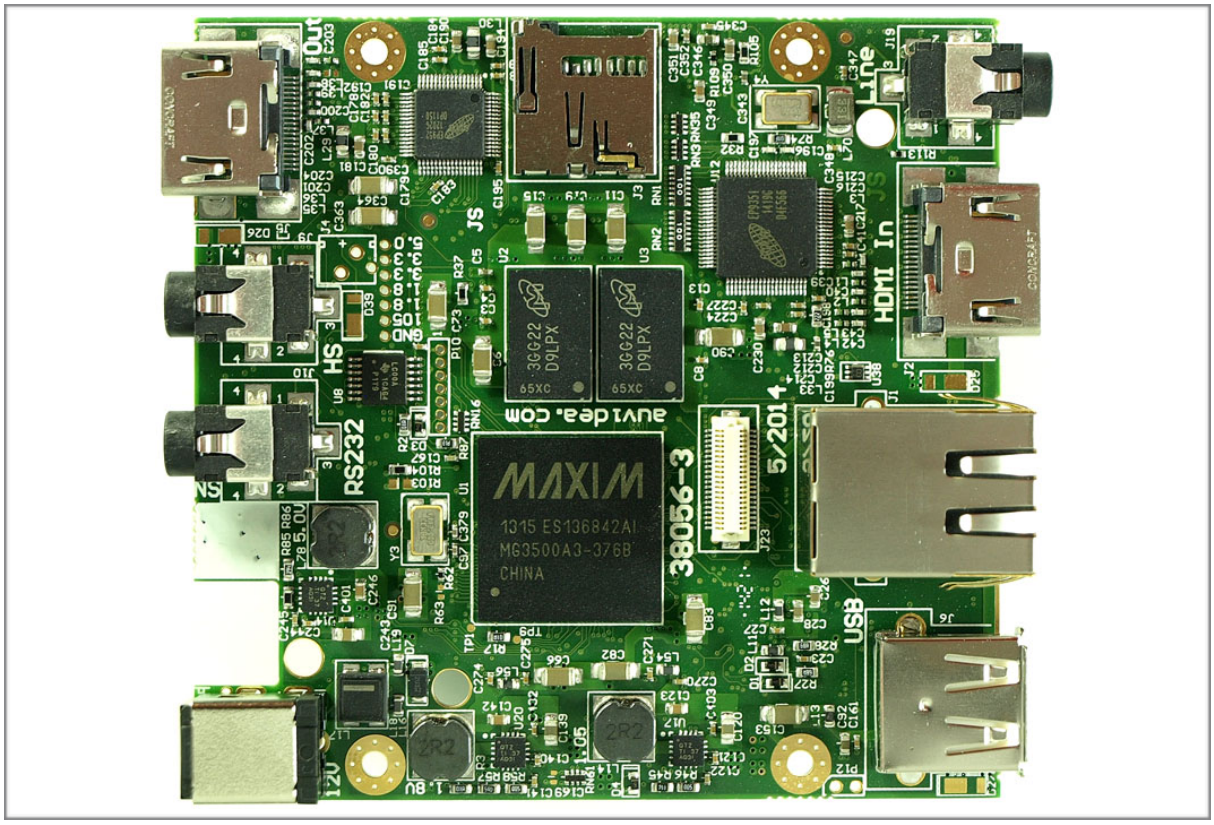
Technical Reference Manual

Version 1.8.300



E110 Encoder/Decoder

March 2015

Auvidea GmbH
Kellerberg 3
D-86920 Denklingen

Tel: +49 8243 7714 622
info@auvidea.com
www.auvidea.com

Copyright Notice

Introduction

Encoder

Compact and low cost single stream AAC audio and H.264 video encoder for recording and streaming compressed audio/video. The compressed video may be recorded on a USB memory device or internal micro SD card. Or it may be live streamed as RTSP stream.

Technical details

- 240 MHz ARM926-EJ Processor
- 128 MB DDR2 RAM (Linux and HW encoder)
- 512 MB on board flash
- USB-2 (480 Mbit/s) type A
- 10/100 Mbit Ethernet (RJ45)
- HDMI video in with embedded audio
- HDMI video out with embedded audio (E110 only)
- analog audio in and out (3.5mm jacks) (E110 only)
- RS232 serial port (console interface - 3.5mm jack)
- user configurable recording and streaming LEDs

Audio and video encoding and decoding

- H.264 codec up to level 4.1 (baseline, main and high profiles)
- video encoding up to 720p50/60, 1080i50/60 or 1080p24/25/30
- audio encoding: high-fidelity, 2-channel AAC-LC codec
- streaming: RTSP stream (128 to 20000 kbit/s)
- live change of bandwidth, GOP size and resolution while streaming
- low latency streaming from E110 encoder to E110 decoder (typically 70 to 100 ms)
- power: 7 - 17V, 3.5W (5.5 x 2.5mm jack)
- size: 77 x 79.3 mm
- weight: 47 gr

Software

- Linux based (kernel 2.6.30)
- control via integrated web GUI interface
- HTTP request based API (low latency: 100ms typically)
- RTSP video streaming and decoding
- full access to system, video and encoding parameters
- time stamp and text overlay
- single event timer
- programmable 256 byte EDID memory (HDMI in)

RTSP video streaming

The HDMI video source is connected to the E100/E110 HDMI video input. The encoder compresses the video with its integrated H.264 encoder (high profile = excellent video quality). Then it packages the compressed video into an RTSP stream and sends it out on its 10/100 network interface to any device requesting this video. This stream may be played with player such as VLC or Quicktime. Just enter the URL: rtsp://<ip-address>:554/<stream-name>. Also the stream may be received by another system to transcode the stream or to process it. Please see the transmuxer section for details.

Important note on HDCP copy protection

The HDMI input to this encoder must not be copy protected with HDCP. Some camcorder models and DVD players enable HDCP by default. This encoder will not be able to capture any video content, which is copy protected with HDCP. Sony camcorders are typically not supported as they output the HDMI video with HDCP protection. Panasonic camcorders are typically fully compatible, as their output is not protected with HDCP. Models tested: HC-V100, HC-V110, HC-V210, and HC-X929.

Timing analyzer

The integrated timing analyzer allows to determine the timing parameters of the HDMI video input signal precisely. This is very handy for the automatic configuration of the video processing software. Just set the „autosize“ parameter and the E110 will automatically configure itself for most HDMI devices connected.

E110 rev 2 improvements (38056-2)

- 12V power input (wide range: 7V to 17V DC)
- 32 pin FPC connector for LED and GPIO daughter board
- 2 additional 3.6mm holes (for mounting the metal case)
- hardware patch for all interlaced video modes (jumper wire is installed)

E110 rev 3 improvements (38056-3)

- JTAG port for source level debugging
- RS485 interface on FPC connector
- 2 additional 3.6mm holes (for mounting the metal case)
- optional internal power connector (to add a locking power connector)
- 40 pin connector for add-on modules such as WLAN module (for transmuxing)

Firmware Releases

Release 1.8.83 (October 2014)

- fixed subnet mask permanent save
- added safe encoder mode (does not support 1080i/p)
- delete recordings
- support for W100 add-on module
- added statistics for streaming and recording

Release 1.8.3 (August 2014)

- PWM api added
- decoder web page enhanced

Release 1.8 (July 2014)
New features:

- web based GUI to exercise the API and to control the E110
- write custom EDID and configure programmable timer
- programmable text overlay (up to 8 strings with 24 characters each)
- timestamp overlay with frame counter
- profile selection: H.264 baseline, main or high
- decoder: play RTSP stream (720p and 1080i) - AAC audio and H.264 video

Release 1.3 (April 2014)
Initial release of the E110 firmware. Key features:

- default configuration in /data/bin/config
- analog audio support
- ntp time server support (UTC time zone) - set date and time (Internet connection required)
- timer - automatically schedule recording or streaming events
- automatically start recording or streaming after power up (set timer appropriately)
- DHCP support
- at boot up config file and startup.log (with IP address) are copied to USB memory device
- www_root file management und upgrade support
- log API calls

Known issues

- do not disconnect the video source while recording or streaming
- switch from decoder mode back to encoder mode requires cold reboot

System Control

The startup script /etc/init.d/99init initializes the functions of the E110. It sets the date and time through ntp and starts the core executable /data/bin/auvidea_enc which supervises and manages all functions of the E110 including the http request based API.

auvidea_enc
Initialize all hardware devices in the E110. The first 2 options have to be executed once after power up to initialize the hardware devices in the system.

Options:
-i reset and init HDMI devices
-a init analog audio
-e start normal operation

Telnet

For security reasons Telnet is disabled and should not be activated.

SSH

By default the root level access is disabled, to protect the system. Experienced users may request the admin password for the „admin“ API call. This password allows to enable SSH access and the other functions of the admin interface. If you request the admin password for your system, you need to sign the warranty release agreement, as Auvidea can not longer guarantee the operation of the system. System operation is then the users full responsibility. Worst case, the system may become dysfunctional, the hardware may be damaged and a complete reflash may be required. At this time, only Auvidea can perform flashing of the system. Auvidea provides this as a service. Please contact us at *support@auvidea.com*. Please include a copy of the startup.log file, which the system copies to the USB devices at boot up time.

Please note, that the root level access will not allow you to port third party applications or custom software to the E110, as there is no C tool chain (a C compiler) installed. The system requires a custom cross tool chain. For customization we recommend to use the optional WLAN add-on module and install custom software on that module.

Web server

This is the main control interface with HTTP requests as described in the following sections. Also this web server may be used to serve standard html pages. They may be copied to the www_root directory with the „www“ API calls. The web server supports no server side processing such as PHP or JSP. Please contact us, if you would require more features. Optionally a regular web server (lighttpd) could be activated.

FTP

FTP is disabled by default.

Default settings

```
VERSION=19
##############################################################################
#  auvidea_enc configuration
#  version 19 (July 2014)
#  warning: do not modify (this file is maintained by auvidea_enc)
##############################################################################
# Audio settings
DIGITAL="1"
LN_VOL_LEFT="0"
LN_VOL_RIGHT="0"
LN_MUTE_LEFT="0"
LN_MUTE_RIGHT="0"
HP_VOL_LEFT="-10"
HP_VOL_RIGHT="-10"
HP_MUTE_LEFT="0"
HP_MUTE_RIGHT="0"
MICBIAS="0"
SAMPLERATE="48000"
# network settings
SUBNET_MASK="255.255.255.0"
# ntp time server
NTP="pool.ntp.org"
UTC_OFFSET="-2"
# audio and video capture
WIDTH="1280"
HEIGHT="720"
HSIZE="1280"
VSIZE="720"
START="0"
START="0"
CROP_LEFT="0"
CROP_TOP="0"
FPS="60"
AUTOSIZE="1"
SWAPUV="0"
# encoding (audio only not supported yet)
VIDEO="2000"
AUDIO="128"
PROFILE="2"
GOP="30"
FPS_DIVIDE="2"
AV="0"
STEREO="1"
PREVIEW="1"
# streaming
PROTOCOL="RTSP"
NAME="stream"
PORT="554"
# timer
START="0"
```

```
DURATION="0"
TIMERMODE="1"
# play
P_ADDRESS="192.168.0.160"
P_NAME="stream"
P_VIDEO="1"
P_FORMAT="2"
P_AV="0"
P_NTSC="0"
P_SAMPLERATE="48000"
# LED
LEDREC="2"
LEDSTREAM="2"
LEDPOWER="2"
```

This startup script initializes the functions of the E110. It sets the date and time through ntp and start the core executable /data/bin/auvidea_enc which supervises and manages all functions of the E110 including the http request based API. This script file is copied to the USB memory device, if one is inserted at bootup time.

