Product revision: 1.0
2018-03-19

CCpilot VC

Software guide

crosscontrol

www.crosscontrol.com

Contents

1. Introduction

This software guide contains information on how to start using the *CCpilot VC* products and how to access service functionality.

These devices are available with Linux as operating system. The reader should be familiar with the respective operating system and computer usage to fully benefit from this material.

1.1. Conventions and defines

1.2. All CCpilot VC devices are in most cases identical in functionality and usage.

The observe symbol is used to highligt information in this document, such as differences between 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 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 the device and the available APIs see the following references.

[1] CCpilot VC – Programmers Guide

[2] SocketCAN, http://developer.berlios.de/projects/socketcan/

[3] CC AUX API specification

[4] CCpilot VC – Technical Manual

2. Basic operation

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

2.1. Login and passwords

By default auto login is enabled and no passwords are required to start using the device with the default graphic application. For other accesses, login is required.

The root/administrators passwords are as follows:

- Under Linux there are two users per default. The normal user is **ccs** and the password is **default**. This user has administrator rights by using the sudo approach. The **root** user also exists with password **suseroot** and has full administrator rights.
- Custom users can be added with the tools from respective operating system. The adduser program is configured to enable this type of operations in this Linux system.

2.2. Using the touch screen

Navigate the startup screen 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.

Touch screen functionality is only available in product versions equipped with touch screen display.

2.3. Calibrating the touch screen

To calibrate the touch screen, start Touch Calibrator from startup screen application. Then follow the step by step sequence to calibrate the display. You can also start these programs from the command line, if required. The startup screen application is further described in 2.9 Default start up application.

It is also possible to launch the calibration application explicitly from any custom application by calling the *ts_calibrate* application.

Touch screen functionality is only available in product versions equipped with touch screen display.

2.4. Keyboard

There is no virtual keyboard available, so a user that requires keyboard access needs to connect a USB keyboard.

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 the 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.

Device start-up behaviour

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, or SS. The SS then supplies the main processor and peripherals with power; from there the device execution begins, starting with the boot loader.

2.6. 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 eventually load the operating system kernel (Linux kernel) into RAM memory. Once finished, the operating system execution takes over. This process takes very little time, in the order of hundreds of milliseconds only.

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 upon need.

2.7. 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 i.MX53 ARM 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 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.

2.7.1. Boot count

If the main system fails to boot successfully for 5 times in a row, the device can automatically boot to rescue system instead of normal system. The reason for failed main system boot can then be examined and corrected in the backup system.

As a control point, the boot is deemed successful if the system start-up process gets to level 60. This means that the script file */etc/rc3.d/S60system-up* executes successfully.

It is possible to start user applications before or after this control point. See chapter Start-up scripts (6.1.4) for a description of start-up scripts.

2.8. Status indication (Button backlight or Buzzer status)

This section describes the basic default status indication behavior. Note that the status indication behavior can be disabled totally and/or overridden by user software through the CC AUX API. CCpilot VC uses the button backlight to indicate various events as described below (only on CCpilot VC models with buttons).

2.8.1. Start-up sequence

During startup, the backlight indications are as follows:

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

2.8.2. Shutdown sequence

Once shutdown (power button is pressed between 4 to 8 seconds) is initiated:

1. Immediately after button press: Backlight intensity is changed to indicate that the power button is pressed down. After 4 to 8 seconds: Backlight blinks at 10Hz to indicate that shutdown sequence has been started. When the power button is released, backlight is constantly on.
2. Backlight is off: Device is off

2.8.3. Suspend/resume sequence

Once suspend (power button is pressed between 0 to 4 seconds) is initiated:

1. Immediately after button press: Backlight intensity is changed to indicate that the power button is pressed down. When the power button is released, backlight is constantly on.
2. Backlight is blinking shortly every 5 seconds: System is suspended.
3. Resume from suspended state by pressing a configured button or the external on/off signal. Backlight is constantly on: System is operational.

2.8.4. Error indication

There are two main types of error indications available in CCpilot VC:

- Three buzzer beeps, no repeat – This indicates during startup an internal error that is recoverable, i.e. the system can continue its execution and normal operation. The error code can be retrieved trough the CC AUX API. The backlight will follow its normal status indication scheme.
- A sequence of beeps is repeated constantly at any time after the system is powered on – The device becomes non-operational, and seems to be dead, despite powered on. The number of beeps indicates the issue. Record the number of buzzer beeps, as this can be useful in an eventual support contact. It may help to just power off and on the device but if the error indication beeps remains it is likely an internal hardware error or problem with the power supply.

