CELF Audio Video Graphics Specification v2.0

Document: CelfAudioVideoGraphicsSpec2_*accepted_20060606*

This is an accepted specification of the CE Linux Forum. Send comments on this version to:

celinux-dev@tree.celinuxforum.org

or AVGWG@list.celinuxforum.org (CELF forum members)

Revision History

Revision	Comment	Reviewer	Editor	Date
Draft1	Initial version		Ruud Derwig	2005-11-07
Draft2	Filled in Audio, Video and Graphics parts		Christophe Hoeppe, John Vugts	2005-11-29
Draft3	Added work in progress		Ruud Derwig	2005-12-01
Draft4	Changed e-mail address for non-member feedback	AVG WG: Mitsubishi, NVIDIA, Philips, Samsung, ST	Ruud Derwig	2005-12-13
Draft5	Incorporated AVG WG feedback	AVG WG	Ruud Derwig, Christophe Hoeppe	2006-02-13
Proposed1	AVG WG feedback, changed [M] into [S]	AVG WG	Ruud Derwig	2006-03-09
Accepted	Approved version	CELF BoD	Ruud Derwig	2006-06-06

0. Intro

Audio, video, and graphics processing is at the core of many CE products. The AVG requirements for CE devices are different than those for PCs/Servers, notably with respect to available resources (e.g. a general CPU v.s. dedicated HW to perform certain tasks), footprint, input devices, interlacing, streaming, etc.. Multiple graphics planes and video planes may be combined using, e.g., alpha blending.

Different platform vendors develop different hardware platforms using the Linux operating system that can be integrated with middleware software in various Audio/Video consumer products by the CE equipment manufacturers. Today for each hardware platform, a different code base is developed, each one providing its own, specific platform API. As a result, the different CE equipment manufacturers each have to do a major investment for creating Audio/Video products using the different platform APIs.

The CELF forum and specifically the Audio Video and Graphics WG is specifying a common set of software interfaces that can be used by multiple platform vendors and CE equipment manufacturers and as such reduce the overall investment required to implement the Audio/Video products. This common API approach supports the development of open and shared assets for this industry.

The common API should support a wide range of products (e.g. digital Set Top Boxes or TV sets, and mobile phones) to maximise the gain of using the API. It should also support a wide range of different platform implementations and implementation techniques (e.g hardware or software media processing) to allow for a wide adoption and use of the API. The API should efficiently support the relevant existing middleware stacks in e.g. digital TV products to allow the usage of existing assets.

For CE products the costs of the product with reference to e.g. Silicon size, CPU cycles, Power consumption, Required Memory and Memory bandwidth are of critical importance. Therefore it should be possible to implement a common API in a very efficient way. The API should support HW acceleration to reduce the possible load on the CPU in such a product. Furthermore, the API should be very scalable to be able to scale from low-end standard products to high-end and more featured products.

A common, consistent and well-defined CELF AVG API is required to enable the different parties to concentrate on the added value of their business. A semiconductor company can concentrate on making a cost effective implementation of this interface. A middleware company can make use of the economy of scale by deploying their middleware on a larger base of products, and they can focus on the middleware services instead of spending time on porting to different APIs. A CE company can select assets from multiple vendors and can focus on adding their specific features to their integrated product.

The CELF AVG API should fit in with the Linux operating system and should fit with Linux existing solutions, especially the ones that were part of the current AVG spec 1.0. However, the goal of this specification is to provide direction and reduce fragmentation, so choices do need to be made. Having a well defined, well supported interface for AVG devices will reduce fragmentation of solutions and encourage the CE community to develop solutions that apply to conforming interfaces, so that they can be deployed across a wider range of systems.

1. Rationale

It is the goal of this specification to make choices and provide direction for selecting the Audio, Video and Graphics interfaces to be used in CE products. Making choices is always difficult, especially when the playing field for a part already established and the existing solutions are so heterogeneous, many and disperse as in the Linux audio, video, graphics domain. Many existing solutions today have their roots in the PC domain, where multimedia functionality has grown very fast the past years. PC architectures and solutions, however, are not always adequate for CE devices.

The interfaces proposed in this document fulfil to a large extend the requirements that were gathered by the CELF Audio Video Graphics Working Group [1]. The main drivers for the choices that were made are:

- The CELF AVG API should support a range of products with the same API where there is a functional overlap. This should now or in the future include all CE domains: Home (Set Top Boxes, analog/digital TVs, digital TVs, Personal Video Recorders applications, Digital Media Adapter products etc.), Mobile (smartphone, featurephone, portable media players, etc.), and Automotive (car infotainment, navigation, etc.)
- The CELF AVG API should provide freedom in the implementation and abstract from different implementations like HW or SW streaming, single or multiple processor solutions, single or multi chip solutions and HW accelerated or pure SW media processing implementations. This allows for different implementations on different targets (e.g. CE products or standard PCs).
  This requirement translates to a preference for a functional, user-space interface over ioctl based device driver interfaces, in order to allow for pure software, hardware, and hardware accelerated implementations.
- The CELF AVG API must support scalability and be able to deal with a large diversity in products (e.g. from a TV product to a STB product etc).
- The CELF AVG API is required to enable the different companies using the API to concentrate on the added value of their business (e.g. a platform implementation on a chip set, or a middleware implementation, or a product integration). For this reason it is important to handle interfaces as independent assets from implementations of the CELF AVG API.
- The CELF AVG API should be well documented, have a consistent and single "look & feel" with consistent syntax, style, documentation, semantics, behaviour, and should have no internal overlap.
- Requirements will be changing over time, implying changes to interfaces. This must be manageable on the interface specifications.
- The CELF AVG API should be an open interface, available without any restrictions or implied costs to all parties that want to make of use it, and it should be possible by any party to implement the CELF AVG API without any restrictions or implied costs other than the actual implementation costs.
- Last, but not least, the CELF AVG API should fit with the Linux operating system and the existing solutions which are part of the current AVG spec 1.0 i.e. DirectFB for graphics. It should allow for an efficient implementation on a Linux based system, leveraging available open source implementations.

Based on these requirements the specification for audio, video and graphics are defined in the next sections of this document. The main solutions the specification is based on are:

- **Universal Home API (UHAPI)**
  This standard has been defined by the industry consortium called UHAPI Forum and is proposed for ISO/IEC standardization via the MPEG organization [2,3]. The standard will be referred to as ISO/IEC 23004: M3W (Multimedia Middleware). An open source implementation of a growing number of UHAPI interfaces is called UHAPI4Linux [4]. Whenever UHAPI interfaces are

mentioned in this document, the UHAPI specification 1.1 or better is meant. An introduction to UHAPI can be found in [8,9], details are in [10].