Auto start
The timer is disabled by setting both TIMERMODE to „0“. To start streaming immediately after power up, set START=„0“, „DURATION“=„0“ and TIMERMODE=„1“.

„CROP_LEFT“ and „CROP_TOP“ are the cropping parameters.

Encoder Settings

Default settings

The default settings allow to stream a 16:9 video source such as the 1280x720p60 video from an Apple iPad mini (HDMI video via Apple AV adapter) at the original resolution of 1280x720 with 60 fps. It is recommended to lower the frame rate to 30 fps by setting „fps_divide=2“.

Low bandwidth

At low bandwidth settings it it recommended to increase the GOP size from 16 to a higher value. A GOP size of 16 indicates that an i-frame is generated every 16 frames. As a rule of thumb, the i-frame is approximately 10 .. 20 times larger than a P or B frame. So a larger GOP size uses the bandwidth available more efficiently. Please note that the GOP size may be dynamically changed while encoding to adjust the encoder to a lower or higher bandwidth available on the uplink channel.

4:3 video of a 16:9 source

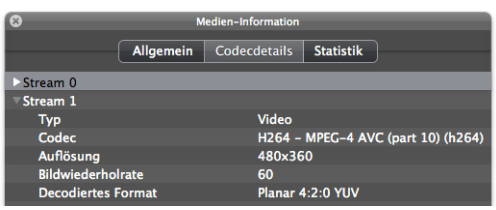
The iPad has a 4:3 display. If this 4:3 video is encapsulated in the 720p output (1280x720) the active video is surrounded on the left and right by black borders. These borders may be cropped of for recording or streaming. The horizontal size of the 4:3 video window with 720 vertical lines is: 720 / 3 * 4 = 960. So (1280 - 960) / 2 = 160 pixels must be cropped of on the left and the right. Please adjust „crop_left=160“ to stream a 960x720 video. The complete POST string is:
http://<ip-address>/api/settings?crop_left=160&width=960&hsize=960&autosize=0

Scaling

Videos may be scaled to a smaller or larger size. By setting the wrong aspect ratio, they may even be distorted. Example: the 960x720 video of the iPad should be reduced to half size (480x360):
crop_left=160&width=960&hsize=480&vsize=360&autosize=0

960x720 video scaled down to 480x360

The top video image shows the scaled video from the iPad. Please note that the „width“ must stay at 960 and the „height“ at 720. If this is changed to 480 and 360, than the video will not be scaled. Instead the video is cropped and only a quarter of the source video is encoded. The second video is cropped down to 480x360 with no scaling:
crop_left=160&width=480&height=360&hsize=480&vsize=360&autosize=0



media information of VLC player (Mac OS 10.9)

960x720 video cropped down to 480x360

RTSP Live Streaming

RTSP stream

The RTSP may be send to local players and servers. As it is a pull stream, it does not support sending the stream to public streaming servers on the Internet, as it cannot cross routers with NAT (network address translation).

If you need to stream to the Internet, please send the stream to a local protocol translator to convert the streaming protocol to RTMP or similar protocols. Please have a look at the Wowza server, VLC or FFMPEG for the protocol translation function. Also compact computers such as the Raspberry Pi may be used for protocol conversion or transmuxing.

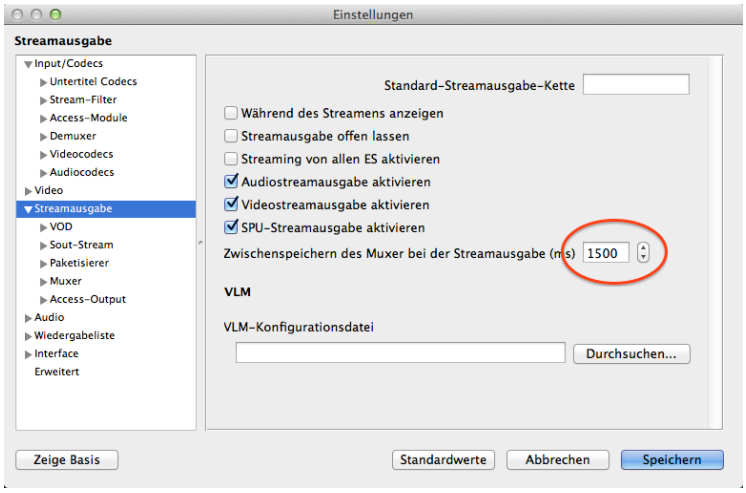
Players

The RTSP stream may be played on the following players:

Quicktime 7
Version 7.6.6 (1709) tested ok

Quicktime 10
Version up 10.2 is ok. Version 10.3 (included in MacOS 10.9) does not play RTSP streams. It no longer falls back to the old QuickTime framework, so it will play just what AVFoundation can play. RTSP streaming is not supported.

VLC
Version 2.1.0 Rincewind (Intel 64 bit) tested ok (on Mac OS 10.9). Please change the buffer setting (default: 1500ms) to reduce the latency.



buffer settings in the VLC player (German version)

Stream Transmuxer

Transmuxer

A transmuxer converts one streaming protocol or video file to another stream protocol or video file. In the transmuxing process the video and audio are not decoded or encoded. The audio and video data is just repackaged. As no computing intensive software is involved, the transmuxing may be performed by little processing resources.

WLAN daughter module

The WLAN daughter module (38094) serves 3 main functions:

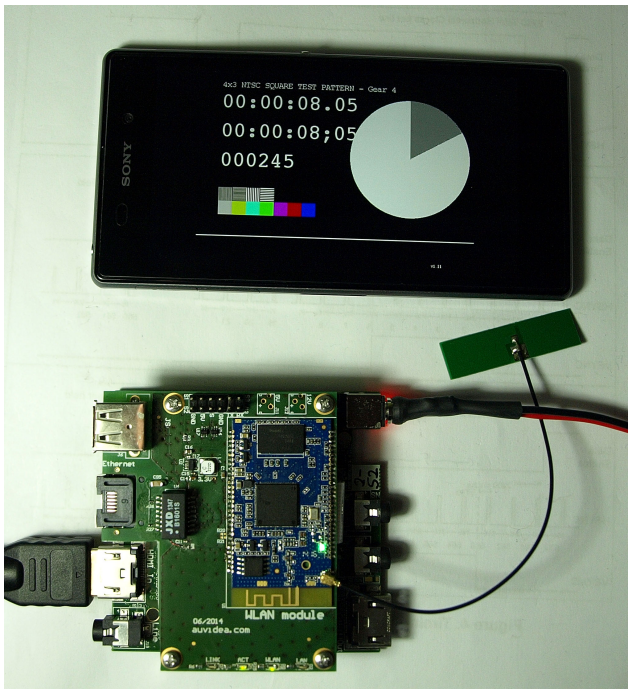
- 2.4 GHz WLAN interface with external antenna (RP-SMA)
- USB port for 3G/4G modems
- RTSP stream transmuxer into other streaming formats (38094-2 with micro SD card)

This module is based of the AR9331 processor and the openWRT operating system (a special Linux for routers). It features a 400 MHz MIPS processor, 64 MB memory, 16 MB Flash, an internal Ethernet port to the E110 (the RJ45 Ethernet connector on the E110 must not be populated), and an external WAN port (RJ45 connector).

openWRT supports a C cross tool chain (development platform), so custom code may be compiled and ported to this module. This is the perfect solution, if it is desired to support streaming formats other than RTSP and if custom software should be added to the E110.

Schedule:

- first prototypes (38094): now
- production (38094-2): September 2014



The 38094 WLAN module is mounted on top of the E110 encoder module. The WLAN module features an U.FL antenna connector and comes with an U.FL to RP-SMA cable, so that any 2.4 GHz antenna may be used. The WLAN module is openWRT based and is configured with a web interface (LuCI).

The 12 pin header features 3 servo outputs and the TTL serial console interface to the WLAN module (AR9331 based).

38094-2 will feature 7 status LEDs located on the front of the board and an on-board micro SD reader for flash extension.

FFMPEG

FFMPEG

FFMPEG is a very versatile transcoder (video encoder and decoder) and transmuxer. It may be ported to various computing platforms and operating systems. For details please visit the site: http://www.ffmpeg.org

Transmuxer

A transcoder converts one video format (codec) to another format (codec). This is very compute intensive so it requires a high performance system. A transmuxer does not recode the audio or video, it just reformats the data. This is a task which only requires limited processing resources, so it may be performed on embedded systems with ARM or MIPS processors.

WLAN module

It is planned to port FFMPEG to the WLAN module.

Compiling FFMPEG

If possible please use a packet manager for your target system to install FFMPEG. Please use the latest release and insure that the required libraries for RTSP and RTMP are installed.

Sample console command

The following command receives an RTSP stream from the E110 and transmuxes it into an RTMP stream, which may be pushed across firewalls to a public server on the Internet.

ffmpeg -i rtsp://192.168.0.160:554/stream -c copy -f rtmp://<server ip address>/<pub point>