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

2.9. Default start up applications

By default, 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 6.6.

It's also possible to connect to the device via a VNC Viewer application (the VNC server must first be enabled, it is not by default). By default, it's only the CCsettings application that's accessible via the VNC protocol, but it can be replaced with any desired Qt application. Further information about that is found in section 6.11.

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 *CCpilot VC – Programmers guide* documentation for general information regarding software development using the devices interfaces.

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

3.1. File storage

The device uses a NAND flash based storage, which is partitioned into protected operating system parts and writable user parts. The file system abstracting the NAND layer is called UBI-FS.

The NAND flash is industrial grade classified and has both static and dynamic wear levelling to prevent a premature aging and to ensure the longest lifetime. Still, NAND flash has 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 shouldn't cache files in the NAND flash file system, since in case of a sudden power loss, the NAND flash's writable partition is made write-protected to protect the files from being corrupted due to the power loss. 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 NAND flash.

Additional details about NAND flash usage recommendations can be found in *CCpilot VC – Programmers guide*.

3.1.1. File system layout

In Linux, the NAND flash is partitioned into two root file system, which are write protected, and one user file system area, which is write enabled by default. The latter area is the preferred location for user software installations. See chapter 6.1 Installing new drivers, applications and system packages.

The two initial partitions hold each operating system mode's root file systems, i.e. first the rescue system root file system and second the main system root file system. The root file system is located in the SD flash card. It is of type UBI-FS and it is mounted as read-only for security reasons. It is mounted by the kernel boot-up sequence.

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 file system is unsupported, device is just not mounted.

Mount point	Mount status	Media
/	Read only	NAND flash, second file system partition if in normal mode
/usr/local or /opt	Read-write	NAND flash, third file system partition
/media/usb	Read-write	USB memory, if available
/media/usb2	Read-write	Second USB memory, if available
/tmp	Read-write	RAM memory, for storing temporary files
/var	Read-write	RAM memory, for storing logs etc. during runtime

The */usr/local/*, or */opt*, partition of the NAND flash contains some default configuration files, but most of its space is reserved for application programs. It is up to the user to decide which applications to run.

3.2. CAN

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

The CAN interfaces can also be accessed with the Linux operating system standard API *SocketCAN*. More information can be found in *CCpilot VC – Programmers Guide*.

3.3. Ethernet

Up to two Ethernet ports are available on the device, thus enabling simultaneously connection with two physical network infrastructures. The 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, i.e. by using the network interfaces file as described in 6.3.

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

3.4. USB

Via the USB port, a multitude of peripherals can be connected to the device. For some peripherals, drivers compatible with the operating system must be installed in order for the peripherals to

function. For such installations, please contact CrossControl for support in adding a suitable driver, or follow the instruction from the device.

3.5. Video in

The video in signal can be viewed by the *CCvideo* or application, further described in chapter 5 CCvideo. The video in signal can also be accessed and controlled using the CC AUX API.

Video input channel 1 can be displayed without any significant CPU performance penalty, in one video window instance.

3.6. Configurable Input

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

For additional technical details, please see the *CCpilot VC – Technical Manual* or the *CC AUX API reference documentation*.

3.7. PWM Output

The user-settable PWM output signals are available for software developers using the CC AUX API. The output status can be viewed or set within CCsettings for test purposes.

For additional technical details, please see the *CCpilot VC – Technical Manual* or the *CC AUX API reference documentation*.

3.8. Backlight

The device has an adjustable backlight intensity level. In addition to the backlight control buttons on the device front, the backlight functionality can also be controlled from *CCsettings* and via software using the CC AUX API. The most recent setting is always used, it is saved between restarts.

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 the device, but it is not enabled by default. It can be set up in *CCsettings* or through the CC AUX API.

3.9. Ambient light sensor

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

3.10. Buzzer

The device is equipped with a buzzer that can play tones in various frequency and intensity levels. The buzzer is accessed through the CC AUX API. It can also be accessed for diagnostic through *CCsettings*.

3.11. Hardware watchdog functionality

There is a hardware watchdog available in the device making it possible for the hardware to monitor an application so it does not stall the system. The watchdog can be controlled by the user. The watchdog timeout is 16 seconds by default, and can be changed using configuration file

/usr/local/etc/watchdog_timeout.conf. The file should contain only a single numeric value on first line – that value is the requested timeout in seconds.

