easily changed with systemctl:

```
~$ systemctl enable weston
```

6.9.1. Graphical application launch

In order to launch a graphical application on Weston, the XDG_RUNTIME_DIR environment variable needs to be set. This can easily be done by either adding the following line to the application's startup script, or running it from the command line before launching the application:

```
export XDG_RUNTIME_DIR=/run/user/root
```

6.9.2. Fixed window position

By default, Weston will place any opened window at a random position. In order to overrun this feature, Weston has been patched to read window coordinates from the file /tmp/weston_fixed_coordinates. If it doesn't exist, or if coordinates provided are negative, it will revert to using the default random positioning. The file needs to be overwritten with new coordinates for each window to be opened at a new position. A typical use case for this feature is showing two windows split screen. Following is an example script which opens two application windows; a pdf-reader at position (0,0) and a text editor at position (640,0):

```
#!/bin/sh

echo "0 0" > /tmp/weston_fixed_coordinates
/usr/bin/qpdfview &

sleep 5

echo "640 0" > /tmp/weston_fixed_coordinates
/usr/bin/weston-editor &
```

6.10. Graphical Qt-applications without Weston.

With i.MX8-devices using KMS and the proprietary driver, there is a need for special configuration to force the plugin into the correct color mode:

First define the correct mode in kms.json.

```
{
"device": "/dev/dri/card0",
"outputs": [
{ "name": "LVDS1", "mode": "800x480", "size": "800x480", "format":
"abgr8888" } ]
}
```

Then export the necessary configuration parameters:

```
export QT_QPA_EGLFS_KMS_CONFIG="/path-to-directory-with/kms.json"
export QT_QPA_EGLFS_INTEGRATION=eglfs_kms
export QT_QPA_EGLFS_KMS_ATOMIC=1
```

And finally, launch your application with:

```
./Application -platform eglfs
```

There is also a native Vivante platform available, which usage is discouraged as the performance is not on par with EGLFS KMS. To use the EGLFS Vivante platform, export the following configuration:

```
export QT_QPA_EGLFS_INTEGRATION=eglfs_viv
export QT_QPA_EGLFS_FORCEVSYNC=0
export QT_QPA_EGLFS_FORCE888=1
./application -platform eglfs
```

6.11. Boot splash

CC Linux comes with a default boot splash screen containing a black background with a white progress bar. The boot splash used is called psplash; third party software released under the GPL2 license. If desired, it is possible to add a picture/logo and change the colors of the boot splash. In order to do so, the psplash source code needs to be modified and the binaries recompiled. Please refer to the document *CC Linux - change boot splash appearance* which can be found at the Knowledge base on the CrossControl support site.

6.12. Serial Number Broadcast configuration

The device 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 the device boot-up and will broadcast a specific identification message to the local network every fifth second. The frequency of the broadcast can be modified, and the service can be completely disabled using the configuration file /etc/ccsnb.conf. 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=5

# 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, if unset '0' is used. String value
UNITTYPE=V700
```

6.13. Watchdog configuration

The device has watchdog daemon (WDT) that keeps updating watchdog after start. If watchdog is enabled, but it's not updated often enough or system has crashed, watchdog will reset the system. WDT is default implementation of watchdog daemon and user can replace it but implementing same functionality to their own application. Setting timeout to zero disables watchdog functionality. Watchdog configuration file is /etc/watchdog_timeout.conf

```
# Set to 20sec by default. For systems with Systemd see
# DefaultTimeoutStopSec.
# If this is shorter than the systemd value, there is a risk that the watchdog
# will reboot the system before all services have shut down.
#
# Timeout = 0 disables the WDT from ever using WDT, but still keeps WDT
# daemon running. In this case it can be disabled/killed without watchdog
# causing reboot.
#
[wdt]
```