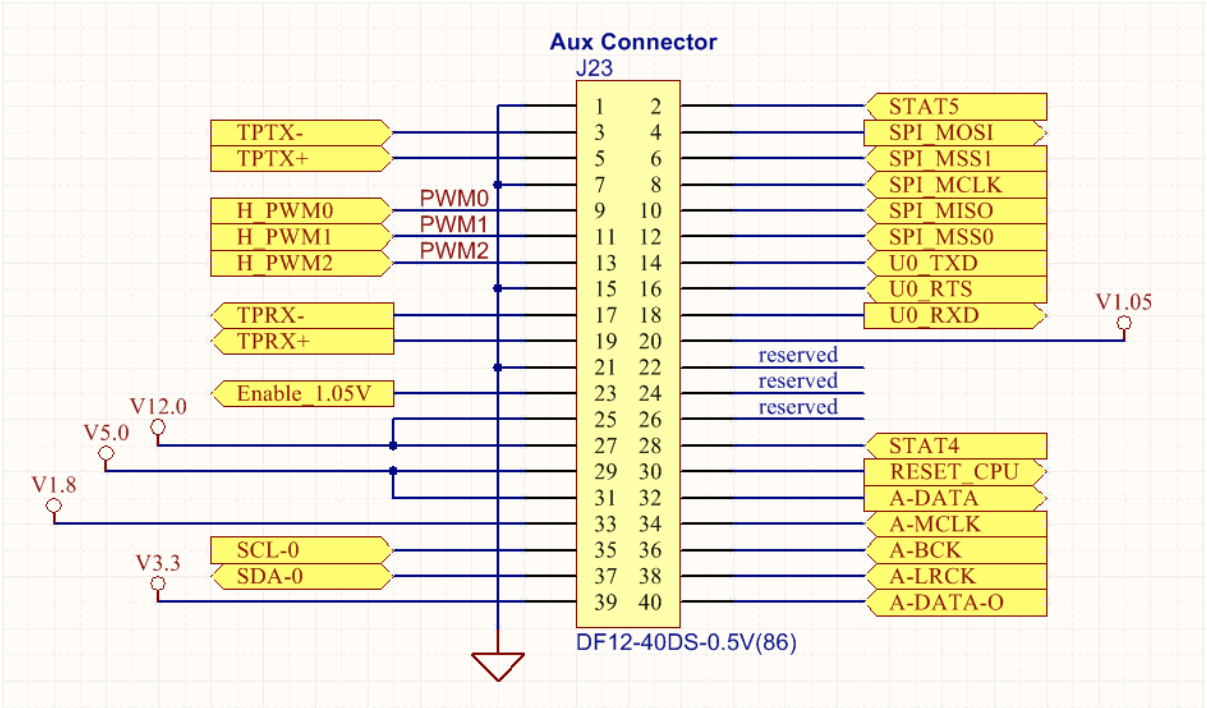
Extension Port

AUX connector: J23

With rev 3 of the E110 the aux connector was redesigned: rev 2 had an 32 pin FPC extension connector, while rev 3 has a 40 pin board to board connector. This allows to add custom hardware to the E110. A typical example is the LED board (38059) and the wifi board (38094). The LED board features LEDs and a STM32F051 Cortex-M3 micro controller, which is a connected to the I2C bus and operates as a slave. So the E110 can read and write registers in the micro controller, to offload functions to micro controller, use it as a GPIO extension and access the integrated RTC (real time clock).

Pins

I2S audio bus	A-DATA-O, A-LRCK, A-BCK, A-MCLK (1.8V)
serial port	U0_RXD, U0_TXD, U0_RTS (RS232 or RS485) (3.3V)
I2C control bus	SCL-0, SDA-0 (3.3V)
RESET_CPU	reset the E110 (open collector, 1.8V)
SPI bus	SPI_MCLK, SPI_MSS0, SPI_MSS1, SPI_MISO, SPI_MOSI (3.3V)
network	10/100 Ethernet (4 pins)
PWM	PWM0, PWM1, and PWM2 (3.3V)
power	power supply: 5.0V, 3.3V, 1.8V, GND



Aux connector on 38056-3 (rev 3 of the E110)

Video input testing

The E110 supports various means, to test the video input.

EDID

The EDID of the HDMI input informs the video source, which video resolutions the E110 supports. So the range of supported video input formats my be restricted. The EDID may be programmed with a custom EDID. Please note, that the EDID data format must be valid, to be recognized. This includes the starting byte pattern and the checksum. For testing purposes the EDID may be set to all „zero“. This is an invalid EDID, but it might be useful for debugging.

HPD

This is an output line of the HDMI input, to inform the video source connected that the input is ready to receive video data and that the EDID may be read.

Cable detect

The video source supplies 5V to the HDMI input. The E110 uses this voltage to determine, whether an HDMI cable and a video source is connected. cable=plugged: 5V are supplied, cable:unplugged: no 5V are supplied, so the E110 assumes that no video source is connected.

EDID enable switch

For testing purpose a read access to the EDID may be blocked. Please use the api call „/api/output“ with „edid=local“ to block EDID access. The default setting is „edid=remote“ to allow remote access to the EDID of the HDMI input.

If you experience problems with the HDMI video source, please use „/api/input“ to analyze the video input parameters. The input timing should conform to the parameters in the table on the following page.

Input timing

The E110 supports various input timings (video resolutions):

Active: the visible resolution of the video (horizontal, vertical)
Total: the total resolution of the video (e.g. 1650 x 750 x 60 = 74,250,000 Hz = pixel frequency)
Timing: 3 horizontal and 3 vertical (front porch, sync width, and back porch)

input timing	resolution	vic	active	total	timing
720p50	1280x720 p (50 Hz)	19	[1280,720]	[1980,750]	[40,440,220,5,5,20]
720p60	1280x720 p (60 Hz)	4	[1280,720]	[1650,750]	[40,110,220,5,5,20]
1080i25	1920x1080 i (50 Hz)	20	[1920,540]	[2640,563]	[44,528,148,5,3,15]
1080i30	1920x1080 i (60 Hz)	5	[1920,540]	[2200,562]	[44,88,148,5,2,15]
1080p24	1920x1080 p (24 Hz)	32	[1920,1080]	[2750,1125]	[44,638,148,5,4,36]
1080p25	1920x1080 p (25 Hz)	33	[1920,1080]	[2640,1125]	[44,528,148,5,4,36]
1080p30	1920x1080 p (30 Hz)	34	[1920,1080]	[2200,1125]	[44,88,148,5,4,36]
NTSC	720x480 i (60 Hz)	7	[1440,240]	[1716,263]	[124,38,114,3,5,15]
PAL	720x576 i (50 Hz)	22	[1440,288]	[1728,313]	[126,24,138,3,3,19]
480p60	720x480 p (60 Hz)	3	[720,480]	[858,525]	[62,16,60,6,9,30]
576p50	720x576 p (50 Hz)	18	[720,576]	[864,625]	[64,12,68,5,5,39]
VGA 60Hz	640x480 p (60 Hz)	1	[640,480]	[800,525]	[96,16,48,2,10,33]
SVGA 60Hz	800x600 p (60 Hz)	0	[800,600]	[1056,628]	[128,40,88,4,1,23]
XGA 60Hz	1024x768 p (60 Hz)	0	[1024,768]	[1344,806]	[136,24,160,6,3,29]

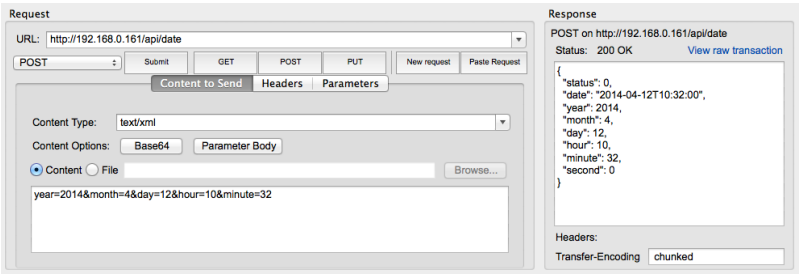
API

The API is HTTP request based and therefore very easy to use. With a simple GET request, internal parameters of the device may be retrieved. With a POST request parameters or commands may be passed to the device.

The GET requests may simply be send by any web browser just by entering the API call as an URL. Example: http://<ip-address>/api/date retrieves the date and time. POST requests my be send by HTML form elements and interactively by some web browser plugins. The web GUI features an API page, which lets you send any GET or POST requests to the system.

Below the Firefox browser and its „HttpRequester“ are used to document a POST request. Just enter the API URL in the field on the top left and the POST request string on the lower left. Any number of parameters may be specified. The delimiter is the „&“ character.

The response is listed on the right formatted in the JSON format.



Firefox HttpRequester sends a POST request to the E110 API

History

Request	Response	Date	ContentLength	ElapsedTime
GET http://192.168.0.161/api/input/edid	200 OK	Apr 13 2014 - 10:48:03 AM	1802	142 ms
GET http://192.168.0.161/api/stream	200 OK	Apr 13 2014 - 10:47:55 AM	41	100 ms
GET http://192.168.0.161/api/record	200 OK	Apr 13 2014 - 10:47:49 AM	60	66 ms
GET http://192.168.0.161/api/settings	200 OK	Apr 13 2014 - 10:47:39 AM	314	78 ms
GET http://192.168.0.161/api/system	200 OK	Apr 13 2014 - 10:47:30 AM	261	363 ms
POST http://192.168.0.161/api/date	200 OK	Apr 13 2014 - 10:34:55 AM	139	156 ms

This API is implemented in a very efficient manner with fast response time and little overhead. Many API requests execute in less than 100ms. All GET API requests have the following format: **http://<ip-address>/api/<api call>.**

Please note that the „options“ must be send as POST request. Sending them with a GET request will not work. Please have a look at the HTML examples.

Feature enhancements

This is the description of the first release of the API. The API is still in development, so new features will be added frequently. These are the main features currently on the to do list:

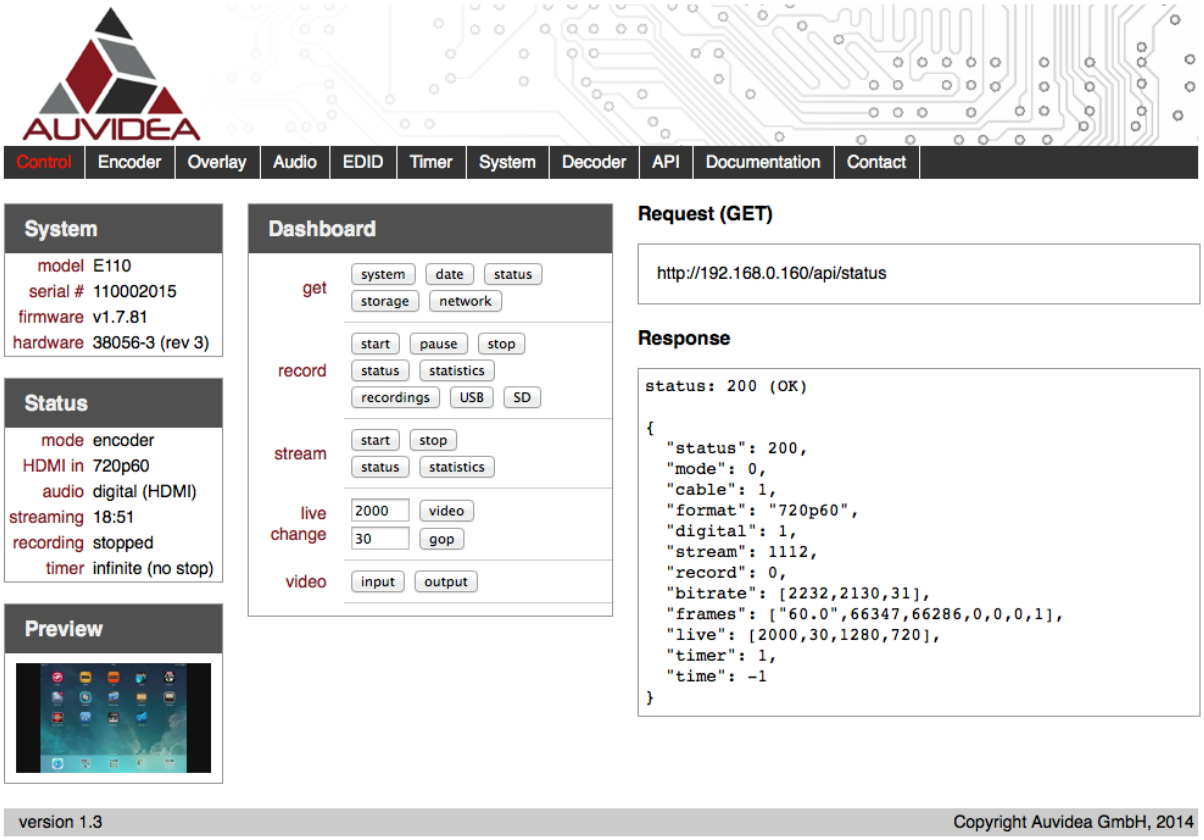
- store and forward of the video files
- dual stream encoding
- audio only encoding

Please let us know, if you would like to recommend new features and functions.