Valid values are from 0 to 127. Using a value zero (0) disables the watchdog.

By default the watchdog is taken care of by specific watchdog kicker software called *wdt*. If user software wants to take over the watchdog handling, then *wdt* task should be killed first (for example **killall wdt**). If the user software does not want to handle the watchdog anymore, then the *wdt* software should be restarted. *wdt* is started at start-up every time if the watchdog timeout is something other than zero.

3.12. Temperature sensors

There are several temperature sensors placed internally in the device. It is possible for an application to retrieve temperature information from the temperatures sensors through the CC AUX API.

3.13. Buttons

CCpilot VC has eight configurable buttons that can be configured in the CC AUX API and in the Buttons dialog in CCsettings (see section 4.16). The buttons are wired to both main processor and SS except in the case of *Only MP action* which effectively makes a button ignored for SS.

Button events can be read from device node /dev/input/event0. Code examples are available in file *buttons_example.cpp* under the API examples directory.

In addition to the examples in the previous file, you can also use normal Qt methods *keyPressEvent* and *keyReleaseEvent* since the CCpilot VC physical buttons are mapped to function buttons F1 to F8 and they are bound to Qt keys from Qt::Key_F1 to Qt::Key_F8, respectively. See the alignment of the buttons in the picture below.

3.14. System time

The CCpilot VC operating system keeps track of the system time using a battery powered real-time clock (RTC). The system time can be updated using the **date** command: **/bin/date -s yyyyMMddhhmm.ss** (other syntax options exist).

The operating system automatically updates its system time from the RTC at start-up and stores the system time to the RTC at shutdown. The **hwclock** command may be used to force a read or write to the RTC but in most cases this is not recommended.

4. CCsettings

The images here are for illustration purposes and may not represent the real device user interface

CCsettings is used for viewing and adjusting 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 CC AUX 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.

4.1. Main menu

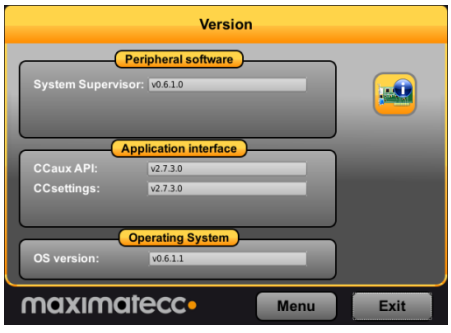
The main screen of *CCsettings* show the available settings pages.

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.

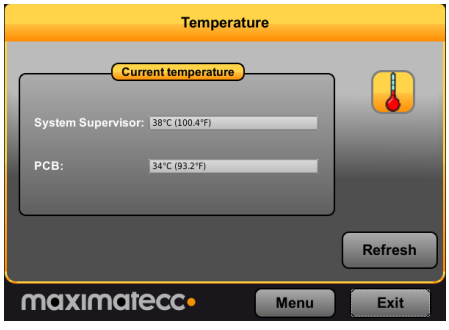
4.2. Version

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

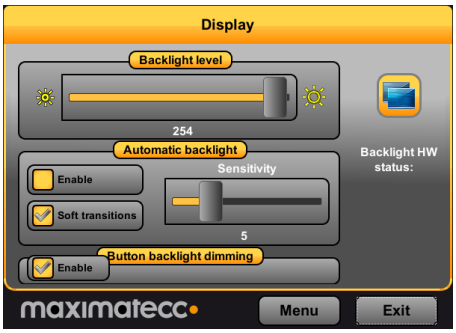


4.3. Temperature

The device internal temperature sensors can be viewed from the temperature page.



4.4. Display



Use the display page to alter the 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 the device light sensor. *Soft transitions* enable smoother adjustment of the backlight.

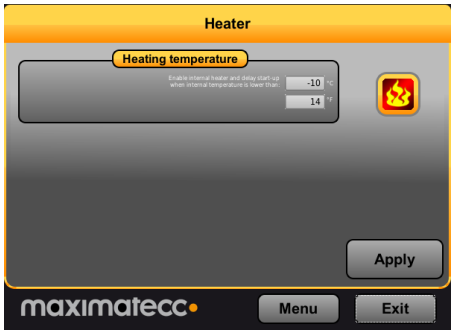
The *Sensitivity* slider adjusts the level and phase of backlight adjustment depending on the surrounding light.

