

**Information Societies Technology (IST) Programme**

**Project N°: FP6-IST- 0028097**

# **ASTRALS**



---

## **Deliverable 3.5.1**

# ***SVC Encoding/Delivery Platform Description***

---

**Author(s):** K. Grüneberg, Th. Schierl (HHI);  
L. Celetto, E. Quacchio (STMicroelectronics)

**Status - Version:** Final

**Date:** 10 July 2007

**Distribution - Confidentiality:** Public

**Code:** ASTRALS\_D3.5.1\_HHI\_FF\_20070710

### **Abstract:**

This document describes deliverable D.3.5.2, the SVC Encoding and Delivery Platform, which is a software package.



## Disclaimer

This document contains material, which is the copyright of certain ASTRALS contractors, and may not be reproduced or copied without permission. All ASTRALS consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

The ASTRALS Consortium consists of the following companies:

No	Participant name	Participant short name	Country	Country
1	Algosystems SA	Algosystems	Co-ordinator	Greece
2	ST Microelectronics	ST	Contractor	Italy
3	Thomson	Thomson	Contractor	France
4	Mitsubishi Electric	Mitsubishi	Contractor	UK
5	Telefonica I + D	TID	Contractor	Spain
6	ProVision Communication Technologies	ProVision	Contractor	UK
7	Fraunhofer / Heinrich Hertz Institute	HHI	Contractor	Germany
8	Institute for Infocomm Research	I <sup>2</sup> R	Contractor	Singapore

The information in this document is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.



**This page has intentionally left blank**



## Table of contents

<b>Disclaimer</b> .....	<b>2</b>
<b>Abbreviations</b> .....	<b>5</b>
<b>Executive Summary</b> .....	<b>6</b>
<b>1. Basic Concepts</b> .....	<b>7</b>
1.1. <i>SVC Encoder as VLC Plug-in</i> .....	7
1.2. <i>Remote Control of the VLC Server</i> .....	7
1.3. <i>Policy Regarding Open Source Projects</i> .....	7
<b>2. Server implementation</b> .....	<b>8</b>
2.1. <i>Video on Demand Server Module</i> .....	11
2.2. <i>SVC Encoder Plug-in</i> .....	12
2.2.1. <i>Internal Interface between SVC Encoder Plug-in and VLC</i> .....	12
2.2.2. <i>SVC Encoder Configuration through Command Line Interface</i> .....	12
2.2.3. <i>SVC Encoder Setup through Configuration Files</i> .....	13
2.3. <i>Other library parts implemented within WP3</i> .....	15
2.3.1. <i>SVC File Format</i> .....	15
2.3.2. <i>Signalization</i> .....	16
2.3.3. <i>Layered Multicast</i> .....	16
2.4. <i>Basic and Extended Use Cases</i> .....	17
2.4.1. <i>File Input</i> .....	17
2.4.2. <i>Cascading</i> .....	18
2.4.3. <i>Live Camera Input</i> .....	18
<b>3. Interfaces with other WPs</b> .....	<b>19</b>
3.1. <i>The Transrater Plug-in Interface</i> .....	19
3.1.1. <i>The Transrater plug-in interface</i> .....	19
3.1.2. <i>The Transrater plug-in parameters</i> .....	22
3.1.3. <i>Configuration and Dynamic Adaptation of the Transrater</i> .....	22
<b>4. Results</b> .....	<b>25</b>
4.1. <i>Tested Features</i> .....	25
4.1.1. <i>SVC Encoder</i> .....	25
4.1.2. <i>Transrater Plug-in</i> .....	25
4.1.3. <i>Remote Control of the Transrater Module</i> .....	26
4.1.4. <i>Layered Multicast</i> .....	26
4.2. <i>SVC Encoder Performance</i> .....	27
<b>5. Conclusion</b> .....	<b>31</b>
<b>6. List of Software Releases</b> .....	<b>31</b>
<b>7. References</b> .....	<b>32</b>