Web based GUI

Web based GUI

The web based GUI allows to easily exercise almost all API functions. The navigation bar lets you navigate to the 8 pages: control, encoder, overlay, audio, EDID, timer, system and API.



Control

This is is master control panel. Here the main functions of the encoder (stream and record) and the decoder (play) may be started. Click on the „Status“ or „Preview“ bars to collapse the fields below. The JPG preview image is updated every 2 seconds. Click on the JPG image, to enlarge it to full size (1280x720 pixels). „live change“ allows to change encoding parameters while streaming or recording.

Encoder

The video, encoding and stream settings of the encoder. Please use the „action=save“ parameter, to save the settings permanently.

Overlay

Up to 7 text lines and a timestamp may be overlaid in the encoded video. No overlay on the loop through video and the decoded video are performed.

Audio

The audio control panel for setting various audio properties.

EDID

Read and write the EDID of the HDMI input.

Timer
Configure a single event timer, to start recording or streaming at a preset time and date. An infinite interval is supported by setting the duration to 0. Autostart is supported as well. Please see the timer page for details.

System
Network, date, www, and logging control.

Decoder
Play out an RTSP audio/video stream (H.264 video and AAC audio). With an E110 as encoder and an E110 decoder you can achieve a very low latency video transmission (typically: 70 .. 100 ms). With software decoders such as the OMX player on the Raspberry Pi or the VLC player to typically latency will be around 1 or 2 seconds. On the VLC player, the latency may be reduced, by reducing the buffer setting (default: 1500ms) in the settings dialog.

API
HTTP request page to send custom GET and POST requests interactively.

Customization of the GUI
All GUI files are located in the www directory. Please go to the www tab in the GUI and press the backup button in the www section. This copies the content of the internal www directory to the USB stick (www_backup/www). You can now modify any of the files. If you just would like to replace the Auvidea logo by a custom logo please replace the logo.png by your custom logo (default size: 980x116 pixel). It is referenced in style.css (#logo). If you would like to change the GUI itself please edit the index.html file.

Tabs can be changed or removed by editing the following section:

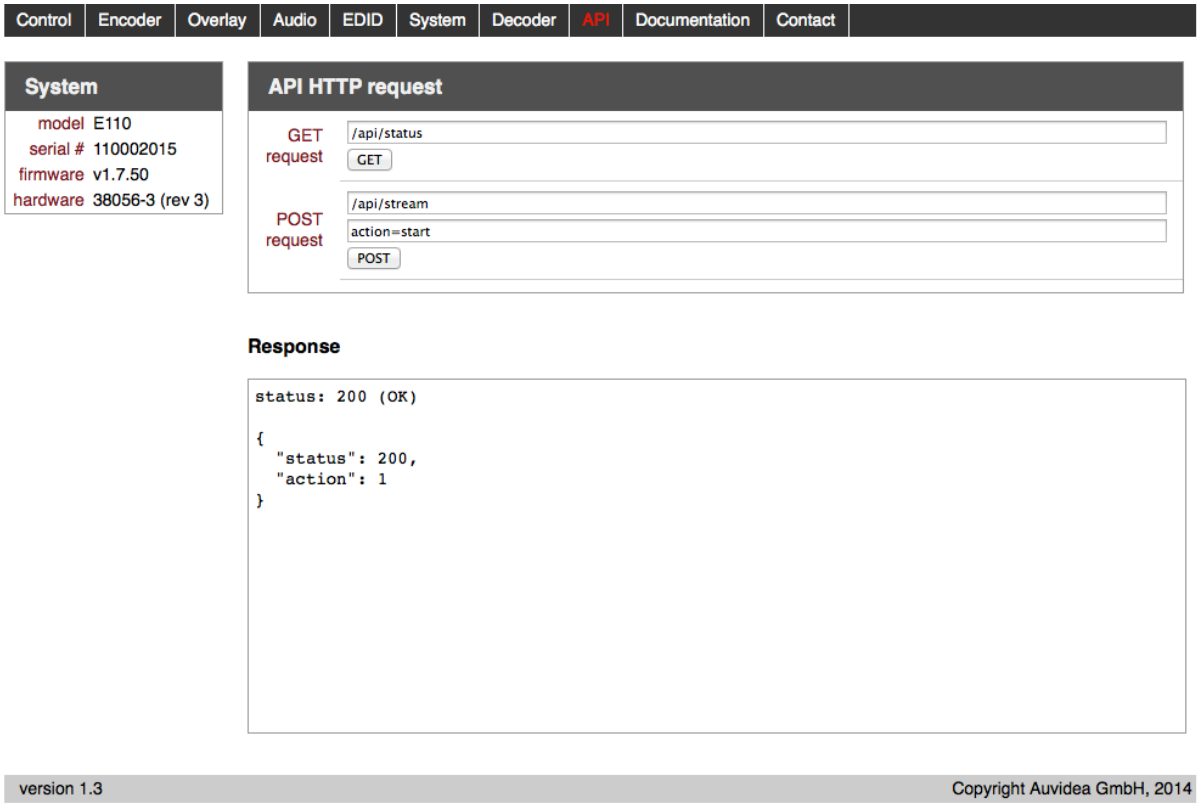
```
<div id="navigation">
  <ul>
    <li><a href="javascript:showonlyone('content_control');" style="color:#E00;">Control</a></li>
    <li><a href="javascript:showonlyone('content_encoder');">Encoder</a></li>
    <li><a href="javascript:showonlyone('content_overlay');">Overlay</a></li>
    <li><a href="javascript:showonlyone('content_audio');">Audio</a></li>
    <li><a href="javascript:showonlyone('content_edid');">EDID</a></li>
    <li><a href="javascript:showonlyone('content_timer');">Timer</a></li>
    <li><a href="javascript:showonlyone('content_decoder');">Decoder</a></li>
    <li><a href="javascript:showonlyone('content_network');">Network</a></li>
    <li><a href="javascript:showonlyone('content_system');">System</a></li>
    <li><a href="javascript:showonlyone('content_api');">API</a></li>
    <li><a href="manual.pdf" target="_blank">Manual</a></li>
    <li><a href="http://www.auvidea.com" target="_blank">Contact</a></li>
  </ul>
</div>
```

After you have made the changes, please move the www directory inside www_backup to the top level of the USB stick. Then press the copy button, to copy the contents of the www directory on the USB stick to the internal www directory.

API HTTP request

Web based API request form

In the web based GUI an API HTTP request form is integrated. This allows to issue custom GET and POST HTTP requests to the API. Just enter the API URL in the first field. For a POST request the mandatory POST string is required, which specifies the POST parameters.



Simple HTML example

The GET calls in the first section call a couple a sample API calls. The <form> element in the second section changes 2 „settings“ parameters (width and height) by displaying a form with 2 input fields and a submit button, which assembles the POST string and sends it to the API interface. This example is included in the E110. Just open „http://<ip-address>/index-min.html“ in a web browser window.

HTML page

HTML code

```
<p><h2>GET calls</h2></p>
<p><strong><a href="api/system">system</a></strong></p>
<p><strong><a href="api/date">date</a></strong></p>
<p><strong><a href="api/network">network</a></strong></p>
<p><strong><a href="api/settings">settings</a></strong></p>
<p><strong><a href="api/audio">audio</a></strong></p>
<p><strong><a href="api/input">input</a></strong></p>

<p><h2>POST calls</h2></p>
<p>
<form name="post1" action="api/settings" method="post">
width <input type="text" name="width">
height <input type="text" name="height">
<input type="submit" value="settings">
</form>
</p>
```

Response

```
{
  "status": 0,
  "width": 900,
  "height": 900,
  ..
}
```

status

GET get the system status (http://<ip-address>/api/status)

This API call is specifically designed to poll the system status in regular intervals and display it on the web page. The parameters „lost frames“ and „fail frames“ should always be 0. This indicates that the encoder performs properly.

Audio and video synchronization
„av sync drop and repeat“ frames are video frames which are inserted or deleted to ensure that audio and video stay 100% in sync. The system master clock of the encoder is derived from the audio clock. As audio and video clocks are sometimes not locked to each other because they may be generated from 2 independent crystals, it is required, that audio and video frames are re-synchronized in regular intervals. In the example below 13 frames were added in 674,000 frames, so roughly one frame every 50,000 frames (at 60 fps this is one frame every 15 minutes). This interval varies with the video and audio sources connected (typically: 10,000 .. 100,000 frames). With many HDMI sources the digital video and digital audio data are perfectly in sync, to no frames have to be deleted or added (no avsync operation).

Parameters (get)

mode	0: encoder, 1: decoder
cable	0: no HDMI cable plugged in, 1: HDMI cable plugged in
format	video input format
digital	0: analog audio in (line), 1: digital audio in (HDMI embedded audio)
stream	0: not streaming, 1+: stream duration in seconds
record	0: not recording, 1+: record duration in seconds
bitrate	average bit rate, current bit rate, audio bit rate
frames	frame rate, total frames, encoded frames, lost frames, fail frames, av sync drops, av sync repeats
live	video bit rate, gop size, encoded video resolution (horizontal, vertical)
timer	0: off, -1: event pending, 1: stream, 2: record, 3: record & stream, 4: play
time	time interval until event start or event start (-1: infinite interval)
codec	status of the H.264 video encoder (off or on)
USB	name of the USB device (empty, if no device mounted)
media	# of video files and # of images recorded

Response

```
{
  "status": 200,
  "mode": 0,
  "cable": 1,
  "format": "720p60",
  "digital": 0,
  "stream": 11244,
  "record": 0,
  "bitrate": [1848,1520,153],
  "frames": ["60.0",674116,673945,0,0,0,13],
  "live": [2000,30,1280,720],
  "timer": -1,
  "time": 85040,
  "codec": "off",
  "usb": "E110",
  "media": [2,3]
}
```

date

GET get the time and date (http://<ip-address>/api/date)

POST set the time and date

Parameters (get and post)

time	xxx	seconds since 1970-01-01 (get only)
isoTime	xxx	ISO formated date string (get only)
year	2014 .. 2100	
month	1 .. 12	
day	1 .. 31	
hour	0 .. 23	
minute	0 .. 59	
second	0 .. 59	
weekday	1 .. 7	day of week (Sunday = 1) (get only)
utcOffset	-12 .. +12	set delta in hours to UTC (get only)
timeZone	GMT, ...	