The button backlights can be automatically dimmed if dimming is enabled. Note that this may not work if a blink sequence is set up by user applications.

4.5. Power

This display is currently disabled in CCpilot VC.

4.6. Heater

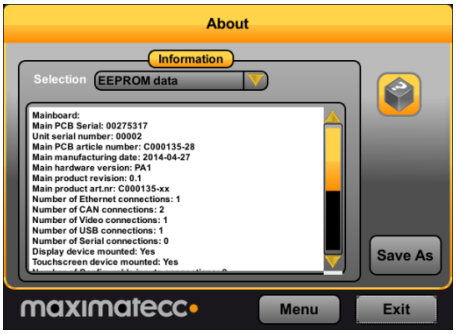


The heater page is used to adjust the starting temperature of the internal heater.

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

The heater page is disabled if the CCpilot VC model does not have the heater functionality.

4.7. About



The about page displays device specific 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.

4.8. Button backlight

The button backlight display is used to view the indication types and to adjust the button backlight behavior during startup.

Enable or disable the button backlight indication during startup of the device using the *Enable* checkbox.

4.9. Buzzer

The Buzzer page is used to manipulate the audio output parameters of the Buzzer.

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

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

4.10. CAN

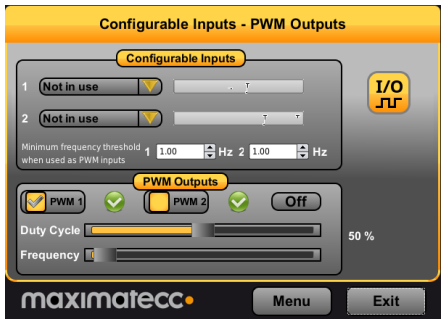
CAN is normally used and configured directly from the controlling software but from *CCsettings* it is possible to adjust the default *Baud rate*.

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

4.11. Configurable Input and PWM Output

The configurable input and PWM Output tab of the VC device is opened from I/O button in the main menu.

For each configurable input, it is possible to choose the input type and frequency threshold. The frequency threshold is used if corresponding input type is Freq floating, pull up or pull down. Configurable inputs are not in use by default.



To adjust Duty Cycle or Frequency of a PWM output, choose PWM1 or PWM2 in the radio button and adjust its parameters from the sliders below.

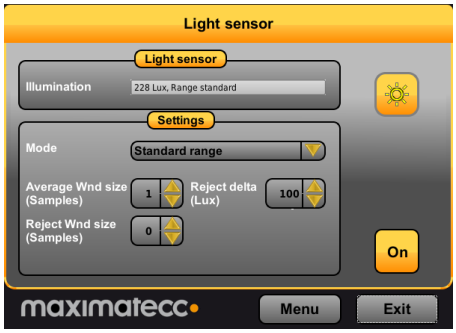
4.12. Touch screen

This display is currently disabled in CCpilot VC.

4.13. Light sensor

The light sensor page is used for manipulating Light sensor parameters.

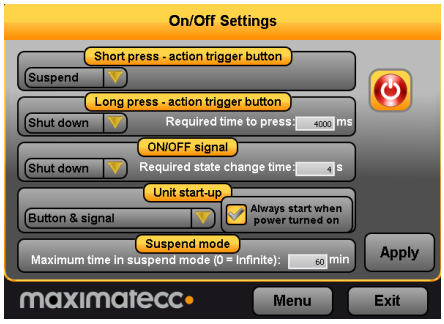
The average calculation works by calculating the average from a number of consecutive samples, the average window size. The reject window is used to discard sudden changes or single extreme values of the measurement.



If the difference of the maximum value and the minimum value in the number of samples in the reject delta window is larger than the reject delta, those samples are discarded.

The ADC of the light sensor has two operating modes or ranges: standard and extended. The extended range can capture a 4 times higher value before being maxed out, but at the cost of lower precision. The extended range allows the device to operate at higher light levels, extending the overall dynamic range by a factor of four. Choose the mode in the Mode menu. Standard range is the default mode.

4.14. On/Off



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

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

Short press – action trigger button - adjust the unit behavior when making a quick press and release of a button configured as an action trigger.