- **DirectFB**,
  DirectFB was already part of the CELF AVG 1.0 specification, and has been implemented by many CE/silicon vendors [5].
- **OpenGL ES**
  OpenGL ES is the de-facto standard for 3D graphics in the mobile domain [6].
- **Framebuffer**
  The framebuffer device is the de-facto low-level standard for many CE devices.
- **ALSA**.
  ALSA is the audio solution part of the Linux kernel [7]. Though it is tailored to the PC+soundcard architecture and does not cover all CE domain requirements, it is a widely adopted solution that can be used and complemented if needed for CE devices requiring more features.

1.1 References

[1] : Consolidated requirements document at
http://tree.celinuxforum.org/CelfPriWiki/AudioVideoGraphicsSpec2

deeplink:
http://tree.celinuxforum.org/CelfPriWiki/AudioVideoGraphicsSpec2?action=AttachFile&do=get&target=CELF_AVG_requirements_v1.1.pdf

[2] The official site of the UHAPI specifications, at www.uhapi.org

[3] UHAPI specification version 1.1 version, donated by the UHAPI Forum to CELF (complying with CELF non-member submission license requirements), at
http://tree.celinuxforum.org/pubwiki/moin.cgi/UHAPI

[4] UHAPI4Linux open source implementation of UHAPI interfaces:
http://sourceforge.net/projects/uhapi4linux/

[5] The DirectFB specification at www.directfb.org

[6] The OpenGL ES specification at http://www.khronos.org/opengles/spec/

[7] ALSA specification at http://www.alsa-project.org or directly http://alsa.opensrc.org/

[8] UHAPI Technical white paper: http://www.uhapi.org/technology/white_papers/uhapitechpaper-v4.1.pdf

[9] UHAPI overview presentation:
http://www.uhapi.org/home/news/UHAPI_Architecture_IFA_20050905.pdf

[10] UHAPI Readers Guide: uhAPISpecificationReadersGuide.pdf, part of the full UHAPI specification package from http://tree.celinuxforum.org/pubwiki/moin.cgi/UHAPI and
http://www.uhapi.org/technology/specification/spec_download/

2. Terminology

2.1 Acronyms and terms

Term	Definition
ALSA	Advanced Linux Sound Architecture -- functional level audio API, now standard in 2.6 Linux kernels, replacing OSS.
API	Application Programmers Interface
ATSC	Advanced Television Systems Committee. American standard body for digital television broadcasting.
CE	Consumer Electronics: a class of devices used in the home or on the move. Includes DVD, DVR, PVR, PDA, TV, set-top box, cellular phones, etc.
CVBS	The analog Composite Video, Blanking Signal: a format for an analog television signal before it is modulated onto an RF carrier.
DLNA	Digital Living Network Alliance, DLNA aligns industry leaders in the CE, mobile, and PC industries through digital interoperability.
DVB	Digital Video Broadcast: European standards body for digital television broadcasting.
FB,Framebuffer	Abstraction of video-out hardware with a low level (ioctl) API. Standard in >2.4 Linux kernel (see the /usr/src/linux/Documentation/fb kernel tree directory for more information).
HDMI	High-Definition Multimedia Interface, uncompressed, all-digital audio/video interface. HDMI supports standard, enhanced, or high-definition video, plus multi-channel digital audio on a single cable.
ISDB	Integrated Services Digital Broadcasting, Japan standard body for digital television broadcasting.
MPEG-1/2/4	Moving Picture Experts Group: a compression standard for digital audio & video with varying levels of complexity and achievable compression ratios.
NTSC	National Television Systems Committee: American standard for analog television broadcasting.
PAL	Phase Alternating Line standard for analog television broadcasting.
PVR	Personal Video Recorder: a consumer electronic device.
SECAM	Système Electronique Couleur Avec Mémoire: French standard for analog television broadcasting.
SCART	The Scart (Syndicat des Constructeurs d'Appareils Radiorécepteurs et Téléviseurs) connector is used for combined audio and video connections.
SPDIF	Sony/Philips Digital Interface, a standard audio file transfer format. Developed jointly by the Sony and Phillips corporations, S/PDIF allows the transfer of digital audio signals from one device to another without having to be converted first to an analog.
SPTS	Single Program Transport Stream, see TS

STC	System Time Clock, reference clock for obtaining synchronization of audio/video streams
TS	Transport Stream: one or more digital streams (usually, MPEG-2 files) multiplexed into a single stream. A Transport Stream consists of fixed length packets.
VBI	Vertical Blanking Interval. The part of a TV signal that is sent between each video frame. This non-viewable part of the signal is used to transmit data like teletext and closed-caption content
V4L	Video for Linux: low level (ioctl) video input and overlay API, standard in the 2.4 Linux Kernel. Originally designed for control of analog video capture and tuner cards, as well as parallel port and USB video cameras.
V4L2	Video for Linux, second version, made to be more flexible and extensible. Added specifications for digital tuner control and capture.
Y/C	Colorspace representation commonly used in digital video broadcasts, and video compression technologies such as MPEG. It uses three orthogonal components, one for luminance (Y) and two for the color-difference signals (Cr,Cb).
YUV	Colorspace representation commonly used in European TV broadcast. It is similar but not the same as Y/C

2.2 Compliance classifiers

Terminology conventions are adopted here as they are defined in IETF RFC 2119, "Key words for use in RFCs to Indicate Requirement Levels" (by S. Bradner, March 1997). A compliance classifier from the following set may be used:

- [M]ust, Required, Shall: This is the minimum set of requirements. The CELF based products are expected to comply with these requirements when expressed in unconditional form. A conditional requirement expressed in the form, "If X, then Y must be implemented", means that the requirement "Y" must be met when the conditional aspect "X" applies to a given implementation.
- [S]hould, Recommended: Recommended items are optional items that are strongly recommended for inclusion in CELF based products. The difference between "recommended" items and "optional" items, below, is one of priority. When considering features for inclusion in a product, recommended items should be included first.
- [O]ptional, May: Optional items are suggestions for features that will enhance the user experience or are offered as a less preferred choice relative to another recommended feature. If optional features are included, they should comply with the requirement to ensure interoperability with other implementations.
- E[X]pressly Forbidden: This term means that an item must not be incorporated in a CELF based product.

3. Specification

3.1 Audio

3.1.1 Rationale

When considering the processes required for an audio subsystem in the CE context, it becomes interesting to make an inventory of the set of functionalities available in most popular Linux APIs that relate to audio: among others, the audio processing functionality available in ALSA (and OSS) has been evaluated and compared to what UHAPI can provide.

Analysis has turned out (see references) that, in the CE context, ALSA-API as well as ALSA-libs are offering a relatively small subset of the functionality that are provided in the UHAPI framework, (as depicted in Figure 1: Relation of UHAPI and ALSA in the context of a simple Analog Broadcast Decoding use case.).