## Abbreviations

<b>ASO</b>	<i>Arbitrary Slice Ordering</i>
<b>AU</b>	<i>Access Unit (all coded media data belonging to one time instant)</i>
<b>AVC</b>	<i>Advanced Video Coding [1] , in this context used for the single layer encoding, in contrast to the Scalable Extensions of H.264/MPEG4-AVC</i>
<b>CIF</b>	<i>Common image format, resolution 352 x 288 pixels.</i>
<b>FRExt</b>	<i>Fidelity Range Extensions (amendment to H.264/MPEG4-AVC)</i>
<b>IETF</b>	<i>Internet Engineering Task Force</i>
<b>JSVM</b>	<i>Joint Scalable Video Model of the JVT</i>
<b>JVT</b>	<i>Joint Video Team of ITU and MPEG</i>
<b>H.264</b>	<i>Video coding standard according to ITU-T Rec. H.264   ISO/IEC 14496-10</i>
<b>MPEG</b>	<i>Motion Picture Experts Group</i>
<b>NAL</b>	<i>Network Abstraction Layer (part of H.264/MPEG4-AVC standard)</i>
<b>QCIF</b>	<i>Quarter CIF, resolution 176 x 144 pixels</i>
<b>QP</b>	<i>Quantizing Parameter</i>
<b>RC</b>	<i>Remote Control</i>
<b>RFC</b>	<i>Request For Comments (IETF standard)</i>
<b>RTSP</b>	<i>Real-Time Streaming Protocol [5]</i>
<b>SVC</b>	<i>Scalable Video Coding, here always referring to the Scalable Extensions of H.264/MPEG4-AVC (ITU-T Rec. H.264   ISO/IEC 14496-10, Amendment 3)</i>
<b>VCL</b>	<i>Video Coding Layer</i>
<b>VLC</b>	<i>VideoLAN Client (Open Source media player with many additional functions)</i>
<b>VoD</b>	<i>Video on Demand</i>
<b>WP</b>	<i>Work package according to the ASTRALS project work plan</i>



## Executive Summary

A complete SVC streaming server has been built up of different components that may all run on the ASTRALS hardware platform, but can as well be split and run on different systems connected by an IP network (e.g. Ethernet). This is one of the first implementations with real-time encoding capability, and it supports temporal, spatial and SNR scalability, see section 2.1 of this document. The framework which has been used for the integration of all components is the VideoLAN Client (VLC), a highly customisable streaming server, which has been extended with SVC capabilities by the ASTRALS project partners in WP3 (cf. section 2). Although most of the plug-ins would be usable with different versions, the VLC release used for ASTRALS integration has been fixed to version 0.8.4a.

Another software package has been developed within WP3 which implements RTSP [5], the RTP session control protocol, see section 2.3.2. It supports a multi-session set-up, which means that a number of RTP sessions are opened, each used for the transport of a certain portion of the scalable video bit-stream.

A number of control interfaces are available for the streaming server. In particular, a remote control interface is used for the configuration of plug-ins developed by other work packages and which are integrated in the streaming system. The software implementation itself is ASTRALS Deliverable 3.5.2, which is available as executable code (binaries) for the restricted use within the project. This document describes its features and performance.

An SVC decoder client is being implemented within Task 4.4 of WP4. As the deliverables related to the decoder are due M20 (August 2007), the SVC encoding and delivery platform has been tested with preliminary versions of the SVC decoder.



## 1. Basic Concepts

### 1.1. SVC Encoder as VLC Plug-in

Software integration of different partners within a project is a very difficult task, provided that each partner has reasons to keep source code undisclosed which makes debugging a never-ending task. Furthermore, interface definitions between partners can consume a lot of time, and if any need for adaptation arises, partners will struggle to prevent modifications within their own code. This is the background on which we decided to use the open source VLC client/server software (version 0.8.4a) as a front-end to ASTRALS developments. VLC seems to be a nice platform for fast development and a good platform for Linux implementation. The code is in general well structured and besides that, it is also portable to Windows. The interfaces in the code are clean and allow extending the original implementation with reasonable effort. Function prototypes define the modalities used to exchange the code between different modules. For all these reasons, our developments can be easily integrated (via plug-ins for the main media players) into existing systems and on existing platforms. Thus, the ASTRALS partners could concentrate on their main task, which means research on and implementation of innovative algorithms.

### 1.2. Remote Control of the VLC Server

The VLC can be controlled through a network interface. This option is the most interesting as it allows us to remotely control the VLC from another application.

A simple function can implement the interface from other applications to the VLC. This function will open a communication channel using the TCP/IP protocols, so the calling applications should know the IP address and the port where the server will listen for incoming connections.

We implemented such functionalities using the standard Linux system calls. We open the connection using IP address and port name passed to the function. If the connection succeeds, we send the data to the channel and then we close the connection.

Through the telnet interface, we can send server configuration commands. The commands are fully described in section 2.1. These commands can be used to add or delete a new multimedia streaming session, but can not be used to change parameters in one already streaming session.

Through the remote control (RC) interface, we can change parameters of the server dynamically during the streaming session. New commands may be used to address a specific plug-in and sending new configuration values at any moment. We described the implementation in more details in section 3.1.3.

### 1.3. Policy Regarding Open Source Projects

The internal license policies of the project partners do not authorize the release of any software developed within ASTRALS as Open Source under any open license as e.g. GPL (valid for VLC source code). According to the aforementioned policies external dynamic link libraries are used, which are only loaded during runtime and are only loaded if requested by user interaction. This dynamically loaded code is not covered by the GPL, but the plug-in code for the VLC calling the external library.