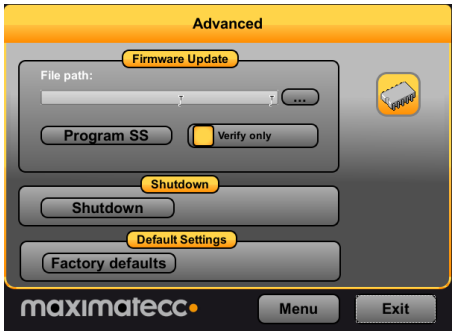
Long press – action trigger button - adjust the unit behavior when pressing and holding a button configured as an action trigger.

ON/OFF signal - adjust the behavior of the On/Off signal through the power connector. The time setting adjust how long the signal must stay low before the 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. Setting the time value to *0* (zero) disables the shut-down feature; the device will stay in suspended mode indefinitely, until started or until power is removed.

4.15. Advanced



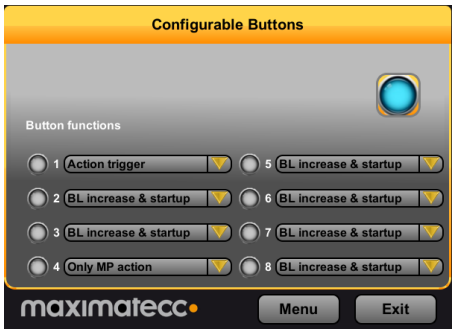
The Advanced page is used for loading new SS firmware into the device and to restore settings to factory defaults.

To update the SS firmware, make sure the firmware file is located on the device. Browse for the file, then press *Program SS* button and wait for the firmware update to complete. If only firmware verification is wanted, choose the file on the device and press Program SS. If the file matches the current SS firmware, it is reported. The firmware is not written during verify.

For detailed information about how to upgrade the firmware, please see additional documentation specifically for firmware update.

4.16. Buttons

The Buttons display allows manipulation of the usage of CCpilot VC configurable buttons. Each button can be configured to various actions as described in the table below. The buttons are connected to main processor (MP) and SS simultaneously and this display is used to choose whether a button is used in main processor or System Software.



Action	Description
Only MP action	Button is not handled in SS. The button can be used in user software via the input event API. The buttons can always be used for MP action, even when configured for any of the other functions below.
Startup trigger	Button is used by SS to trigger startup action; powers up the system or wakes up from suspend state.
Action trigger	Button is used as an ON/OFF action trigger. See ON/OFF display menu how this button is handled in SS. (Short and long press behavior can be configured)
Startup & Action trigger	Button is used as both Startup and Action trigger.
BL decrease	Button decreases backlight intensity.
BL decrease & startup	Button decreases backlight intensity and acts as a startup trigger.
BL increase	Button increases backlight intensity.
BL increase & startup	Button increases backlight intensity and acts as a startup trigger.

5. CCvideo

CCvideo is a pre-loaded application for viewing the device video in signals. The CCvideo application can show video from the input channel using the CC AUX API.

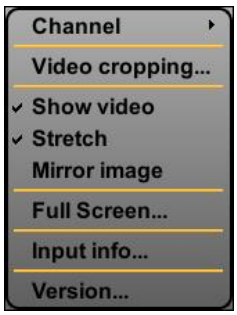
All functionality and settings accessed using *CCvideo* is also available through the CC AUX API to enable integration of settings and functionality within any additional user application. The API is further described in [1].

5.1. CCvideo main window

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

5.2. Selection menu

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

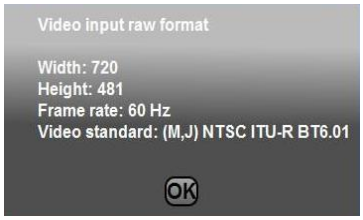


- *Channel* - Select the video in signal to display. Only one channel available.
- *Video cropping* - Opens the Video cropping dialog.
- *Show video* - Enables/disables the video in feed to be displayed.
- *Stretch* - Enables/disables stretching of the video image to fit the size of the *CCvideo* application window.
- *Mirror image* - Enables/disables video image mirroring.
- *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*.

5.3. Input info

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

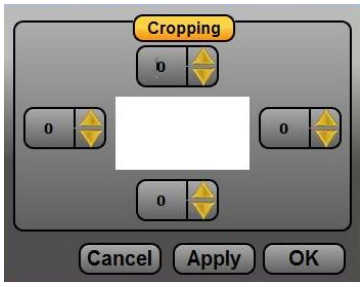
- *Width* - Native with of the signal in pixels.
- *Height* - Native height of the signal in pixels.
- *Frame rate* - Current frame rate of the video signal.
- *Video standard* – The detected video standard of the video signal.