Response

```
{
  "status": 200,
  "time": 10906,
  "isoTime": "2014-05-12T13:01:46",
  "year": 1970,
  "month": 1,
  "day": 1,
  "hour": 3,
  "minute": 1,
  "second": 46,
  "weekday": 5,
  "utcOffset": 0,
  "timeZone": "UTC"
}
```

date/ntp

GET get the ntp settings (http://<ip-address>/api/date/ntp)

POST set the ntp settings

Parameters (get and post)

action	save, ntp_update	save: permanently save settings ntp_update: retrieve current date/time from ntp server
ntp	server address	default: pool.ntp.org
utcOffset	-12 .. +12	set delta in hours to UTC

Response

```
{
  "status": 200,
  "ntp": "pool.ntp.org",
  "utcOffset": 0
}
```

audio

GET get audio settings (http://<ip-address>/api/audio)

POST set audio properties

A switch from analog to digital audio and vice versa may only performed, while the encoder is not recording or streaming. The switch over time is approximately 6 seconds. If settings should be permanently saved, place the „action=save“ parameter at the beginning of the post string. Only parameters after „action=save“ are permanently saved.

Parameters (get and post)

action	*save: permanently save the settings*
digital	*1: HDMI embedded audio, 0: analog audio*
stereo	*1: stereo, 0: mono (left channel on left and right) (t.d.a.)*
volLeft	*volume on left analog channel (-12 .. +20dB)*
volRight	*volume on right analog channel (-12 .. +20dB)*
muteLeft	*mute analog audio left channel (1: mute, 0: no mute)*
muteRight	*mute analog audio right channel (1: mute, 0: no mute)*
sampleRate	*analog sample rate (HDMI: set by HDMI input signal) (t.b.a.)*
hpLeft	*headphone volume left channel (-68 .. +29dB)*
hpRight	*headphone volume right channel (-68 .. +29dB)*
hpMuteLeft	*headphone mute left channel (1: mute, 0: no mute)*
hpMuteRight	*headphone mute right channel (1: mute, 0: no mute)*

Response


```
{
  "status": 200,
  "digital": 1,
  "volLeft": 0,
  "volRight": 0,
  "muteLeft": 0,
  "muteRight": 0,
  "hpLeft": -10,
  "hpRight": -10,
  "hpMuteLeft": 0,
  "hpMuteRight": 0,
  "micBias": 0,
  "sampleRate": 48000,
  "stereo": 1
}
```

input

GET get the input timing of the HDMI input (http://<ip-address>/api/input)

POST set input properties

„colorInput“ is the color code as received with the AVI info frame. Normally the HDMI receiver chip is set to the same „color“ code. This setting may be manually overwritten (post).

Use action=save&color=x with x = 0,1 or 2 to set the initial color space. This avoids wrong colors with the initial video display.

Parameters (get and post)

cable	*plugged: HDMI cable plugged into HDMI input*
	unplugged: no HDMI cable plugged in
hpd	*on: HPD line active (default), off: HPD line inactive (no video)*
edid	*remote: EDID readable (default), local: EDID blocked*
format	*video format of the HDMI input signal*
active	*video resolution (h, v)*
total	*resolution of the video container (h, v)*
timing	*h front porch, h width, h back porch, v front porch, v width, v back porch*
interlaced	*true or false*
aviInfoframe	*info frame received*
color	*color code overwrite (0: RGB, 1: YUV 4:2:2, 2: YUV 4:4:4) (get and post)*
colorInput	*color code as received in the AVI info frame*
vic	*video information code (0 .. 127)*
audioInfoframe	*audio info frame received*
msInfoframe	*MS info frame received*

Response

```
{
  "status": 200,
  "cable": "plugged",
  "hpd": "on",
  "edid": "remote",
  "format": "720p60",
  "active": [1280,720],
  "total": [1650,750],
  "timing": [40,110,220,5,5,20],
  "interlaced": false,
  "aviIinfoframe": true,
  „aviInfoframeData“: ["0x02“,"0x73","0x50","0xa8","0x00","0x04","0x00","0x00","0x00",
                       "0x00","0x00","0x00","0x00","0x00","0x00"],
  "color": 2,
  "colorInput": 2,
  "vic": 4,
  "audioInfoframe": true,
  "audioInfoframeData": ["0x01","0x70","0x01","0x00","0x00","0x00"],
  "msInfoframe": false
}
```

input/edid

GET get the EDID memory of the HDMI input of this system (http://<ip-address>/api/input/edid)

POST set the EDID memory of the HDMI input of this system with a custom EDID

Sample GET to read decimal EDID setting: http://<ip-address>/api/input/edid?format=1

Parameters (get)

format	*0: read byte values in hexadecimal format (default)*
	1: read byte values in decimal format
base	*128 byte base set*
baseChecksum	*first number: calculated total sum, second number: calculated sum modulo 256 (checksum)*
extended	*128 byte extended set*
extendedCheck.	*first number: calculated total sum, second number: calculated sum modulo 256 (checksum)*

Parameters (post)

action	*save: permanently save the custom EDID*
	default: delete saved custom EDID and revert back to the factory default setting
base	*128 byte base set (128 decimal values separated by ",")*
extended	*128 byte extended set (128 decimal values separated by ",")*

Sample base POST string:

action=save&base=0,255,255,255,255,255,255,0,65,165,1,0,0,0,0,0,26,23,1,3,128,16,9,120,10,238,145,163,84,76,153,38,15,80,84,33,8,0,129,192,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,254,0,86,51,95,48,0,0,0,0,0,0,0,0,0,1,29,0,188,82,208,30,32,184,40,85,64,160,90,0,0,0,30,0,0,0,253,0,24,60,26,81,8,0,10,32,32,32,32,32,32,0,0,0,252,0,85,78,79,10,32,32,32,32,32,32,32,32,32,1,159

Please note that a valid base set must start with the sequence: 0,255,255,255,255,255,255,0. Numbers in hexadecimal format are not supported.

The last byte is the checksum byte (here: 159). All 128 bytes must add up to a multiple of 256 (here: 6912).

The range of each byte is: 0 .. 255.

„action=save“ permanently saves the custom EDID in the file: /data/bin/edid.hex.

„action=default“ reverts back to the factory default EDID setting.

Sample extended POST string:

extended=2,3,32,244,72,147,132,5,20,3,18,160,161,35,9,7,7,131,1,0,0,103,3,12,0,16,0,128,16,226,0,42,1,29,128,208,114,28,22,32,16,44,37,128,160,90,0,0,0,158,1,29,128,24,113,28,22,32,88,44,37,0,160,90,0,0,0,158,1,29,0,114,81,208,30,32,110,40,85,0,160,90,0,0,0,30,140,10,208,144,32,64,49,32,12,64,85,0,160,90,0,0,0,30,140,10,208,138,32,224,45,16,16,62,150,0,160,90,0,0,0,30,0,0,0,0,0,89

The last byte is the checksum byte (here: 89). All 128 bytes must add up to a multiple of 256 (here: 7424).

Response


```
{
  "status": 200,
  "base": [
             "0x00","0xff","0xff","0xff","0xff","0xff","0xff","0x00",
             "0x41","0xa5","0x01","0x00","0x00","0x00","0x00","0x00",
             "0x1a","0x17","0x01","0x03","0x80","0x10","0x09","0x78",
             "0x0a","0xee","0x91","0xa3","0x54","0x4c","0x99","0x26",
             "0x0f","0x50","0x54","0x21","0x08","0x00","0x81","0xc0",
             "0x01","0x01","0x01","0x01","0x01","0x01","0x01","0x01",
             "0x01","0x01","0x01","0x01","0x01","0x01","0x00","0x00",
             "0x00","0xfe","0x00","0x56","0x33","0x5f","0x30","0x00",
             "0x00","0x00","0x00","0x00","0x00","0x00","0x00","0x00",
             "0x01","0x1d","0x00","0xbc","0x52","0xd0","0x1e","0x20",
             "0xb8","0x28","0x55","0x40","0xa0","0x5a","0x00","0x00",
             "0x00","0x1e","0x00","0x00","0x00","0xfd","0x00","0x18",
             "0x3c","0x1a","0x51","0x08","0x00","0x0a","0x20","0x20",
             "0x20","0x20","0x20","0x20","0x00","0x00","0x00","0xfc",
             "0x00","0x55","0x4e","0x4f","0x0a","0x20","0x20","0x20",
             "0x20","0x20","0x20","0x20","0x20","0x20","0x01","0x9f"
            ],
  "baseChecksum": [6912,0],
  "extended": [
             "0x02","0x03","0x20","0xf4","0x48","0x93","0x84","0x05",
             "0x14","0x03","0x12","0xa0","0xa1","0x23","0x09","0x07",
             "0x07","0x83","0x01","0x00","0x00","0x67","0x03","0x0c",
             "0x00","0x10","0x00","0x80","0x10","0xe2","0x00","0x2a",
             "0x01","0x1d","0x80","0xd0","0x72","0x1c","0x16","0x20",
             "0x10","0x2c","0x25","0x80","0xa0","0x5a","0x00","0x00",
             "0x00","0x9e","0x01","0x1d","0x80","0x18","0x71","0x1c",
             "0x16","0x20","0x58","0x2c","0x25","0x00","0xa0","0x5a",
             "0x00","0x00","0x00","0x9e","0x01","0x1d","0x00","0x72",
             "0x51","0xd0","0x1e","0x20","0x6e","0x28","0x55","0x00",
             "0xa0","0x5a","0x00","0x00","0x00","0x1e","0x8c","0x0a",
             "0xd0","0x90","0x20","0x40","0x31","0x20","0x0c","0x40",
             "0x55","0x00","0xa0","0x5a","0x00","0x00","0x00","0x1e",
             "0x8c","0x0a","0xd0","0x8a","0x20","0xe0","0x2d","0x10",
             "0x10","0x3e","0x96","0x00","0xa0","0x5a","0x00","0x00",
             "0x00","0x1e","0x00","0x00","0x00","0x00","0x00","0x59"
            ],
  "extendedChecksum": [7424,0]
}
```

system

GET get the system properties (http://<ip-address>/api/system)

Please feel free to monitor the „freeMemory“ parameter. If this parameter decreases over time and approaches 0, then the system has a memory leak problem. Please send us a bug report with as much information as possible, to that we can duplicate the problem and fix it. And we make sure that we punish the software developers.

Parameters (get)

model	*model description*
hostname	*Linux hostname*
serialNumber	*serial number*
macAddress	*MAC address*
firmware	*version of the firmware (auvidea_enc)*
software	*software version of the system*
hardware	*hardware version of the system*
linux	*Linux version*
freeMemory	*memory available (typically between 5 and 15 MB)*
systemUptime	*uptime in seconds since last cold boot of the system*
uptime	*uptime of the firmware since the last restart*

Response


```
{
  "status": 200,
  "model": "Auvidea E110 H.264/AAC encoder/decoder",
  "hostname": "e110",
  "serialNumber": "110001014",
  "macAddress": "00:40:05:73:54:0E",
  "firmware": "v1.4",
  "version": "software release 1.4",
  "hardware": "38056-2",
  "linux": "2.6.30.mobi.merlin-mg3500.custom",
  "freeMemory": "12164kB",
  "systemUptime": 11320,
  "uptime": 567
}
```

network

GET get the network properties (http://<ip-address>/api/network)

POST set the network properties

Network changes may be made temporarily (until the next boot) or permanently. For permanent changes please use the POST parameter „action=save“. Only parameters after „action“ are permanently saved. So „action“ should be the first parameter in the POST string.