To summarize:

- The UHAPI specifications are covering Analog and Digital Broadcast decoding functionality (including the ATSC standard, see Table 3-1), and the widest range of typical audio post-processing and rendering facilities used in CE devices, see Table 3-1and Table 3-2, respectively;
- The ALSA-API and ALSA-libs offer mostly functionality related to digitizing, synthesizing, writing/reading files, offered are also a couple of basic audio features, such as volume, balance, mixing, and tone control, and a wide range of MIDI interfaces; it is to be noted that the ALSA tools package allows decoding of AC-3 stream.
- Compared to other Linux APIs providing also some audio functionality (a.o LinuxDVB, Video4Linux 2), UHAPI has the widest coverage for all standards and for audio processing algorithms (besides the usual volume, mute, mixing, balance and tone control that one can find in other Linux APIs).

Other important reasons for choosing UHAPI as a solution for the audio related functionality in a system are, a.o:

- The concepts behind UHAPI make it possible to extend the range of applications covered by the UHAPI solution: new components can be defined and implemented to cover applications such as "portable audio", where one could define a generic audio file encoder and a generic file decoder.
- It supports a wide range of product families due to the scalability being designed into the API.
- It is well documented.
- It provides a mature, wide and proven set of interfaces for controlling the audio part of a platform.
- It provides an implementation independent API, this allows one to choose different implementations on different platforms for controlling dedicated solutions for audio in embedded devices.

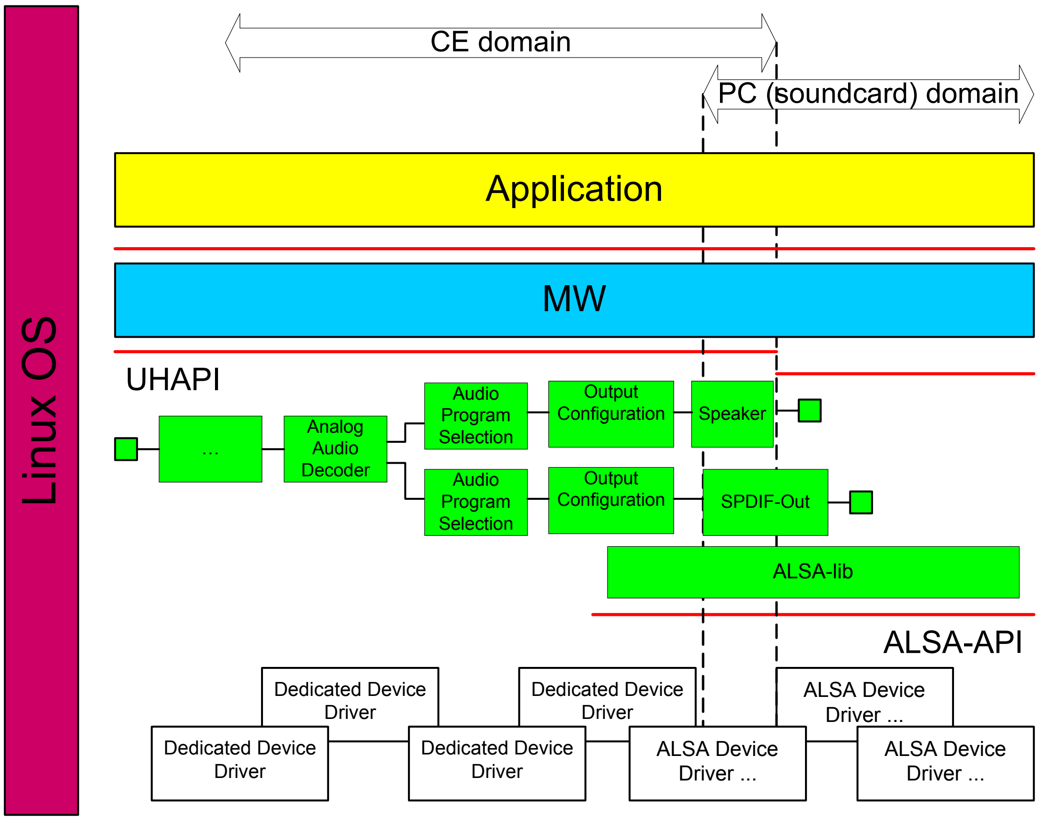

Figure 1: Relation of UHAPI and ALSA in the context of a simple Analog Broadcast Decoding use case.

3.1.2 Specification

[S] For controlling broadcast audio decoding functions, the audio interfaces specified by the UHAPI Forum should be used. This includes the interface specifications as listed in Table 3-1. See Table 3-3 in the notes section for a description of the individual specifications.

Note that only those specifications that are relevant for the product are being used and implemented in that product. So for example, if a compliant product has no SPDIF inputs, then the uhISpdifIn specification is not used.

Analog Audio Decoding	SPDIF-in
ATSC Decoder	SPDIF-out
SPDIF Decoding	

Table 3-1: Broadcast audio decoding specifications.

[S] For controlling audio processing and rendering functions, the audio interfaces specified by the UHAPI Forum should be used. This includes the interface specifications as listed in Table 3-2. See Table 3-4 in the notes section for a description of the individual specifications.

Audio Automatic Volume Levelling	Audio Program Selection

Audio Bass Enhancement	Audio Volume Control
Audio Dynamic Range Control	Equalizing
Audio Mixing	Output Configuration
Audio Noise Generation	Speaker Set/Headphones

Table 3-2: Audio processing and rendering specifications.

[S] For controlling PC (sound cards) domain functionality (synthesizing, MIDI interfaces), the ALSA functional level interfaces should be used.

[O] For other domains, requirements and solutions have not been analysed in depth for this version of the CELF specification. Existing ALSA and UHAPI interfaces may be used.

[O] Implementations of the UHAPI audio interfaces for non-PC domain functionality may internally use ALSA or OSS or other – dedicated - implementations..

3.1.3 Notes

In this section, the individual specifications with their name and a brief description of the addressed functionality in the specification are listed.

Specification	Name	Description
uhIAnaAdec	Analog Audio Decoding	Decodes an encoded analog audio input stream, and outputs the decoded programs. Determines the sound standard of the incoming stream from a set of allowed sound standards and provides notifications of various changes that occur in the input stream.
uhIAtscDec	ATSC Decoder	This component is responsible for demultiplexing and decoding an ATSC compliant Transport Stream (TS) into its constituent components.
uhISpdifDec	SPDIF Decoding	Decodes an originally SPDIF-formatted encoded audio input stream, and outputs a decoded audio stream. Determines the compression standard of the encoded audio incoming stream and provides notifications of various changes that occur in the input stream.
uhISpdifIn	SPDIF-in	Parses the data from a digital input signal that complies to the SPDIF format. The parsed meta data is available to the client. Both meta data and the audio data (linear PCM or encoded audio) are available at the output.
uhISpdifOut	SPDIF-out	Transforms various formats of a digital audio stream to a SPDIF IEC compliant stream.

Table 3-3: Description of the broadcast audio decoding specifications.

In Figure 2 an example graph is shown based on the interface specifications listed for decoding a digital and an analog input signal.