5.4. 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.



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

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*.

6.1.1. 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 /
```

6.1.2. 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.

6.1.3. 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.

6.1.4. 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 6 is dedicated for shutdown. When the device is shutting down the scripts from */etc/rc6.d/* and */opt/etc/rc6.d/* are executed to perform last clean-up actions. Required naming and execution order rules comply with start-up level 3.

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.

6.2. Text editor

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

6.3. 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.

6.3.1. 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 same static address as the graphical method above.

```
auto lo
iface lo inet loopback

auto eth0
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
```

6.4. Remote access

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

6.4.1. SSH

To connect to the device, 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.

6.4.2. 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 device to a host, use the following command:

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

To copy a file from the 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.

6.4.3. 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 device), execute the command below and enter an empty passphrase when prompted.

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

Copy (or append) the created *xa_rsa.pub* file into target device as a */etc/ssh/authorized_keys* -file. Note, this method needs to be done with root file system mounted as read-write.

Move the *xa_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/xa_rsa ccs@X.X.X.X
```

6.4.4. 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.

6.4.5. VNC

It is possible to enable VNC server in the GUI for remote control. Currently the only way to enable it is to edit GUI startup script file /opt/etc/init.d/StartupGui directly:

```
~$ sudo nano /usr/local/etc/init.d/StartupGui
```

Look for the setting USE_VNC=0 in the file and change it:

```
USE_VNC=1
```

Press Ctrl-X, y and ENTER to save your changes. After that GUI must be restarted either in the command line:

```
~$ sudo /opt/etc/init.d/StartupGui stop
~$ sudo /opt/etc/init.d/StartupGui start
```

Or by rebooting the device. VNC will be enabled now after every boot. To disable it, edit /opt/etc/init.d/StartupGui script again and put the line USE_VNC=0 back.

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

6.5. Frame buffer graphics

The graphics framework uses the Linux frame buffer for graphics operations. It's fast and efficient, and has vast standard support in the Linux user space world. For instance, Qt has a plugin that enables the Qt libraries to be built with direct frame buffer drawing possibilities. For Qt applications the impact for that means that it simply needs to be started with a specific flag, -qws, and built with the correct development libraries.

Other application software that uses the frame buffer directly can also be used. However, there is no default windowing system on the frame buffer, so careful consideration there needs to be taken when using this kind of applications.

Multiple instances of Qt applications work fine, and can be seen as a windowing system as well, since that's a characteristic of the Qt QWS system. However, only one Qt application should act as the GUI server, i.e. use the –qws flag when starting. Make sure that the graphical applications adhere to that requirement for proper usage of Qt applications on the frame buffer.

6.6. Default startup application

As mentioned in 2.9 Default start up application, Some CCpilot VC versions have a GUI application that is started by default. The default application looks like this:

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 6 (during shutdown/reboot).

Note: This script is part of the rc script solution mention in section 6.1.4. Make sure the correct script behavior is met; otherwise it can break the VC device startup procedure.

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

```
#!/bin/sh
. /etc/init.d/cross-common

#
# END USER: Replace this with your own application. Take care using correct Qt
flags.
#
APPLICATION_PATH="/usr/bin"
APPLICATION="StartupLauncherGui"
```

```
#
# END USER: To enable VNC, change following value from 0 to 1
#
# There will be performance penalty when VNC is enabled even if VNC
# connection is not open.
#
USE_VNC=1
####

RESOLUTION=$(get_display_resolution)
DEFAULT_RESOLUTION="800x480"

case "$1" in
 start)
     if [ -z $RESOLUTION ] ; then
             RESOLUTION=$DEFAULT_RESOLUTION
     fi
     if check_feature DISPLAY || check_feature EXTDISPLAY; then
             if [ $USE_VNC -ne 0 ]; then
                ( QWS_KEYBOARD="LinuxInput:/dev/kb0 LinuxInput:/dev/kb1
LinuxInput:/dev/kb2" \
                        QWS_MOUSE_PROTO="LinuxInput:/dev/touch0
MouseMan:/dev/mouse0" \
                        $APPLICATION_PATH/$APPLICATION -qws \
                        -display "Multi: transformed:
VNC::size=${RESOLUTION}:depth=32:0" &> /dev/null & )
             else
                ( QWS_KEYBOARD="LinuxInput:/dev/kb0 LinuxInput:/dev/kb1
LinuxInput:/dev/kb2" \
                        QWS_MOUSE_PROTO="LinuxInput:/dev/touch0
MouseMan:/dev/mouse0" \
                        $APPLICATION_PATH/$APPLICATION -qws -display
transformed &> /dev/null & )
             fi
           else
             # Display less unit supports only VNC
             if [ $USE_VNC -ne 0 ]; then
                ( QWS_KEYBOARD="LinuxInput:/dev/kb0 LinuxInput:/dev/kb1
LinuxInput:/dev/kb2" \
                        QWS_MOUSE_PROTO="LinuxInput:/dev/touch0
MouseMan:/dev/mouse0" \
                        $APPLICATION_PATH/$APPLICATION -qws \
                        -display "VNC::size=${RESOLUTION}:depth=32:0" &>
/dev/null & )
             fi
           fi
           ;;
 stop)
           killall $APPLICATION
           ;;
 *)
```

```
            echo $"Usage: $0 {start|stop}"
            exit 1
esac

exit $?
```