DHCP
Please use this parameter carefully, as this system has little means to communicate the dynamically assigned IP address. DHCP settings changes are active with the next reboot of the system. If „dhcp“ is set to „1“ without „action=save“, then it will effect only the next reboot, and on the reboot after it will revert to „dhcp=0“. Please use this scheme, to first try out DHCP. Please retrieve the IP address from the DCHP tables in your DHCP server (typically your router) or by inserting a USB stick at boot up time. Remove the stick when the boot has completed and open the startup.log file with a text editor. Also you will find a copy of the config file on the USB stick.

Parameters (get and post)

action	*save: permanently save the settings*
ip	*IP address*
subnet	*subnet mask*
gateway	*gateway address*
dns	*DNS (domain name server) address*
mac	*MAC address (get only)*
dhcp	*1: DHCP active, 0: DHCP inactive*
RXbytes	*number of bytes received (Ethernet) (GET only)*
TXbytes	*number of bytes transmitted (Ethernet) (GET only)*

Response

```
{
  "status": 0,
  "ip": "192.168.0.160",
  "subnet": "255.255.0.0",
  "gateway": "fritz.box",
  "dns": "192.168.0.1",
  "mac": „00:40:05:73:53:A7"
  "dhcp": "0"
}
```

led

GET get the LED status (http://<ip-address>/api/led)

POST set the LED status

3 LEDs are located in the front of the board below the connectors as indicated below. „auto“ is the default setting. Here the LEDs are controlled automatically. The rec LED lights up, when the system is recording. The stream LED lights up, when the system is streaming.

Parameters (get and post)

rec	*on, off or auto (red LED below USB connector)*
stream	*on, off or auto (green LED below Ethernet connector)*
power	*on, off or auto (red LED below front audio jack)*

Response

```
{
  "status": 0,
  "rec": "auto",
  "stream": "auto",
  "power": "auto"
}
```

pwm

GET get the pam status (http://<ip-address>/api/pwm)

POST set the PWM parameters

The system features 3 PWM (pulse width modulation) outputs (PWM_0, PWM_1 and PWM_2 on expansion connector J23). The pulse width and the frequency may be set. On the WLAN module these signals control the 3 servos outputs. For standard servos the frequency should be 50 Hz. The pulse high time should vary from 0.5ms (servo=-1000) to 1.5ms (servo=+1000). For other applications the low time of the pulses may be specified directly. At 50 Hz the total low time is approx. 666,000 (timer ticks).

Parameters (get and post)

servo	*servo control from -1000 to +1000 (the low time is computed automatically)*
frequency	*typically 50 Hz*
lowtime	*low time of the pulses (in timer ticks)*

Response

```
{
  "status": 200,
  "servo1": -500,
  "frequency1": 50,
  "lowtime1": 628500,
  "servo2": 0,
  "frequency2": 50,
  "lowtime2": 617000,
  "servo3": 1000,
  "frequency3": 50,
  "lowtime3": 594000
}
```

storage

GET get the properties of the USB device and SD card (http://<ip-address>/api/storage)

This system can either record video files to a USB memory stick or internal micro USB card. Please use „record“, to select the target device for recording.

The internal flash memory stores various applications and system data as well as the www_root directory. If large amounts of data are copied to the www_root directory, then this parameter needs to be monitored.

Parameters (get)

power	*on: power on (5V 500mA), off: power off*
mounted	*yes, no*
name	*name of the storage media*
path	*Linux mount path*
filesystem	*fat (FAT32), ext (Linux formatting: ext2 or ext3)*
total	*total size of the storage media*
free	*size size of the storage media*
flash	*total: total size of internal flash memory, free: free size of internal flash memory*

Parameters (post)

power	*on*	*power on (5V 500mA) (default)*
	off	*power off*

Response

```
{
  "status": 0,
  "power": "on",
  "usb": {
    "mounted": "yes",
    "name": "Intenso",
    "path": "/media/Intenso",
    "filesystem": "fat",
    "total": "1949 MB",
    "free": "328 MB"},
  "sd": {
    "mounted": "yes",
    "name": "SDCARD8GB",
    "path": "/media/SDCARD8GB",
    "filesystem": "fat",
    "total": "7631 MB",
    "free": "7628 MB"},
  "flash": {
    "total": "426 MB",
    "free": "418 MB"}
}
```

settings

GET get the encoding settings (http://<ip-address>/api/settings)

POST set the encoding settings

It is recommended to keep the autosize property set, so that the encoding resolution is automatically configured. To scale the video before encoding or to configure video formats, which are not automatically detected, please set this property to 0. In this case all parameters may be manually configured.

JPEG preview
This features allows to push JPEG images onto a web page for preview purposes. The E110 creates an JPEG image every 2 seconds and overwrites the previous one, as all images are saved unter the same name: <www-root>/image.jpg.

Parameters (get and post)

action	*save: permanently save settings (e.g: action=save&fpsDivide=1)*
width, height	*size of the video before scaling*
hsize, vsize	*size of the video after scaling (modulo 16)*
cropLeft	*cropping from left (must be even)*
cropTop	*cropping fro top*
hstart, vstart	*video positioning (typically: sync width plus back porch)*
total	*total size of the storage media*
interlaced	*1: interlaced, 0: progressive video*
fpsDivide	*encoded frame rate is input frame rate / fps_divide*
gop	*0 .. 1000 (1: I-frame only, 1+: IP frames, 0: infinite size)*
video	*100 .. 10000: video bandwidth in kbps (2000 = 2 Mbit/s)*
audio	*32 .. 256: audio bandwidth in kbps (128 = 128 kbit/s)*
profile	*base, main, or high*
av	*0: audio and video, 1: video only, 2: audio only (t.b.a.)*
autosize	*1: determine encoding resolution from detected video format, 0: manual configuration*
swapUV	*1: swap Cr and Cb color channels (swap red and blue), 0: normal operation*
name	*name of the RTSP stream*
port	*port of the RTSP stream (default: 554)*
preview	*0: no JPEG preview, 1: a JPEG preview image is created every 2 seconds (/tmp/image.jpg)*

Response


```
{
  "status": 200,
  "width": 1280,
  "height": 720,
  "hsize": 1280,
  "vsize": 720,
  "cropLeft": 0,
  "cropTop": 0,
  "hstart": 260,
  "vstart": 25,
  "fps": 60,
  "interlaced": 0,
  "fpsDivide": 2,
  "gop": 30,
  "video": 2000,
  "audio": 128,
  "profile": "high",
  "av": 0,
  "autosize": 1,
  "swapUV": 0,
  "name": "stream",
  "port": 554,
  "preview": 1
}
```

stream

GET get the stream status (http://<ip-address>/api/stream)

POST set the encoding settings

The system supports RTSP live streaming to a local player (such as a VLC player) and a local server (such as a Wowza server). Some parameters may may changed while streaming or recording, to adjust the bandwidth output of the encoder to the bandwidth available on the communications link. However, it is not recommended to change the resolution of the video, while encoding. Players do not accept a live change of the video resolution. Please restart the player.

Parameters (get)

stream	*streaming, stopped*
startTime	*start time of streaming*
address	*address of the RTSP live stream*

Parameters (post)

action	*start, stop*
video	*100 .. 15000: change the video bandwidth while streaming or while recording*
gop	*1 .. 1000: change the top size while streaming or recording*
hsize, vsize	*change the encoding size while streaming or recording (for experimental use only)*

Response


```
{
  "status": 200,
  "action": 0,
  "stream": "streaming",
  "startTime": 11518,
  "address": "rtsp://192.168.0.160:554/stream"
}
```

record

GET get the record status (http://<ip-address>/api/record)

POST set the record settings

Some recording parameters (gop and video bandwidth) may may changed while streaming or recording. Please use the „stream“ call, to adjust these two settings.

Parameters (get)

record	*recording, stopped*
startTime	*start time of recording*
isoStartTime	*start time and date in ISO format*
duration	*recording time in seconds*
file	*video file name and path*
frames	*total number of frames recorded*
lost	*number of frames lost (this should stay 0)*
device	*target recording device (USB or SD)*

Parameters (post)

action	*start, stop*
device	*USB, SD*

Response

```
{
  "status": 200,
  "record": "recording",
  "startTime": 1404075705,
  "isoStarTime": "2014-05-12T13:13:00+0000",
  "duration": 5,
  "file": "/media/Intenso/video_2014-05-12_13-11-53/video.mp4",
  "device": "USB"
}
```

statistics

GET get the encoding statistics (http://<ip-address>/api/statistics)

If recording is active and fail_frames is not zero, some frames could not be written to the recording device. This is an indication, that the memory device being recorded to is not fast enough. Please ensure that a video class 6 or better device (SD card in USB card reader) is used, as these are optimized for recording applications. Most USB memory sticks have low sustained write rates, as sometimes they are busy with internal housekeeping tasks at certain times.

Encoded_frames vs. total_frames: if FPS_DIVIDE is greater than 1, then the video is encoded at a reduced frame rate. Consequently the number of encoded frames is lower than the number of total frames.

Audio and video synchronization
Frames are inserted or deleted to keep the video in sync with the audio. Please see /api/status for details.

Parameters (get)

videoBitrate	*average video bitrate (kbit/s)*
videoCurBitrate	*current video biter ate (kbit/s)*
audioBitrate	*average audio nitrate (kbit/s)*
frameRate	*encoded frame rate (fps)*
totalFrames	*total frames captured (at video frame rate)*
encodedFrames	*number of frames encoded*
failFrames	*number of frames failed (should be 0 for normal encoding)*
avsyncFrames	*drop/repeat frames: inserted/deleted frames, to ensure that audio and video stay in sync*
encoderSize	*encoded video resolution (horizontal, vertical)*
encoderLoad	*hardware and software load of the encoder in percent (here: HW 510% and SW 10%)*

Response


```
{
  "status": 200,
  "videoBitrate": 1848,
  "videoCurBitrate": 1520,
  "audioBitrate": 151,
  "frameRate": "60.0",
  "totalFrames": 693554,
  "encodedFrames": 693382,
  "failFrames": [0,0],
  "avsyncFrames": [0,13],
  "encoderSize": [1280,736],
  "encoderLoad": [51,10]
}
```

recordings

GET get the file list of the recorded video files (http://<ip-address>/api/recordings)

POST delete one video file, which has been recorded

Switch from USB to SD with the „record“ => „device“ parameter.