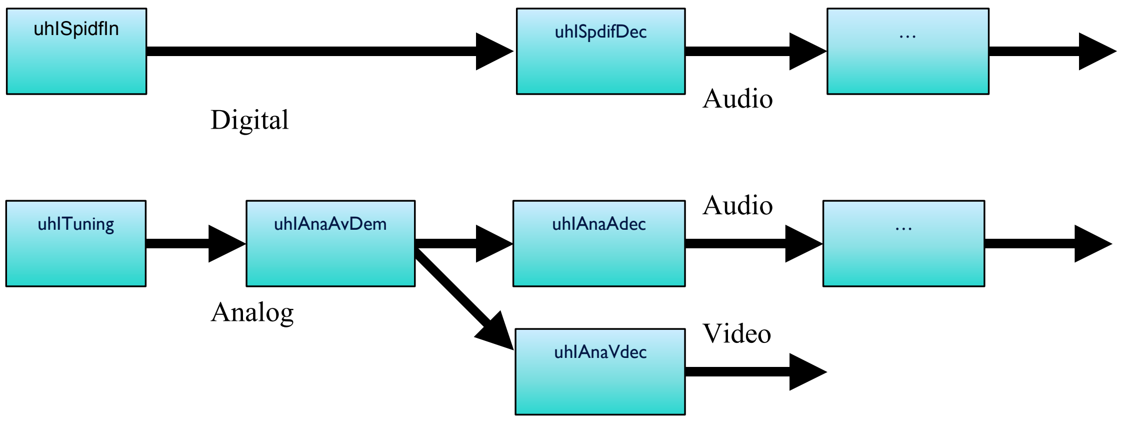

Figure 2: Simplified example Audio decoding graph.

Specification	Name	Description
uhIAavl	Audio Automatic Volume Levelling	Compensates for the variation in the volume level, that are e.g. caused by a change in the audio content (advertisements) or a change in the source of the incoming audio signal, resulting in less annoyance for the end-user.
uhIAbassEnh	Audio Bass Enhancement	Provides control on a set of signal processing algorithms that improve listening experience for low-frequency (bass) sounds. An appropriate choice of the algorithm depends on the kind of bass rendering capabilities the system speakers have.
uhIAdrc	Audio Dynamic Range Control	Reduces the dynamic range of the audio incoming signal to get low-level and high-level signals closer together in level, the incoming signal can thus be adapted to the reproduction apparatus and/or the listening conditions.
uhIAmix	Audio Mixing	Mixes a number of incoming (uncompressed, baseband) audio streams into a single ouput audio stream; individual control is provided to mute, unmute, and control the gain of each of the incoming streams.
uhIAnoiseGen	Audio Noise Generation	Provides control on the injection of noise into audio channels to be fed to a set of speakers (noise injected a channel at a time), enabling the calibration of the inter-channel relative reproduction levels at the listening position.
uhIAprogSel	Audio Program Selection	Selects one (or more) audio program (e.g. a language) from an audio incoming stream with multiple programs.
uhIAvolCtrl	Audio Volume Control	Provides control on the sound level of the audio signal outputted by the audio playback system (master

		volume); also provides control on muting of the audio signal, and on Loudness processing.
uhIEqualizer	Equalizing	Provides control to adjust the signal level of the audio incoming signal in various frequency bands.
uhIOutConf	Output Configuration	Provides control to a set of algorithms to get an audio stream with a specific output configuration from an audio incoming stream: selection among a set of possible algorithms covering surround processing, virtualizing, downmixing and upmixing processes and selection of the number of channels in the output stream.
UhISpkr	Speaker Set/Headphones	Activates the speakers to be used in the system and provides control to bass-redirection, trim control and delay functionalities.

Table 3-4: Description of the audio Processing & Rendering.

In Figure 3 an example graph is shown based on the interface specifications listed in Table 3-4.

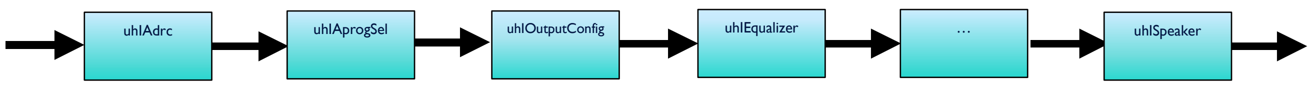

Figure 3 Example Audio processing & Rendering graph.

3.1.4 References

This section lists references to other standards or to implementations.

Comparison of Linux Audio interfaces	http://tree.celinuxforum.org/CelfPubWiki/AvgAudioAPIs
The official site of the UHAPI specifications:	www.uhapi.org
UHAPI specification version 1.1 version, donated by the UHAPI Forum to CELF	http://tree.celinuxforum.org/pubwiki/moin.cgi/UHAPI
Open Source implementation of UHAPI:	http://sourceforge.net/projects/uhapi4linux/
ALSA specification:	http://www.alsa-project.org or directly http://alsa.opensrc.org/
LinuxDVB specification:	www.linuxtv.org or directly http://www.linuxtv.org/downloads/linux-dvb-api-1.0.0.pdf
Video4Linux 2 specification:	http://v4l2spec.bytesex.org/spec/

3.2 Video

3.2.1 Rationale

3.2.1.1 Analog and Digital Broadcast

The required control interfaces for the analog and digital broadcast video is to a large extent defined by the broadcast standards (NTSC, PAL, SECAM, ATSC, DVB, ISDB). Compared to other Linux APIs, UHAPI has the widest and most uniform coverage for all standards. Currently ISDB is not yet supported (by any known API). Support for ISDB can be added by just adding one decoder component (e.g. uhIIsdbDec).

Other important reasons for choosing UHAPI as a solution are:

- It supports a wide range of product families due to the scalability being designed into the API.
- It is well documented.
- It provides a mature, wide and proven set of interfaces for controlling the video part of a platform.
- It provides an implementation independent API, this allows one to choose different implementations on different platforms for controlling dedicated solutions for video in embedded devices.
- UHAPI matches well with existing implementation in the Linux domain like e.g. LinuxDVB. LinuxDVB could for example be used as an implementation as demonstrated by the open source project: UHAPI4Linux.

3.2.1.2 PVR

Many solutions exist for basic, low-level timeshift / recording solutions. UHAPI provides a consistent solution to support timeshift / recording functionality and the required controls to be DLNA compatible. This and the fact that the UHAPI solution for PVR is consistent with the broadcast solution is the reason to choose for UHAPI in the PVR domain.

3.2.1.3 Networked Media

At this point in time the Networked Media domain is not yet supported by this CELF AVG specification. There are no good existing solutions known yet. This is to be added in a future extension of the CELF AVG specification.

Within the UHAPI forum a new WG is started that is specifically addressing Networked Media also known as the Digital Media Adapter domain. This should be considered as part of the future extension of the CELF AVG specification.

3.2.1.4 Mixing Video & Graphics and controlling multiple planes

DirectFB and UHAPI provide complementary functionality, but have some overlap. The main focus of DirectFB is Graphics, and the main focus of UHAPI is Audio and Video control. Both use very similar interface technology (Vtable based interfaces, and explicit functional interfaces (not ioctl based). The abstraction level of DirectFB and UHAPI is also very similar. Both provide primitive interfaces to abstract the acceleration HW (different HW platforms), and focus on providing the middleware with a cross platform control interface.