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 6.1.4 Start-up scripts.

6.7. Late load modules

To improve the startup time of the device, a few device drivers are being loaded after the application startup process has begun, which means that the graphical application is started early. This is because some hardware specific modules will take up a large proportion of the time it takes for the device to boot.

This can be changed, if critical functions in drivers are needed earlier, at the expense of slower startup of applications. The user can edit a configuration file to change the behavior. The file is */opt/etc/late-loaded-modules*, and changes are done by commenting out or removing corresponding modules from the file. By default it contains modules for devices like video, audio, and USB host controller.

6.8. Serial Number Broadcast configuration

The 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 the 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*. 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, if unset '0' is used. String value
#UNITTYPE=VC
```

6.9. 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.

6.9.1. 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 start up.

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 (cc-auto.sh) 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
```

6.10. 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 VC does not support audio. Each supported kind of video can be played with the command:

```
# gplay <videofile>
```

This command launches a player with interactive ASCII interface, but 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.

6.11. VNC

The VC device also has support for a partial VNC server solution so that a GUI application can be accessed remotely when a graphical remote environment is desired. The basics for the VNC functionality comes from support in the Qt libraries, and thus it is required that the VNC Server is actually a Qt application designed for that use case.

To access the device via VNC, you will need to use a VNC viewer application and enter the IP address of the VC device. By default, there is no application made available for access through VNC, but it can be enabled to use a default one, or custom made by the end customer.

On the device, the Qt VNC application requires the QWS flag as well as the VNC flag to operate properly as a VNC server. In the default version of the StartupGui startup script (below), a separate Qt VNC session can be started by setting a specific USE_VNC flag, see the script for details.

Specifically starting a Qt VNC application is done as follows:

```
$/path/to/myapp -qws –display VNC::size=800x480:depth=32:0" &
```

Any application that should be run as the VNC server can be started in the same way in a startup script, but pay attention to the design guides for scripting, not locking the startup sequence.

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

The VC 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 VC 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 CrossControl 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.

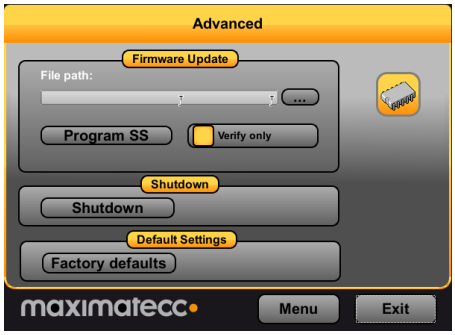
The restoration is done via the command

```
# 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.

7. Software update and recovery

7.1. 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.

7.2. Updating firmware components

The Advanced page is also used for loading new SS firmware into the device. To update the SS firmware, make sure the firmware file is located on the device. Browse for the file, then press *Program SS* button and wait for the firmware update to complete. If only firmware verification is wanted, choose the file on the device and press Verify SS (the button text is updated). If the file matches the current SS firmware, it is reported. The firmware is not written during verify. After each program or verify, 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 *CCsettings* program. It is also possible to use the command line based application *ccsettingsconsole*, or the API directly from your own program using the CC AUX API functions.

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. 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.

For specific details on how to perform firmware component updates, please see additional documentation for firmware updates.

7.3. Updating operating system

The Linux system on a 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 be then shipped to factory for repair, or have internal parts replaced through service interfaces.

7.4. 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 five main parts in the NAND flash: main kernel, backup system kernel, backup system root file system, main root file system and user defined area. Additionally, there is boot loader software in the first part of the NAND flash. Normally, software updates concern only main kernel and root file system, occasionally also the other parts.

7.5. Updating from console

Warning: An update erases and replaces the old root file system completely! This should not affect contents of */usr/local/*, however backup of important files is advised.

This method is possible when the device has working backup system.

7.5.1. Preparing update within Linux

If main system fails to boot 5 times in a row, backup system is automatically entered. From a working main system, entering the backup system is done using following reboot command:

```
~# reboot-rescue.sh
```

Once the module has entered backup system, copy in the update images to unit */usr/local/* folder. **scp**, USB-memory or NFS mount can all be used for copying. Access the Linux console, either over SSH connection from another host, or using a serial console terminal access to Linux.

Warning: Avoid SSH method, if your Ethernet network is susceptible for connection breaks, as the image write can get interrupted.

7.5.2. Updating device within Linux

Prerequisite: New file images shall be located at the */usr/local* folder on the module, or in other appropriate directory. The files that can be present are normally a subset of these files below, depending on which parts require the update:

- *ccpilot-vc_kernel.bin*
- *ccpilot-vc_rootfs.bin*
- *ccpilot-vc_u-boot.bin*
- *ccpilot-vc_mbr.bin*
- *ccpilot-vc.md5sum*
- *fullup.sh*

Update tool command **fullup.sh** should be copied to target explicitly as well.

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

```
# ./fullup.sh
```

Example output:

```
==== CrossControl: Device Update ===== A8.2
Now running on   : BACKUP side
 Requested action: NORMAL system update
    Checking MD5: cross-vc_kernel.bin: OK
    Checking MD5: cross-vc_rootfs.bin: OK

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

User verification: ARE YOU SURE ?

  - If yes, press Enter to continue.
```

```
  - IF NOT, press CTRL+C now to cancel update.
 This is you last change to do cancel!

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

The process is pausing here and waiting for Enter or cancellation.

```
      Now doing: Requested reprogramming:

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

Erasing old kernel..
Erasing 128 Kibyte @ 3e0000 -- 96 % complete.
Copying new kernel image..
Writing data to block 0
Writing data to block 20000
--clip--
Writing data to block 140000
Erasing old root..
Erasing 128 Kibyte @ 1da0000 -- 37 % complete.
Skipping bad block at 0x01dc0000
Erasing 128 Kibyte @ 4fe0000 -- 99 % complete.
Copying new root-fs image..
Writing data to block 0
Writing data to block 20000
--clip--
Writing data to block fe0000
Completed. Rebooting..
```

7.6. Update automation

This example shows how to automate remote update so that user interaction is not required. Similar automation can also be saved to USB memory; so that inserting the USB memory into a working device will execute an update without interaction. This method was introduced in 6.9 USB memory installer.

7.6.1. Updating automatically

Power-up the device, make sure the login less SSH access is set up as according to 6.4.3. Copy the update image files, *fullup.sh* and *update.sh* scripts 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 xa_rsa root@X.X.X.X:/usr/local/
ssh -i xa_rsa root@X.X.X.X "/usr/sbin/reboot-rescue.sh;/sbin/reboot"
```

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

7.6.2. 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.

5. *cc-auto.sh* creates folder */usr/local/fw_update/*.
6. *cc-auto.sh* copies all necessary files to that folder, including file called *update.sh*.
7. *cc-auto.sh* runs *reboot-rescue.sh*.

   At this point the USB memory can and should be removed.
8. Device reboots to backup system.
9. Backup system executes */usr/local/fw_updata/update.sh*.
10. */usr/local/fw_updata/update.sh* does whatever it wants and then deletes the */usr/local/fw_updata/* folder to prevent these actions from executing twice.
11. The last thing the script does is to reboot the device again, into normal operating mode.

7.7. 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.

7.7.1. Preparing update within Linux

Copy the backup system update images as listed:

- ccpilot-*vc-bs_kernel.bin*
- *ccpilot-vc-bs_rootfs.bin*
- *ccpilot-vc-bs.md5sum*
- *fullup.sh*

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

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

7.7.2. 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.

7.7.2.1. 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.

Technical support

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 find on the brand label.
- Date of purchase, which is found on the invoice.
- The conditions and circumstances under which the problem arises.
- Status indicator patterns (i.e. buzzer beep pattern).
- 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).

Trademark, etc.