Parameters (get & post)

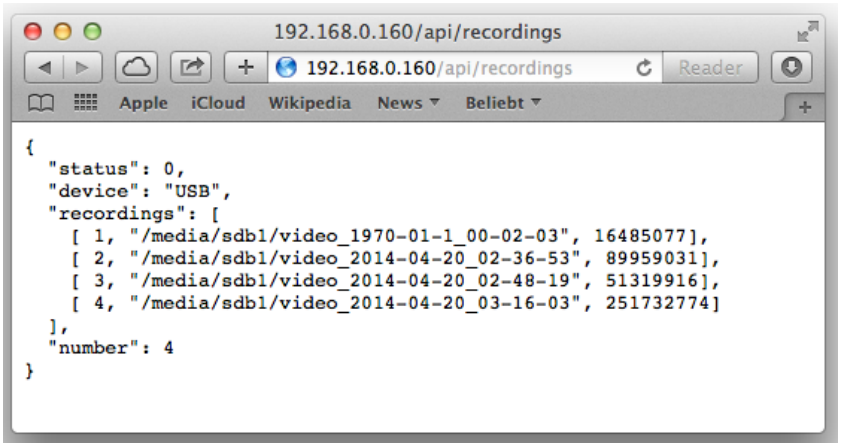
device	*target recording device (USB or SD) (get)*
recording	*list of file names of the recordings (get)*
number	*number of video files recorded (get)*
delete	*delete one video file (post)*

Response

```
{
  "status": 0,
  "device": "USB",
  "recordings": [
    [ 1, "/media/Intenso/video_2014-04-11_13-30-25"],
    [ 2, "/media/Intenso/video_2014-04-11_15-20-00"],
    [ 3, "/media/Intenso/video_2014-04-12_11-49-34"],
    [ 4, "/media/Intenso/video_2014-04-12_14-00-15"]
  ],
  "number": 4
}
```

Post error codes

406	*video file is being recorded (please stop recording first)*
407	*bad directory path*
408	*no USB device name*
409	*no USB device*
410	*video file path not valid*



GET request issued simply with the URL (Safari browser on Mac OS 10.9)

timer

GET get the timer status (http://<ip-address>/api/timer)

POST set the timer settings

Schedule a recording or streaming event. Minimum duration: 60 seconds. Set duration to 0, to create an event of infinite length.

Auto start of system
Set „time“ to a time in the past. Best set it to time=0 (1970-01-01 0:00), as this is the time the system is set to, if there is no access to the ntp server. Set duration to 0. Alternatively to an extremely large number (e.g. 2,000,000,000). Then select the mode, the system should auto start with. This will set the auto start mode. Please make sure, to use the „action=save“ parameter, so that these settings are saved in the config file for permanent save. „action=save“ mud be the first parameter in the post string, as only parameters after the „action=save“ are saved permanently.

Parameters (get and post)

action	*save: save settings permanently*
time	*record or stream start time*
ISO_time	*record or stream start time in ISO format*
duration	*duration of the event - in seconds (60: minimum, 0: infinite)*
mode	*0: off, 1: stream, 2: record, 3: record & stream, 4: play*
startIn	*time interval in seconds, until timer event will start*
remaining	*time interval in seconds, until timer event will stop*
recording	*1: recording in progress (started by timer)*
streaming	*1: streaming in progress (started by timer)*
playing	*1: playing in progress (started by timer)*
active	*1: timer has started streaming, recording or playing*
pending	*1: timer pending (recording, streaming or playing will start at „time“)*
result	*-1: idle, 0: completed, 1 .. 8: record could not start, 10: record started*

Response


```
{
  "status": 200,
  "time": 1405955220,
  "isoTime": "2014-07-21T15:07:00",
  "year": 2014,
  "month": 7,
  "day": 21,
  "hour": 15,
  "minute": 7,
  "second": 0,
  "duration": 1000,
  "mode": 1,
  "startIn": 84591,
  "remaining": 0,
  "recording": 0,
  "streaming": 0,
  "playing": 0,
  "active": 0,
  "pending": 1,
  "result": -1
}
```

play

GET get the play properties (http://<ip-address>/api/play)

POST set the play properties

„play“ can decode an RTSP video stream and play it out on the HDMI output port.

Note: the E110 can easily switch from encode (record and stream) to decode (play) mode, but it cannot switch back from decode to encode. To switch back the E110 needs to be powered down and up again (cold reboot). If the decoder does not receive a video stream, it will show show a blue screen. If the stream got interrupted, the decoder will sense this and restart, waiting for the stream to continue. In this state, it will also show a blue screen.

Parameters (get and post)

play	*status: playing or not playing*
startTime	*start time of recording*
isoStartTime	*start time and date in ISO format*
duration	*recording time in seconds*
address	*IP address of the RTSP video encoder*
name	*stream name of the RTSP stream*
video	*video resolution: 720p50, 720p60, 1080i25, or 1080i30 (1080i 50 and 60 fps)*
format	*rtsp*
av	*0: audio and video, 1: video only*
ntsc	*pixel clock: 0: 74.25MHz (25/30/50/60Hz), 1: 74.17MHz (29.97/59.94Hz)*
sampleRate	*audio sample rate (e.g. 48000)*
timeoutStreamLoss	*timeout for stream loss - after this encoder is restarted and will wait for stream*
timeoutNoStream	*timeout for no stream - play was started, but no stream was received - after this interval the encoder will be restarted (this may be set to large number)*

Parameters (post)

action	*start*	*start the play of an RTSP stream*
	stop	*stop the play of an RTSP stream*
	save	*permanently save the settings*

Response


```
{
  "status": 200,
  "state": 1,
  "play": "playing",
  "startTime": 1407332537,
  "isoStartTime": "2014-08-06T13:42:17+0000",
  "duration": 193,
  "address": "192.168.0.165",
  "name": "stream",
  "video": "720p50",
  "format": "rtsp",
  "av": 0,
  "ntsc": 0,
  "sampleRate": 48000,
  "timeoutStreamLoss": 1,
  "timeoutNoStream": 50
}
```

playStatistics

GET get the play statistics (http://<ip-address>/api/playStatistics)

Parameters (get)

state	*0: waiting for stream, 1: playing, 2: stream ended, 3: stopped*
videoBitrate	*average video bit rate (kbit/s)*
	current video bit rate (kbit/s)
frameRate	*video frame rate (fps)*
decodedFrames	*number of video frames decoded*
	number of audio frames decoded
receivedFrames	*number of total video frames received*
	number of video frames dropped
	number of video frames repeated
	number of video underflows
clocks	*video pixel clock (Hz)*
	audio sample rate (Hz)
decoderLoad	*hardware load of the decoder (%)*
	software load of the decoder (%)

Response


```
{
  "status": 200,
  "state": 0,
  "videoBitrate": [1960,1943],
  "frameRate": "49.9",
  "decodedFrames": [37737,35402],
  "receivedFrames": [37736,7,0,0],
  "clocks": [74250000,48000],
  "decoderLoad": [22,2]
}
```

output

GET get the properties of the HDMI output (http://<ip-address>/api/output)

POST set the color mode of the HDMI output

Recommendation: for most applications leave „color“ in the default setting for true loop through, as the HDMI output will have the same color model as the HDMI input. For debugging purposes the HDMI output may be connected to the HDMI input. This allows to read the EDID of the HDMI input and the HPD output line of the HDMI input.

Parameters (get)

power	*on: enable 5V 500mA on HDMI output, off: enable 5V 55mA on HDMI output*
hpd	*on: HPD input line active, off: HPD input line inactive*
vic	*vic code in the AVI info frames, which are send out to the monitor (mirrored from input)*
color	*color code of the HDMI output*
colorInput	*color code of the HDMI input (AVI info frame of HDMI input)*
total	*total resolution of the video put out (this should match the input resolution)*

Parameters (post)

power	*on*	*high power mode (5V 500mA)*
	off	*low power mode (5V 55mA)*
color	*0*	*set HDMI out to RGB*
	1	*set HDMI output to YUV 4:2:2*
	2	*set HDMI output to YUV 4:4:4*
	3	*the HDMI output automatically follows the HDMI input color mode (default)*

Response

```
{
  "status": 200,
  "power": "on",
  "hpd": "on",
  "vic": 4,
  "color": 3,
  "colorInput": 2,
  "total": [ 1650, 750 ]
}
```

output/edid

GET get the EDID memory of the monitor, which is connected to the HDMI output in the back (http://<ip-address>/api/output/edid)

Parameters (get)

base	*128 byte base set*
extended	*128 byte extended set*

Response


```
{
  "status": 200,
  "base": [
             "0x00","0xff","0xff","0xff","0xff","0xff","0xff","0x00",
             "0x4c","0x2d","0x26","0x06","0x00","0x00","0x00","0x00",
             "0x1f","0x13","0x01","0x03","0x80","0x10","0x09","0x78",
             "0x0a","0xee","0x91","0xa3","0x54","0x4c","0x99","0x26",
             "0x0f","0x50","0x54","0xbf","0xef","0x80","0x71","0x4f",
             "0x81","0x00","0x81","0x40","0x81","0x80","0x95","0x00",
             "0x95","0x0f","0xa9","0x40","0xb3","0x00","0x02","0x3a",
             "0x80","0x18","0x71","0x38","0x2d","0x40","0x58","0x2c",
             "0x45","0x00","0xa0","0x5a","0x00","0x00","0x00","0x1e",
             "0x01","0x1d","0x00","0xbc","0x52","0xd0","0x1e","0x20",
             "0xb8","0x28","0x55","0x40","0xa0","0x5a","0x00","0x00",
             "0x00","0x1e","0x00","0x00","0x00","0xfd","0x00","0x18",
             "0x4b","0x1a","0x51","0x17","0x00","0x0a","0x20","0x20",
             "0x20","0x20","0x20","0x20","0x00","0x00","0x00","0xfc",
             "0x00","0x53","0x79","0x6e","0x63","0x4d","0x61","0x73",
             "0x74","0x65","0x72","0x0a","0x20","0x20","0x01","0x48"
           ],
  "extended": [
             "0x02","0x03","0x1c","0xf1","0x4b","0x93","0x04","0x05",
             "0x14","0x03","0x12","0x10","0x1f","0x20","0x21","0x22",
             "0x23","0x09","0x07","0x07","0x67","0x03","0x0c","0x00",
             "0x10","0x00","0xb8","0x2d","0x01","0x1d","0x80","0xd0",
             "0x72","0x1c","0x16","0x20","0x10","0x2c","0x25","0x80",
             "0xa0","0x5a","0x00","0x00","0x00","0x9e","0x01","0x1d",
             "0x80","0x18","0x71","0x1c","0x16","0x20","0x58","0x2c",
             "0x25","0x00","0xa0","0x5a","0x00","0x00","0x00","0x9e",
             "0x01","0x1d","0x00","0x72","0x51","0xd0","0x1e","0x20",
             "0x6e","0x28","0x55","0x00","0xa0","0x5a","0x00","0x00",
             "0x00","0x1e","0x8c","0x0a","0xd0","0x90","0x20","0x40",
             "0x31","0x20","0x0c","0x40","0x55","0x00","0xa0","0x5a",
             "0x00","0x00","0x00","0x1e","0x8c","0x0a","0xd0","0x8a",
             "0x20","0xe0","0x2d","0x10","0x10","0x3e","0x96","0x00",
             "0xa0","0x5a","0x00","0x00","0x00","0x1e","0x00","0x00",
             "0x00","0x00","0x00","0x00","0x00","0x00","0x00","0xc7"
           ]
}
```

www

GET get the www root directory content (http://<ip-address>/api/www)

POST modify the content of the www root directory

This functions to manage the directory content of the www_root directory. There should not be a subdirectory named „api“, as this might collide with the API calls.