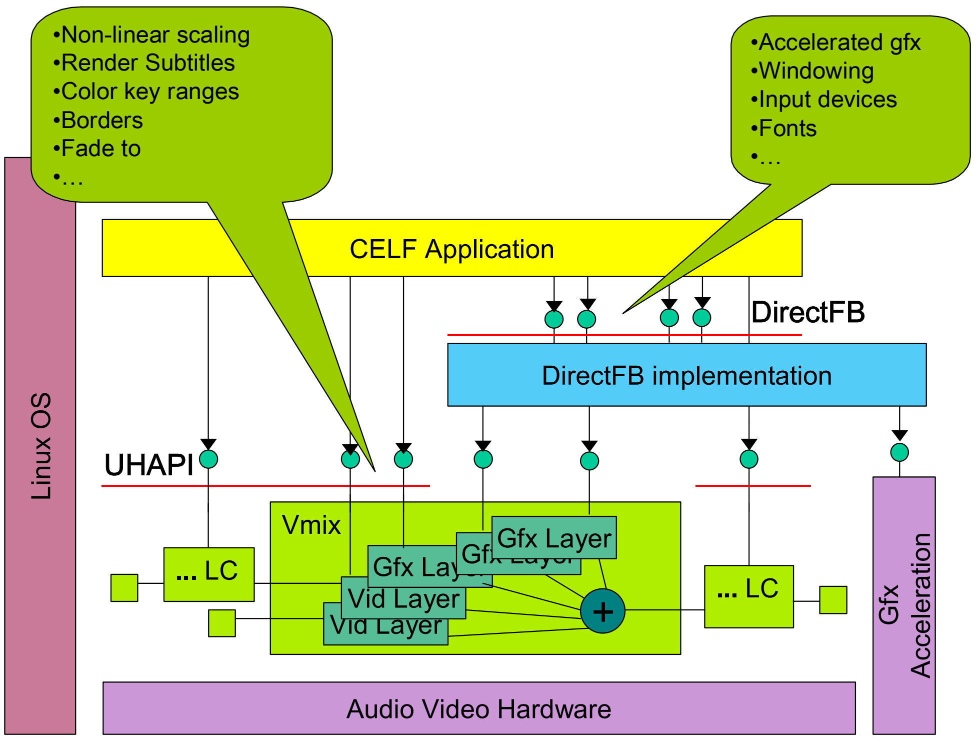

Figure 4 DirectFB in relation with the uhIVmix interface specification

Both DirectFB and UHAPI provide mechanism to compose different layers (video and gfx) into one output (e.g. by means of blending). This is exactly the area where DirectFB and UHAPI meet. The picture below illustrates the combination of UHAPI and DirectFB. Given the richer functionality of UHAPI it has been chosen to use UHAPI for the control of video planes and optional graphics planes for video related functionality like subtitles. A number of planes can be assigned to DirectFB for used for all kinds of graphics purposes.

An alternative solution for mixing video and graphics is to define multiple framebuffer devices (/dev/fb0, dev/fb1, …) in combination with some specific ioctl's for controlling the layering and composition. The preference for functional style interfaces instead of ioctls and additional support for e.g. non-linear scaling etc, has lead to the choice for the UHAPI/DirectFB solution.

3.2.2 Specification

[S] For controlling broadcast video decoding functions, the video decoding interfaces specified by the UHAPI Forum should be used. This includes the following interface specifications as listed in Table 5. See Table 8 in the notes section for a description of the individual specifications.

Note that only those specifications that are relevant for the product are being used and implemented in that product. So for example, if a compliant product has ATSC decoding functionality in it, then this ATSC decoding functionality is to be controlled by the middleware via the ATSC Decoder specification. In case of an analog TV product, the ATSC Decoder specification is of course not relevant and is not used. In this case the Analog Video Decoding specification is of course used.

Analog Audio & Video Demodulation	RF Amplification

Analog Video Decoding	Signal Strength
ATSC Decoder	STC Decoder
Analog AV Input	Transport Stream Demultiplexing
Analog AV Output	Transport Stream Multiplexing
Channel Decoding	Tuning
HdmiIn	URL Source
Image Decoding 2	VBI Slicing
Out Of Band Tuning & Demodulation	

Table 5 Broadcast decoding specifications.

[S] For controlling video processing and rendering functions, the video interfaces specified by the UHAPI Forum should be used. This includes the following interface specifications as specified in Table 6. See Table 9 in the notes section for a description of the individual specifications.

Ambient Level	Histogram Modification
Analog Video Encoder	Noise Measurement
Analog Video Encryption	Scan Rate Conversion 2
Anti Aging	Sharpness Enhancement
Black Bar Detection	Sharpness Measurement
Video Color Enhancement	Basic Video Featuring
Color Transient Improvement	Video Mixing
Dynamic Noise Reduction	

Table 6 Video Processing & Rendering.

[S] For controlling PVR video functions, the PVR interfaces specified by the UHAPI Forum should be used. This includes the following interface specifications as specified in Table 7. See Table 10 in the notes section for a description of the individual specifications.

Data Injecting	SPTS Transmuxing
Data Extracting	

Table 7 Personal Video Recording related.

Note that the interface specifications mentioned in Table 7 are added to support PVR functionality. However, for a complete product typical broadcast interface specifications as mentioned in Table 5 and Table 6 are also used.

[O] For other domains requirements and solutions have not been analysed in depth for this version of the CELF specification. Existing UHAPI interfaces may be used.

[O] Implementations of the UHAPI interfaces may internally use existing V4L(2) or Linux DVB implementations.

3.2.3 Notes

In this section the individual specifications with their name and a brief description of the addressed functionality in the specification are listed. Some examples are given to indicate the context of the different specifications. For the actual individual specifications, please refer to the specifications on the UHAPI web site where they can be downloaded (www.uhapi.org).