```
Timeout=20
```

6.14. USB memory installer

If a USB memory is inserted after startup, 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, see chapter 7.5.

To activate the automatic execution, add a script named cc-auto.sh to the root of a FAT16/32 formatted USB memory. CC Linux is configured to automatically run any script with this name at insertion of USB memory. 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 is recommended that all desired commands are applied within the cc-auto.sh script process.

Note: 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 automatically gets an argument from the OS containing the path to where the USB memory is mounted. Never use absolute paths to the USB memory from within cc-auto.sh as the path may vary.

6.15. Video file playback

Video file playback is supported by the gstreamer multimedia framework, which supports multiple video (MPEG2, H.264/AVC, H.265/HEVC, DivX, Xvid) and container formats (MPEG, AVI, mkv).

Generally, each supported kind of video can be played with the command:

```
$ gst-launch-1.0 playbin uri=file://path-to-file
```

There are some exceptions on the i.MX8-based devices using VPU decoding. Gstreamer must be explicitly forced to use the imxvideoconvert_g2d-plugin.

```
$ gst-launch-1.0 playbin uri=file://path-to-file video-
sink="imxvideoconvert_g2d ! queue ! autovideosink"
```

Additionally, gplay-1.0 should be used (not gst-play1.0). These limitations are related to the platform using a specifically imx-patched version of gstreamer.

This command can also be used to play video files over a network and plays the file automatically so it can be launched from a script. For custom applications, gstreamer-1.0 C API can also be used, see documentation at [7].

http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html.

6.16. Audio files playback

Audio playback is also supported by gstreamer multimedia framework, which supports multiple audio codecs (wav, MP3, AAC). Note that the device currently supports only codecs for WAV audio playback. Audio files 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.

By default, the device will play on both the speaker and line out simultaneously. To play on one at a time, use **amixer** to mute the unwanted peripheral.

To mute the speaker:

```
#amixer set 'Line Out' mute
```

To mute audio out:

```
#amixer set 'Speaker' mute
```

Note: the speaker in **amixer** is physically wired to the audio out on the device, and vice versa.

X900	V700	V1x00	Vx10	Yukon
✓	X	X	X	X

6.17. Adding user defined fonts to system

CC Linux uses fontconfig to handle installed fonts. Pre-installed fonts reside in /usr/share/fonts/ttf. New fonts can be added to a user specific directory, such as /opt/user-fonts directory. The new fonts need to be added to fontconfig using:

```
$ fc-cache </opt/user-fonts>
```

It is possible to list all the installed fonts by running:

```
$ fc-list
```

6.18. Triggers

The triggers are configurable actions which can be used to request change for the system state. Triggers are driven by external inputs such as ignition signal and buttons. The following categories of triggers are available.

Wakeup triggers

Causes the device to resume from the suspend mode. The wakeup triggers are configured with the *--wakeup_trigger_* prefix in ccsettingsconsole.

Startup triggers

Causes the device to start-up from the Off mode. The startup triggers are configured with the *--startup_trigger_* prefix in ccsettingsconsole.

Action triggers

Causes the device to either suspend or shutdown depending on the trigger configuration. Action triggers are configured with the *--action_trigger_* prefix in ccsettingsconsole.

On/off signal trigger

The on/off signal (device specific implementation) causes the device to suspend or shutdown depending on the trigger configuration. The on/off trigger is configured with the *--onoff_signal* parameter in ccsettingsconsole.

Please note that supported triggers vary depending on the device used. The CCSettingsConsole help text shows the available trigger configurations. The following trigger configurations are under development and are currently not available:

- IO MCU triggers (startup/action/wakeup triggers)
- Ethernet trigger (startup/wakeup triggers)
- CAN trigger (startup/wakeup triggers)

7. Software update and recovery

7.1. Restore firmware settings

CCSettingsConsole can be used to reset the firmware settings to the factory default settings, if needed. Use the following command:

```
$ sudo ccsettingsconsole --advanced --factory
```

7.2. Updating firmware components

The *advanced* category of CCSettingsConsole is also used for loading new SS firmware and Display MCU firmware into the device.

It is also possible to verify that a given file matches the current firmware; with “--verify" a mismatch can be detected and reported. The firmware is not written during verify.

After each program or verify action, the device must be shut down. Press the shutdown button or other alternatives to shut down the OS after a firmware update.

The preferred method to perform the update is by using the CCSettingsConsole application. It is also possible to use the API directly from your own program using the CCAux API functions.

Warning: Errors during an update can set the module in an unrecoverable state. In such case, the module must be shipped to factory for repair, or have internal parts replaced through service interfaces. Make sure that you carefully choose the correct files for updating and follow the instructions from the tools, including powering off the device when prompted.

To update the firmware, copy the firmware file to a USB stick and connect it to the device. Reboot the device to the rescue system using the following command:

```
$ sudo reboot-rescue.sh
```

Login and issue the following commands to update:

7.2.1. SS firmware

```
$ sudo ccsettingsconsole --advanced --update=SS --filepath=<full-path-to-file>
```

If only firmware verification is wanted, use the following command instead:

```
$ sudo ccsettingsconsole --advanced --verify=SS --filepath=<full-path-to-file>
```

7.2.2. Display MCU firmware

Login and issue the following command to update the Display MCU:

```
$ sudo ccsettingsconsole --advanced --update=Display --filepath=<full-path-to-file>
```

If only firmware verification is wanted, use the following command instead:

```
$ sudo ccsettingsconsole --advanced --verify=Display --filepath=<full-path-to-file>
```

X900	V700	V1x00	Vx10	Yukon
X	X	X	✓	✓

7.3. Factory reset of operating system

As explained, user data is stored on the rw-partiton that is overlayed on to the ro-root filesystem. It is possible to remove all the settings and files under that location, and have the device generate the default contents back upon a restart of the device. If you have modified the ro-filesystem, the files there will not be changed back.

Note: This can potentially also remove some applications and files that are part of the factory installation by CrossControl and delivered to you as such (For example LinX and CODESYS installations). 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

```
$ sudo reboot-rescue.sh clear
```

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

7.4. Updating the operating system

The CC Linux system on the device can be updated by an administration user. 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 then be shipped to our factory for repair, or have internal parts replaced through service partners.

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 five main parts in the eMMC/CFast: main kernel, rescue system kernel, main root file system, rescue system root file system, and user defined area. Additionally, there is boot loader software in the first part of the eMMC/CFast. Normally, software updates concern only the main kernel and root file system, though they may occasionally concern the other partitions.

Note: Always update the **main system first**, and **the rescue system second**. Alternatively, in order to update both systems at once, CrossControl recommends using the uuu approach described in *CC Linux – update using uuu (not available for X900)*.

File names will differ depending on which parts are being updated. Additionally, file names will vary depending on your device model. In the below sections, CCpilot V700 will be used as an example.

7.4.1. Updating main system

Warning: An update erases and replaces the old root (ro) file system completely! This should not affect contents of the rw-partition. However, creating a backup of important files is still advised.

This method requires the device to have a working rescue system.

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

Follow these steps to update the main system:

1. Copy the main system update images to /usr/local/fw_update. **The folder name is important, otherwise, the update will not start**. Copy method choice is free; scp, USB memory or NFS mount can all be used.
    - ccpilot-v700_rootfs.bin
    - ccpilot-v700_u-boot.bin
    - ccpilot-v700.md5sum
    - fullup.sh
2. Reboot the device to rescue system:

```
$ sudo reboot-rescue.sh
```

3. Wait for the update to automatically start and reboot to main system when complete.

Example output on the device on-screen terminal:

```
Please wait: booting…

Recovery system vs/dev/tty1
V700 login: ===== CrossControl: Device Update ===== AF
   System Hardware: ccpilot-v700
 Now running on     : BACKUP system
  Requested action: NORMAL system update
      Checking MD5: ccpilot-v700_rootfs.bin: ccpilot_rootfs.bin= OK
OK

           === Listing: Actions: ===
            Updating: root-fs   ALL FILES ON ROOT-FS WILL BE LOST
                        Files on /usr/local and /media cf unaffected

          Verification: Skipped, continuing..