Please use the „clear“ option carefully, as the complete www_root directory is wiped out. On the next system restart the logical link to the image.jpg file in the ramdisk (/tmp) is automatically restored (image.jpg -> /tmp/image.jpg). Please note that a „clear“ also deletes the web GUI. So please backup these files first.

Parameters (post)

action	*clear*	*empty the www_root directory (delete all files)*
	copy	*copy files from „<usb device>/www“ to www_root*
	backup	*copy files from www_root to „<usb device>/backup/www“*

Response (get)


```
{
  "status": 200,
  "www_root": [
    "       53353 Jul 20 12:53 Auvidea_Logo_980x116.png",
    "        5726 Jul 20 12:58 api.html",
    "       10434 Jul 20 12:58 audio.html",
    "        4536 Jul 20 12:57 decoder.html",
    "        4704 Jul 20 12:56 edid.html",
    "       14279 Jul 20 12:54 encoder.html",
    "        8119 Jul 19 17:29 functions.js",
    "          14 Jul 20 17:15 image.jpg -> /tmp/image.jpg",
    "         756 Jan  1  1970 index-min.html",
    "       13366 Jul 20 16:42 index.html",
    "      141763 Jan  1  1970 jquery.js",
    "     3593665 Jan  1  1970 manual.pdf",
    "        4711 Jul 13 20:06 no_video_preview.png",
    "       13710 Jul 20 12:56 overlay.html",
    "         646 Jul 12 15:03 player.html",
    "        4271 Jul 20 14:50 style.css",
    "       16842 Jul 20 17:00 system.html",
    "        7367 Jul 20 12:43 timer.html",
    "       13798 Jan  1  1970 video.html"
    ]
}
```

upgrade

POST upgrade control (http://<ip-address>/api/upgrade)

The firmware and software may be upgraded by supplying a special upgrade file on a USB memory device. The upgrade should only take a few seconds. After the upgrade a restart should be issued, to load the new firmware. Or better do a cold boot. Restart may be used independently of the upgrade, to just perform a warm boot. Important: the sequence in the post string must be: „file=<filename>“ and then optionally the „action=restart“ parameter. Or just „file=<filename>“ or just „action=restart>.

Parameters (post)

file	*<filename>*	*name of the upgrade file on USB memory device*
action	*restart*	*warm boot the new firmware*

Response (status)

status	*0*	*upgrade successful (plus version and message are shown)*
	1	*file not found*
	2	*illegal upgrade file*
	3	*restart illegal, as system is recording or streaming*
	4	*restart initiated*

Response

```
{
  "status": 0,
  "version": "software 1.4",
  "message": "upgrade successful"
}
```

log

GET get API call log (http://<ip-address>/api/log)

POST set logging settings

Parameters (post)

action	*start*	*start logging API calls*
	stop	*stop logging (default)*
	clear	*clear entire log*
autoclear	*on*	*automatically delete entire log once 100 calls have been logged*
	off	*stop after logging 100 calls (default)*

Response

```
{
  "status": 200,
  "quantity": 5,
  "log": [
    [  1, "GET ", "/api/system", "", 0 ],
    [  2, "GET ", "/api/network", "", 0 ],
    [  3, "GET ", "/api/settings", "", 0 ],
    [  4, "POST", "/api/settings", "port=554&name=teststream", 24 ],
    [  5, "GET ", "/api/log", "", 0 ]
    ]
}
```

text overlay (OSD)

POST set text overlay parameters (http://<ip-address>/api/text)

Up to 8 text strings with up to 24 characters each may be instantiated. The position of each string may be specified with the following restriction: the horizontal position must be a multiple of 64, the vertical position must be a multiple of 16.

Sample POST string: enable=1&index=2&x=900&y=96&text=sample text overlay

The horizontal and vertical position is automatically adjusted to the nearest legitimate value.

Parameters (post)

enable	*0: text overlay disabled, 1: text overlay enabled*
index	*0 .. 7: index of the text string*
font	*8, 16 or 32: font size (8x8, 16x16, or 32x32)*
x	*horizontal position*
y	*vertical position*
text	*overlay text string (up to 24 characters)*

Response

```
{
  "status": 200,
  "enable": 1,
  "index": 2,
  "font": 16,
  "x": 896,
  "y": 96,
  "text": "sample text overlay",
  "length": 19
}
```

timestamp overlay

POST set timestamp overlay parameters (http://<ip-address>/api/timestamp)

One timestamp with 22 characters may be instantiated. The position of the timestamp overlay may be specified with the following restriction: the horizontal position must be a multiple of 64, the vertical position must be a multiple of 16.

Sample timestamp string: 2014-06-19 13:59:40:20

The last number is the frame counter.

Sample POST string: enable=1&index=1&x=128&y=16&font=16

The horizontal and vertical position is automatically adjusted to the nearest legitimate value.

Parameters (post)

enable	*0: text overlay disabled, 1: text overlay enabled*
index	*0 .. 7: index of the text string*
font	*8, 16 or 32: font size (8x8, 16x16, or 32x32)*
x	*horizontal position*
y	*vertical position*

Response

```
{
  "status": 200,
  "enable": 1,
  "index": 1,
  "font": 16,
  "x": 128,
  "y": 16
}
```

timeout

POST admin control (http://<ip-address>/api/timeout)

The E110 features a software watchdog, which may be enabled to guarantee a reliable 24/7 operation. The watchdog is automatically kicked by the E110 firmware. If the firmware stops kicking the watchdog and it times out after the preset time, the firmware and any encoding or decoding processes are restarted. For a true hardware reset watchdog please use the 38116 GPIO add-on board.

On timeout the auvidea_enc is terminated, as it has started the watchdog process. If the auvidea_enc process cannot be terminated, then Linux will reboot. When the auvidea_enc process is terminated, the 99init script will restart this process in an endless while loop.

Parameters (get and post)

action	*save: permanently save the timeout settings*
timeoutEnable	*0: stop timeout feature*
	1: start timeout feature
timeoutCount	*time period until the timeout process times out (5 .. 10000 seconds)*
timeoutTest	*0: normal operation*
	1: stop kicking the timeout process, to force a timeout (no permanent save of this option)

Response

```
{
  "status": 200,
  "timeoutEnable": 1,
  "timeoutCount": 45,
  "timeoutTest": 0
}
```

rs232

GET get status of RS232 port (http://<ip-address>/api/rs232)
POST send data out on the RS232 port (http://<ip-address>/api/rs232)

The E110 can send and receive data on the RS232 to control a Panasonic block camera. In this mode the last to bytes to be send to the camera are automatically appended. The first byte is the checksum of the data packet. The last byte is always 255 (0xFF) to terminate the data packet. The firmware then waits for the acknowledge (250) and any data returned by the camera.

Parameters (get and post)

data	*list of bytes in decimal format to be send out via the RS232 port to the camera (post)* *list is comma separated, the value range is 0 .. 255 for each value*

Response

```
{
  "status": 200
}
```

GPIO add-on module (38116-2)

POST admin control (http://<ip-address>/api/addon)

This add-on module features a micro-controller, which implements various UI and monitoring functions. 4 switches are monitored. 4 LEDs are controlled. A hardware watchdog monitors the operation of the E110 firmware. Should the E110 stop kicking the watchdog (I2C write access to register 8 with a dummy data byte), then the micro controller will time out after a predefined time and assert a hardware reset to the E110. This action is flagged by the LED (D3). It will be on for 5 seconds. Normally it blinks, which indicates that the microc ontroller is operating normally.

Parameters (get)

action	*save: permanently save the watchdog settings (post only)*
model	*model number of the add-on module*
firmware	*firmware version of the micro controller on the add-on module*
time	*real time clock (reset on each power up) - RTC still to be implemented*
buttons	*record, stream, JPEG, and misc (1: button currently pressed)*
temperature	*temperature in Celsius of the temp sensor on the GPIO module*
dcInput	*main input voltage (typical: 12V)*
dc3V3	*internal 3.3V power supply*
dc1V8	*internal 1.8V power supply*
dcAin	*external analog input (P14 pin 1)*
watchdogEnable	*0: watchdog disabled, 1: watchdog enabled (get and post)*
watchdogCount	*time until the watchdog times out (5 .. 255 seconds) (get and post)*
watchdogTest	*0: normal operation, 1: stop kicking watchdog (for test purposes) (get and post)*

Response


```
{
  "status": 200,
  "model": "38116-2 GPIO module",
  "firmware": 16,
  "time": 4927,
  "buttons": [0,0,0,0],
  "temperature": "35.8C",
  "dcInput": "11.8V",
  "dc3P3": "3.30V",
  "dc1P8": "1.78V",
  "dcAin": "0.0V",
  "watchdogEnable": 1,
  "watchdogCount": 248,
  "watchdogTest": 0
}
```

admin

POST admin control (http://<ip-address>/api/admin)

This admin interface is password protected, to protect the system. A system specific password may be requested. Please see the chapter „System Control“ for details.

Parameters (post)

password	*xxxx*	
action	*startSSH*	*start the SSH server for root level access*
	stopSSH	*stop the SSH server (default setting)*
	getPS	*retrieve the process list*
	restart	*restart the audio/video encoder (this resets the frame counters and re-initializes all encoding parameters) - no password protection (note: it does not restart the firmware)*

Response

```
{
 "ps": „
    PID USER       VSZ STAT COMMAND
      1 root      1496 S    init
      2 root         0 SW<  [kthreadd]
      3 root         0 SW<  [ksoftirqd/0]
      4 root         0 SW<  [events/0]
      5 root         0 SW<  [khelper]
      8 root         0 SW<  [async/mgr]
     55 root         0 SW<  [kblockd/0]
     75 root         0 SW   [pdflush]
     76 root         0 SW   [pdflush]
     77 root         0 SW<  [kswapd0]
     78 root         0 SW<  [aio/0]
    187 root         0 SW<  [ksuspend_usbd]
    192 root         0 SW<  [khubd]
    227 root         0 SW<  [scsi_eh_0]
    228 root         0 SW<  [usb-storage]
    367 root         0 SW<  [mtdblockd]
    390 root      1496 S    /bin/sh /etc/init.d/rcS
    429 root       888 S <  /sbin/udevd --daemon
    661 root      1508 S    /sbin/syslogd -S -C
    663 root      1488 S    /sbin/klogd
    722 root         0 SW<  [i2c-mux]
   1151 root         0 SW<  [kmmcd]
   1260 root      1496 S    /usr/sbin/inetd
  "
}
```

FAQ

1. E110 shows an initial video with wrong colors

During bootup the color space need to be changed. Please use the API call: /api/input with POST action=save&color=x where x = 0, 1 or 2.

Disclaimer

Thank you for reading this manual. If you have found any typos or errors in this document or any bugs or issues in the software or API, please let us know.

The Auvidea Team