Specification	Name	Description
UhIAnaAvDem	Analog Audio & Video Demodulation	Demodulate the signal, and output the resulting Y/CVBS video and encoded audio signal. Determines the TV system of the incoming signal so the sound carrier can be located.
UhIAnaVdec	Analog Video Decoding	Decode a composite video stream (e.g. CVBS, Y/C) into a component decoded signal (e.g. YUV) or to decode a component video stream into another type of component video stream. It also notifies properties of the decoded video signal.
UhIAtscDec	ATSC Decoder	This component is responsible for demultiplexing and decoding an ATSC compliant Transport Stream (TS) into its constituent components.
UhIAvIn	Analog AV Input	Detect base band signal properties associated with an AV input cluster (e.g. a SCART) and notifies changes of these properties.
UhIAvOut	Analog AV Output	Mute audio and video outputs and signal base band signal properties associated with an AV input cluster (e.g. a SCART).
UhIChanDec	Channel Decoding	Control digital demodulation for DVB cable (DvbC), terrestrial (DvbT), and satellite (DvbS) reception.
UhIHdmiIn	HdmiIn	Extracts info frames from a HDMI stream and passes them on to the client.
uhIImageDec2	Image Decoding 2	Decode full images to a requested format, get thumbnails in a requested format, and extract raw meta-data.
UhIOob	Out Of Band Tuning & Demodulation	Receive an external out of band signal, tune to the correct frequency and demodulate that signal. This is typically used by clients that adhere to the OpenCable POD host control interface standard.
UhIRfAmp	RF Amplification	Control the amplification of an incoming RF signal in order to optimize the signal processing by a tuner. Typically needed in regions where transmissions are weak, over modulated, or heavily distorted.
UhISigStrength	Signal Strength	represents functionality to measure the strength of an incoming IF signal.
UhIStcDec	STC Decoder	Provide a constant time base for synchronized presentation of components (audio, video, subtitles, etc.)

		of a digital MPEG stream. The STCD synthesizes the STC from time stamps that have been extracted from an MPEG stream (in push mode), or it bases the STC on an autonomously running clock that is linked to, but not driven by, the PTS values in the decoded video or audio elementary stream (in pull mode).
UhITsDmx	Transport Stream Demultiplexing	TS Demux represents functionality to filter specific content (e.g. PES packets, sections) from a transport stream (TS).
UhITsMux	Transport Stream Multiplexing	Control the rate and number of insertions per section into this transport stream and output this multiplexed transport stream.
UhITuning	Tuning	Tune to a specific frequency (for e.g.. an antenna, cable or satellite dish). It can perform many operations like tuning, fine-tuning and optionally Automatic Frequency Control (AFC) and searching on the input signal.
UhIUrlSrc	URL Source	A URL Source is a streaming source that streams data identified by a URL into the platform (based on protocols like e.g. http, ftp, rtsp, file etc.).
uhIVbiSlice	VBI Slicing	Extract Vertical Blanking Interval data from an analog video stream.

Table 8 Description of the broadcast decoding specifications.

In Figure 5 an example graph is shown based on the interface specification listed for decoding a digital and an analog input signal.

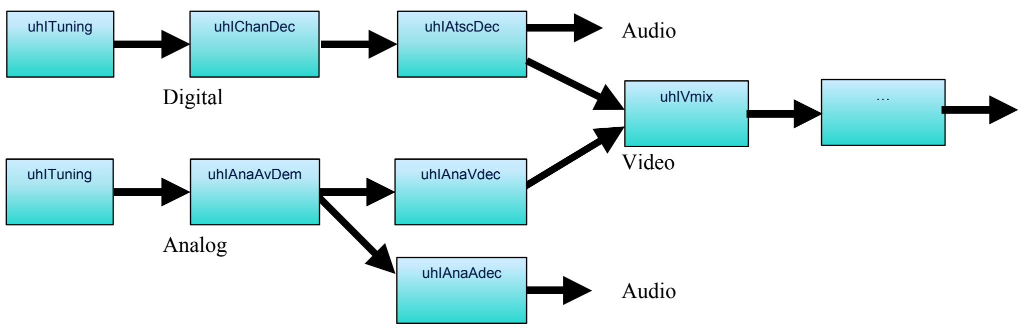

Figure 5 Simplified example graph.

Specification	Name	Description
uhIAmbientLevel	Ambient Level	Set the measured ambient light. This can then be used by the platform to optimise the video processing algorithms based on this.

UhIAnaVenc	Analog Video Encoder	The Analog Video Encoder gives control over the encoding of a digital video stream into an analog video signal.
uhIAnaVencrypt	Analog Video Encryption	The Analog Video Encryptor gives control over an anti-taping feature.
uhIAntiAging	Anti Aging	Control the algorithm used to prevent aging in case of static images being displayed (e.g. logo’s).
UhIBbarDet	Black Bar Detection	Detection of the position of the black bars in the video signal. These are often present if the transmission uses a different aspect ratio then the original recording.
UhIColEnh	Video Color Enhancement	Enhance the colors of the incoming video signal. Color enhancement deals with the color features Skin Tone Correction, Blue Stretch, and Green Enhancement.
UhICti	Color Transient Improvement	Allows control over the color transient improvement functionality (to increase the perceived color sharpness).
UhIDnr	Dynamic Noise Reduction	To control the reduction of the noise in a video stream.
uhIHistoMod	Histogram Modification	Histogram Modification can be used to improve the local contrast in a picture of which the distribution of grey levels is sub-optimal.
uhINoiseMeas	Noise Measurement	The Noise Measurement offers an abstract view on the amount of noise in a video signal.
uhIScanRateConv2	Scan Rate Conversion 2	Scan Rate Conversion provides the control on different algorithms to change the scan rate and or scan type of a signal (progressive, interlaced).
uhISharpEnh	Sharpness Enhancement	Sharpness Enhancement deals with enhancing the sharpness of the luminance (Y) component of a YUV signal.
uhISharpMeas	Sharpness Measurement	The Sharpness Measurement logical component has a YUV signal as input. It measures the sharpness of the Y signal (luma sharpness) and the sharpness of the UV signal (color sharpness).
UhIVfeat	Basic Video Featuring	Basic Video Featuring represents functionality to control the brightness, contrast, saturation, hue, and white point of a video signal.
UhIVmix	Video Mixing	Video Mixer represents functionality to compose a single video stream from multiple inputs, each of which can either be video or graphics.

Table 9 Description of the video Processing & Rendering.

In Figure 6 an example graph is shown based on the interface specifications listed in Table 9.

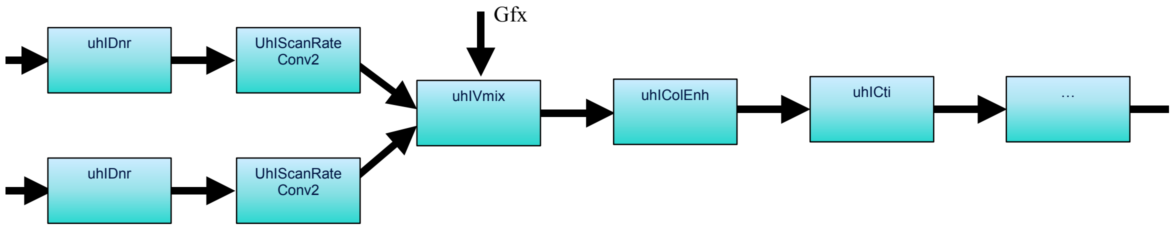

Figure 6 Possible Video processing & Rendering graph.

Specification	Name	Description
uhIDataInject	Data Injecting	The Data Injecting component allows the injection of streaming data into the platform. This can be used for a PVR application.
uhIDataExtract	Data Extracting	The Data Extracting component allows the extraction of streaming data from the platform. This can be used for a PVR application.
uhISptsTrMux	SPTS Transmuxing	Provides functionality to transmux an incoming Transport Stream to a SPTS, to insert sections in this SPTS and to notify clients of the locations of video frames in the SPTS output.