      === Now doing: Requested actions ===

 * DO NOT BREAK PROCESS OR SHUTDOWN THE UNIT before update is complete.
*

Copying new kernel image..
```

```
Copying new root-fs image..
```

7.4.2. Updating rescue system

Note: Updating only the rescue system does not affect the main system. When the rescue system requires an update, it is updated from the main system side. If the main side is non-operational; update the main side first and continue to this step.

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

Follow these steps to update the rescue system:

1. Copy the backup system update images to /usr/local/folder (folder name has no importance when updating rescue system). Copy method choice is free; scp, USB memory or NFS mount can all be used.
    - ccpilot-v700_rescue_rootfs.bin
    - ccpilot-v700_rescue.md5sum
    - fullup.sh
2. Access the Linux console, either over SSH connection from another host, or using a serial console terminal access to Linux.

    Warning: Avoid using the SSH method, if your Ethernet network is susceptible to connection breaks, as the image write can get interrupted.
3. Update the rescue system from within the update folder:

```
/usr/local/folder$ sudo ./fullup.sh -s
```

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

Example output on terminal:

```
===== CrossControl: Device Update ===== AF
  System Hardware: ccpilot-v700
 Now running on   : normal side
  Requested action: BACKUP system update
OK
    Checking MD5: ccpilot-v700_rescue_rootfs.bin: ccpilot-
v700_rescue_rootfs.bin: OK
OK

     === Listing: Actions: ===
       Updating: root-fs     ALL FILES ON ROOT-FS WILL BE LOST
              Files on /usr/local and /media/cf unaffected

  === Verification: ARE YOU SURE ? ===

   - If yes, press Enter to continue.
   - IF NOT, press CTRL+C now to cancel update.
  This is your last chance to do cancel!
```

```
** IF YOU CONTINUE, DO NOT INTERRUPT THE PROCESS UNTIL READY **
```

The process is pausing here, waiting for ‘Enter’ or cancellation.

```
  === Now doing: Requested actions ===

 * DO NOT BREAK PROCESS OR SHUTDOWN THE UNIT before update is complete.
*

Copying new kernel image..
Copying new root-fs image..
131072+0 records in
131072+0 records out
67108864 bytes (67 MB, 64 MiB) copied, 14.0712 s, 4.8 MB/s
Completed. Rebooting..

Broadcast message from root@v700 (pts/0) (Thu Nov  9 15:02:02 2017):

The system is going down for reboot NOW!

End of all actions, H o l d   o n   t i g h t
```

7.5. Update automation

This chapter shows how to use the auto started script cc-auto.sh to automate the update. In these examples the update process is automated to start upon insertion of a USB memory stick.

These examples can be modified in order to use the cc-auto.sh script to update the system remotely over SSH.

7.5.1. Automated update of main system using USB memory

The directory name fw_update must be kept for the update to work properly.

1. Create a directory fw_update in the root directory of the USB memory and copy the update images and fullup.sh script there.
2. Add a script cc-auto.sh to the root directory of the USB memory. Note that the script name must be correct for the update to start. An example of the script contents:

```
echo "Starting release update"

#
# To prevent accidental updates, use this stamp file for it.
# Remove file so if user inserts USB - stick second time the
# update will be started regardless.
#
if [ -e $1/update-done ] ; then
   rm $1/update-done
   exit 0
fi

touch $1/update-done
```

```
cd $1 && cp -a fw_update /opt

reboot-rescue.sh
```

3. Insert the USB stick to the device and the update script will start automatically, without requiring any user interaction to complete.

7.5.2. Automated update of rescue system using USB memory

For the rescue system update, the directory name can be chosen freely, but must match the directory in the created script.

1. Create a directory foldername in the root directory of the USB memory and copy the update images and fullup.sh script there
2. Add a script cc-auto.sh to the root directory of the USB memory. Note that the script name must be correct in order for the update to start. An example of the script contents:

```
echo "Starting rescue update"
#
# To prevent accidental updates, use this stamp file for it.
# Remove file so if user inserts USB - stick second time the
# update will be started regardless.
#
if [ -e $1/update-done ] ; then
   rm $1/update-done
   exit 0
fi

touch $1/update-done

cd $1/foldername && ./fullup.sh -f -s
```

3. Insert the USB stick to the device and the update script will start automatically, without requiring any further user interaction to complete.

8. CCpilot X900

This chapter covers features and behavior unique to the CCpilot X900.

8.1. Status indication (LED)

This section describes the basic default status indication behavior. Note that the status LED behavior can be disabled totally and/or configured by user software through the CCAux API.

8.1.1. Startup sequence

During startup, the status LED indications are as follows:

- Flashing yellow at 1 Hz: pre-heating is activated and in effect, start-up is delayed while the device is heated.
- Flashing yellow at 2 Hz: device start-up phase begins.
- Operating system is then started, and specific service software begins to execute.
- Static green: device is operational.

8.1.2. Suspend

When the device is in suspend mode, the status LED indications are as follows:

- Flashing yellow at 0.2 Hz: system is suspended.
- Flashing yellow at 2 Hz: system is resuming from suspend.
- Static green: device is operational.

8.1.3. Shutdown sequence

Once shutdown (power button is pressed, or ignition signal released) is initiated:

- Static yellow: device is shutting down, rebooting or entering suspend mode.
- LED off: device is off.

9. CCpilot V700

This chapter covers features and behavior unique to the CCpilot V700.

9.1. Status indication (LED and buzzer)

This section describes the basic default status indication behavior. Note that the status indication behavior can be disabled totally and/or configured by user software through the CCAux API.

9.1.1. Startup sequence - main system

During startup to the main system, the status LED and buzzer indications are as follows:

- Flashing yellow at 1 Hz: pre-heating is activated and in effect, start-up is delayed while the device is heated.
- Status LED is blinking yellow at 2 Hz and buzzer makes short beep: Device start-up phase begins.
- Operating system is then started, and specific service software begins to execute.
- Status LED is constant green: Device is operational.

9.1.2. Startup sequence - rescue system

During startup to the rescue system, the status LED and buzzer indications are as follows:

- Status LED is constant orange and buzzer makes short beep: Device start-up phase begins.
- Operating system is then started, and specific service software begins to execute.
- Status LED is blinking green at 2 Hz: Device is operational.

9.1.3. Suspend

When the device is in suspend mode, the status LED indications are as follows:

- Flashing yellow at 0.2 Hz: system is suspended.
- Flashing yellow at 2 Hz: system is resuming from suspend.
- Static green: device is operational.

9.1.4. Shutdown sequence

Once ignition signal has been released and shutdown is initiated:

- Status LED is constant yellow, and buzzer makes short beep: shutdown sequence has started.

Status LED is off: Device is off

10. CCpilot V1000/V1200/V510/V710/V705 and Yukon development board

This chapter covers features and behavior unique to the CCpilot V1000/V1200/V510/V710/V705 and the Yukon development board.

10.1. Status indication (LED and buzzer)

This section describes the basic default status indication behavior. Note that the status indication behavior can be disabled totally and/or configured by user software through the CCAux API.

10.1.1. Startup sequence - main system

During startup to the main system, the status LED and buzzer indications are as follows:

- Flashing yellow at 1 Hz: pre-heating is activated and in effect, start-up is delayed while the device is heated.
- Status LED is blinking yellow at 2 Hz and buzzer makes short beep: Device start-up phase begins.
- Operating system is then started, and specific service software begins to execute.
- Status LED is constant green: Device is operational.

10.1.2. Startup sequence - rescue system

During startup to the rescue system, the status LED and buzzer indications are as follows:

- Status LED is constant orange and buzzer makes short beep: Device start-up phase begins.
- Operating system is then started, and specific service software begins to execute.
- Status LED is blinking green at 2 Hz: Device is operational.

10.1.3. Suspend

When the device is in suspend mode, the status LED indications are as follows:

- Flashing yellow at 0.2 Hz: system is suspended.
- Flashing yellow at 2 Hz: system is resuming from suspend.
- Static green: device is operational.

10.1.4. Shutdown sequence

Once ignition signal has been released and shutdown is initiated:

- Status LED is constant yellow, and buzzer makes short beep: shutdown sequence has started.

Status LED is off: Device is off

11. Buttons for CCpilot V510/V710 and Yukon development board