Table 10 Description of the personal Video Recording related.

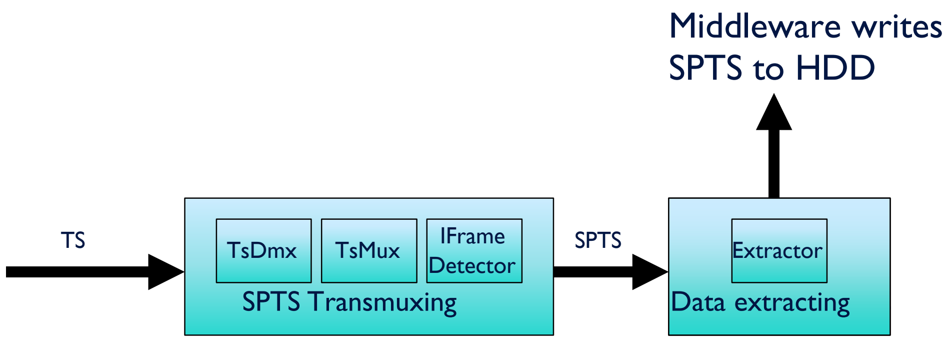

Figure 7 Extracting a Single Program Transport Stream for recording.

In Figure 7 a simplified context is given. Here the SPTS Transmux is used to reduce the bandwidth, by selecting the specific stream to be recorded, of the incoming stream. After this bandwidth reduction, the stream can be recorded to any device that is typically supported by Linux (e.g. a harddisk). Obviously in a real product there will be a tuner and channel decoder etc. in front of the SPTS Transmux from Table 10.

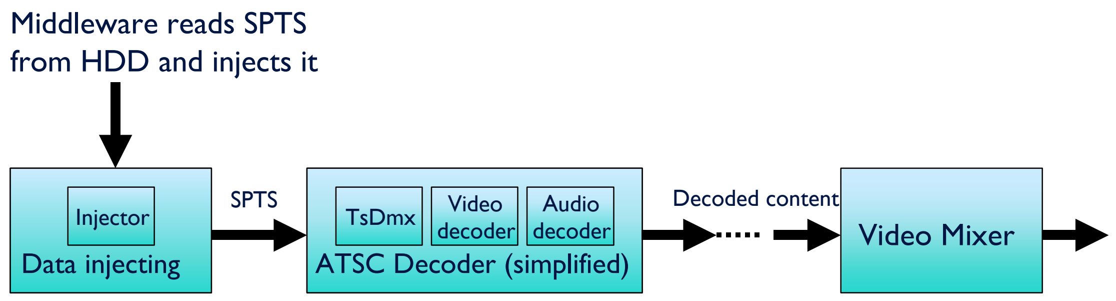

Figure 8 Injecting a Single Program Transport Steam for playback.

In Figure 8 a simplified context is shown for the Data Injector to show how to play back a recorded stream. The normal Atsc Decoder can be used to decode the recorded stream. The AtscDec provides the controls for the trickplay functionality (e.g. control playback speed). Obviously in a real product more components will be there like e.g. Video Processing and Rendering components (see Table 9).

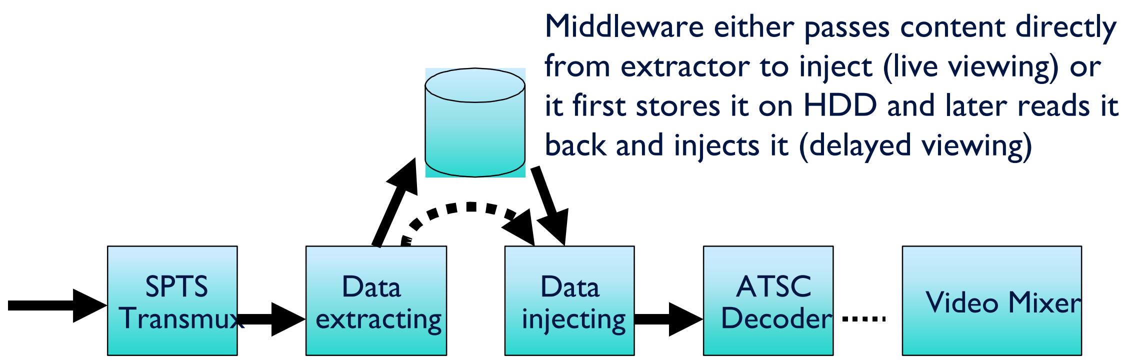
Middleware either passes content directly from extractor to inject (live viewing) or it first stores it on HDD and later reads it back and injects it (delayed viewing)
SPTS Transmux
Data extracting
Data injecting
ATSC Decoder
Video Mixer

Figure 9 Combining Data Extraction and Injection for delayed viewing.

In Figure 9 a simplified combined usage is shown from the SPTS Transmux, the Data Extractor and the Data Injector interface specifications. In this case a watch delayed video use case is given. A real product obviously contains more interface specification like e.g. a Tuner, Channel Decoder and Video Processing and Rendering components (see Table 9).

3.2.4 References

This section lists references to other standards or to implementations.

The official site of the UHAPI specifications:	www.uhapi.org
UHAPI specification version 1.1 version, donated by the UHAPI Forum to CELF	http://tree.celinuxforum.org/pubwiki/moin.cgi/UHAPI
Open Source implementation of UHAPI:	http://sourceforge.net/projects/uhapi4linux/
LinuxDVB specification:	www.linuxtv.org or directly http://www.linuxtv.org/downloads/linux-dvb-api-1.0.0.pdf
Video4Linux specification:	http://www.linuxtv.org/v4lwiki
Video4Linux 2 specification:	http://v4l2spec.bytesex.org/spec/

3.3 Graphics

3.3.1 Rationale

Given the widespread use of both DirectFB (especially in the TV domain), and standard FrameBuffer (in the PDA/Mobile domain), both are recommended. When graphics is combined with video, however, the DirectFB + UHAPI solution described above is preferred. The focus of DirectFB is on rendering and acceleration of 2D Gfx.

The focus of the UHAPI Video Mixing specification is on the composition of the Gfx and the video, and on scaling the video.

Given this focus difference it is specified that DirectFB is only controlling the graphics layers, and the video layers are to be controlled by the uhIVmix specification as depicted in Figure 4 in the previous section. IDirectFBScreen does not provide enough control on e.g. the scanrate and scantype conversion (use uhIScanRateConv2 instead to control the scan rate conversion, use uhIVfeat instead to control the brightness, white point etc.).
The capability on IDirectFBScreen to select output connectors does not support enough, and making connections or setting up graphs is more complex and HW specific. It must also be possible to setup the entire streaming graphs in one atomic action to avoid artefacts, uhIConnMgr is to be used for this.

OpenGL ES is the de-facto industry standard in the mobile domain and seems to be equally well applicable in the home domain for accelerating 3D graphics. The focus of OpenGL ES is on rendering and acceleration of 3D Gfx.