Depending on your device there are 8 (V510) or 10 (V710) configurable buttons. For the Yukon development board 16 buttons are available. The configuration of each button can be done using CCAux API or CCSettingsConsole. All button functions are controlled using the button module in CCAux API or the --button category in CCSettingsConsole.

The following functions are available:

Function	Description	Comment	CCAux API function
Standard keyboard input	Button events can be read from the device node /dev/input/event2. Buttons will generate events corresponding to F1-F16 keys. Raw data from the buttons can also be read using CCAux API or CCSettingsConsole	This functionality is always available.	Button_getButtonDownStatus()
Start up	Button press will start the unit from off state.	Only short press can be used.	Button_setStartupButtons()
Shut down	Button press will shut down the unit.	Short press or long press can be used.	Button_setShutdownButtons()
Suspend	Button press will make unit go to suspend.	Short press or long press can be used.	Button_setSuspendButtons()
Wake up	Button press will wake up the unit from suspend.	Only short press can be used.	Button_setWakeupButtons()

Note that one button can have several functions at the same time. For instance, both start up and shut down. Also note there must always be at least one way to start unit from off. If start up using ignition and pre-ignition (where applicable) is disabled, start up from at least one button must be enabled. For shut down and suspend it is possible to demand a long button press to trigger the function. The time required for a long-press is adjustable using the CCAux API or CCSettingsConsole.

Each button has a backlight LED. The backlight LEDs are programmable using CCAux API or CCSettingsConsole. It is also possible to configure a LED pattern that automatically will be activated during boot up of the unit. If such a pattern is activated, it is up to the user to set the LED status to desired values after boot is finished using a start script or user application.

The following functions are available:

Function	Description	Comment	CCAux API function
Set button backlight pattern	Set which buttons will have backlight on and also the brightness which is common for all button backlights. Also possible to set on and off time to get a blinking pattern.	Use this call in application or script to change to desired pattern any time after unit has booted	Button_setBacklightPattern()
Set button backlight pattern during boot		Will be set automatically when unit boots	Button_setBootBacklightPattern()
Get button backlight pattern	Get current setting		Button_getBacklightPattern()
Get button backlight pattern during boot	Get current setting		Button_getBootBacklightPattern()

Technical support

Additional sources of information are available on the CrossControl support site:
https://crosscontrol.com/support/
You will need to register to the site in order to be able to access all available information

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

- The part number and serial number of the device, which you can find on the brand label.
- Date of purchase, which can be found on the invoice.
- The conditions and circumstances under which the problem arises.
- Status indicator patterns (i.e., LED blink pattern).
- Prepare a system report on the device, using CCSettingsConsole (if possible).
- Detailed description of all external equipment connected to the unit (when relevant to the problem).

Trademarks and terms of use

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

CCpilot is a trademark which is the property of CrossControl.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

CC Linux is an official Linux distribution pursuant to the terms of the Linux Sublicense Agreement

Microsoft® and Windows® are registered trademarks which belong to Microsoft Corporation in the USA and/or other countries.

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Qt is a registered trademark of The Qt Company Ltd. and its subsidiaries.

CrossControl is not responsible for editing errors, technical errors or for material which has been omitted in this document. CrossControl 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.

For CrossControl licensed software, CrossControl grants you a license, to under CrossControl’s intellectual property rights, to use, reproduce, distribute, market and sell the software, only as a part of or integrated within, the devices for which this documentation concerns. Any other usage, such as, but not limited to, reproduction, distribution, marketing, sales and reverse engineering of this documentation, licensed software source code or any other affiliated material may not be performed without the written consent of CrossControl.

CrossControl respects the intellectual property of others, and we ask our users to do the same. Where software based on CrossControl 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 device, refer to the copyright notices documentation.