DirectFB, OpenGL and UHAPI combinations have been proven, and implementations exist. OpenGL ES is a subset of OpenGL.

The interfaces for Window Management and the Graphical User Interface are outside the CELF AVG specification. Known window managers can be used on top of DirectFB (e.g. X based solutions or Gtk+ which is implemented on top of DirectFB (http://www.directfb.org/index.php?path=Development%2FProjects%2FGTK%2B)).

3.3.2 Specification

[S] The standard Framebuffer is recommended for use in CE devices.

[S] DirectFB is recommended for use in CE devices.

[S] When graphics is combined with video, DirectFB in combination with UHAPI should be used. In this combination the video layers are controlled via the uhIVmix specification. The uhIVmix specification can expose one or more of the graphics layers to DirectFB for control via DirectFB.

[S] For control of 3D graphics OpenGL ES should be used.

Obviously, if the CELF 2.0 compliant product does not support or need 3D Gfx, the OpenGL ES part is not relevant for that specific product instance.

Note that not all of the available interfaces from DirectFB are part of this specification. When DirectFB is used, the following interfaces must [M] be used[1]:

• IDirectFB	Main interface
• IDirectFBSurface	Core graphics functionality
• IDirectFBFont	Font loading, metrics and measurements

[1] Note that IDirectFBEventBuffer and IDirectFBInputDevice are not mentioned here since they are not related to Graphics.

• IDirectFBDisplayLayer	Managing size, format, scaling, blending of layers Note that the IDirectFBDisplayLayer interface only operates on the layers that are controlled by DirectFB, not the video layers controlled by the uhIVmix specification.
• IDirectFBWindow	Multiple apps on one (graphics) layer
• IDirectFBPalette	Modify palette of surfaces with indexed format

[O] The following interfaces are optional for the layers controlled by DirectFB. This supports the cases where a dedicated decoder is not yet properly defined or where the middleware/application decodes the video itself (SW), and wants to render this to a IDirectFBSurface:

• IDirectFBImageProvider	Getting information about and loading one image from file.
• IDirectFBVideoProvider	Rendering video data by a software decoder in the middleware into a surface.
• IDirectFBDataBuffer	Streaming or static data for image or video providers.

[X] The following interface must not be used:

• IDirectFBScreen	Display encoder, output connectors. Note that EnumScreens on the IDirectFB interface will not list any available screens since the IDirectFBScreen interface is not supported.

[M] There are other restrictions to the use of DirectFB in the CELF API context:

- IDirectFBDisplayLayer
  - GetScreen does not return an interface since the IDirectFBScreen interface is not supported. It just returns DFB_UNSUPPORTED
  - GetSourceDescriptions must only return one source ID and name.
  - GetCurrentOutputField is not supported. This is not useful and incomplete. It just returns DFB_UNSUPPORTED
  - TestConfiguration, SetConfiguration is not supported for a Video Layer. It just returns DFB_UNSUPPORTED
  - SetFieldParity is not supported. It just returns DFB_UNSUPPORTED.
  - GetColorAdjustment, SetColorAdjustment is not supported. The Layers capabilities will reflect this. Use the uhIVfeat instead.

3.3.3 Notes

In Figure 10 in Section 3.3.4 the relation is shown between the different specification elements of the CELF 2.0 AVG API. In a next version of this CELF specification, the relation between DirectFB and an OpenGL ES implementation is to be further detailed (currently DirectFB only provides the IDirectFBGL interface).

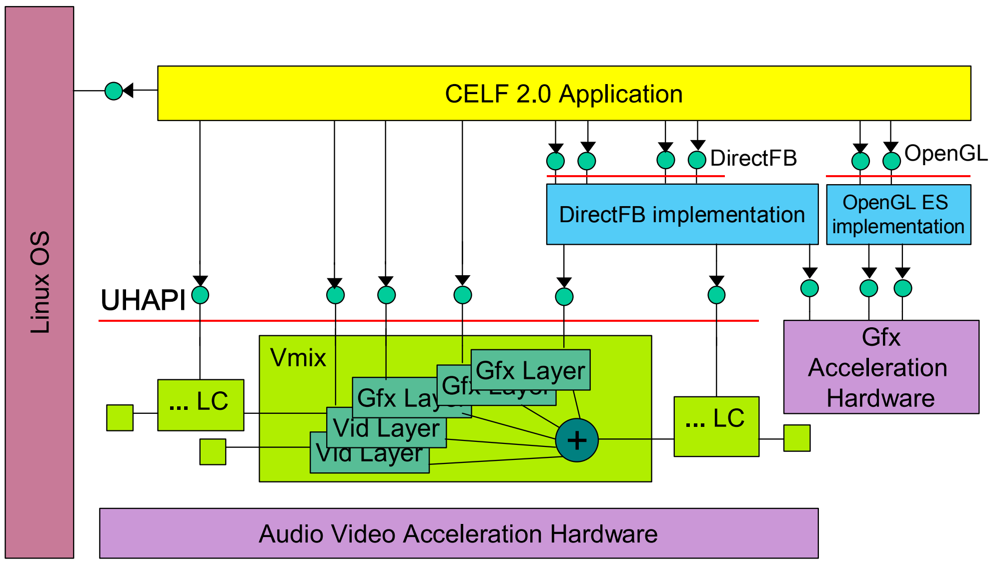

Figure 10 Relation of UHAPI, DirectFB and OpenGL ES interfaces.

3.3.4 References

This section lists references to other standards or to implementations.

The DirectFB specification:	www.directfb.org
The OpenGL ES specification:	http://www.khronos.org/opengles/spec/
The Graphics Tool Kit specification (or Gtk+):	http://www.gtk.org/

4. Work in progress

The following sections list ongoing and planned activities of the Audio Video Graphics working group.

4.1 Graphics

OpenGL and DirectFB combinations have been demonstrated in the PC domain. A number of member companies are currently implementing the combination of OpenGL ES and DirectFB. Results will be published on the CELF AVG Wiki pages.

4.2 Media Processing Frameworks

There are several interesting developments in the domain of media processing frameworks. The Khronos group is finishing the OpenMax specification, and the open source Gstreamer solution is being used in a number of CE devices or being investigated. A next version of the AVG specification will include a specification for media processing framework. For now, we only mention a number of considerations.

Media Processing Frameworks can be used to implement audio and video system functionality by creating graphs of audio/video processing components, codecs, sources and sinks. A media processing framework facilitates connecting components and takes care of communicating and buffering of data between components. The main function of media processing frameworks is this streaming functionality, also called ‘sidebar connections’. Codecs, source, sink and filter components have also functional control interfaces, e.g. to set the volume level or brightness. Since the CELF AVG specification recommends specific interfaces for this functional control in the previous sections, a media processing framework should match with these interfaces. A framework should not define control interfaces itself, or it should define lower-level interfaces on top of which the CELF interfaces can be realized.