## 2. Server implementation

The server we use for the integration of all related ASTRALS software developments is the VideoLAN Client (VLC) software project, to which an SVC encoder plug-in has been implemented by HHI. Figure 1 gives an overview of the VLC server and the connected plug-ins. STMicroelectronics has implemented RTP and RTSP modules for the VLC server. A matching SVC decoder plug-in will be delivered by WP4. In sake of compatibility and a smooth final integration of the concurrent developments, we use the VLC defined interfaces as defined in the VideoLAN project. Already existing features of the VLC software can be used as far as possible.

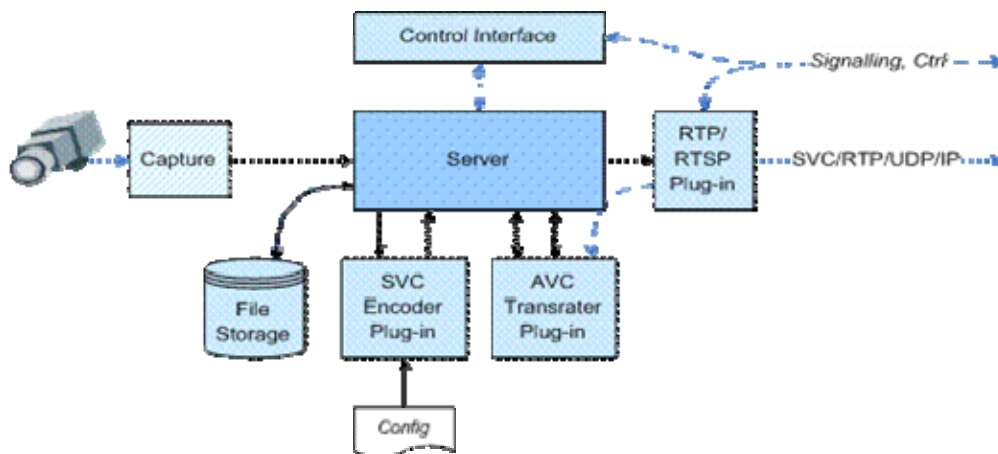


Figure 1: ASTRALS modules connected to the VLC server

In this section we provide a short description of the interface between the encoder and the server. VLC has a specific function prototype that is used to encode a single frame and return the corresponding bit stream.

As an example, a complying function call can be:

```
block_t *Encode( encoder_t *p_enc, picture_t *p_pict ) ;
```

where `block_t` contains the encoded bit stream (cf. Table 1), `p_enc` contains the internal status of the encoder (cf. Table 2), while `p_pict` contains the picture to be encoded (cf. Table 3).

Table 1: The `block_t` structure

Type	Field name	Description
block_t	p_next	Pointer to the next block structure in a double-linked list
block_t	p_prev	Pointer to the previous block structure in a double-linked list
uint32_t	i_flags	Attributes of the current block, regarding its contents (I,P,B frame bit stream, etc.)
mtime_t	i_pts	The Program TimeStamp for audio/video synchronization
mtime_t	i_dts	The Display TimeStamp for audio/video synchronization, to be used in alternative to the previous field
int	i_samples	The audio samples contained in current bit stream and available when data is decoded
int	i_rate	
int	i_buffer	The buffer size in bytes



<i>Type</i>	<i>Field name</i>	<i>Description</i>
uint8_t *	p_buffer	The pointer to the buffer containing the bit stream and holding i_bytes data.
void	(*pf_release)	The pointer to the function that de-allocates the previous buffer
vlc_object_t *	p_manager	The VLC object manages the blocks reducing memory allocations/de-allocations
block_sys_t *	p_sys	Contains proprietary data of the block_t structure. Not used, and not to be used according to the comments in the code.

Table 2: The encoder\_t structure

<i>Type</i>	<i>Field Name</i>	<i>Description</i>
module_t *	p_module	Contains some basic information about module capabilities, to be used to choose the right functionality dynamically
encoder_sys_t *	p_sys	Contains the internal status of the encoder (its settings and variables).
es_format_t *	fmt_in	Contains the incoming data format and may be used to exchange some preliminary information on the following modules in the chain by means of the i_extra and p_extra fields.
es_format_t *	fmt_out	Contains the format of the outgoing data for reference to the following modules in the chain. This means that fmt_out will become the fmt_in of the following module in the chain.
block_t *	(*pf_encode_video)	This function pointer contains the address of the called function and is used to encode one frame. The resulting bit stream will be delivered by means of the returning block_t * pointer.
block_t *	(*pf_encode_audio)	This function pointer contains the address of the called function and is used to encode one audio data chunk.
block_t *	(*pf_encode_sub)	This function pointer is used to encode the sub-titles to the current fram/scene.
int	i_threads	This contains the number of threads to be used in the current encoding (useful for multi-processor/multi-threading architectures)
int	i_iframes	Contains reference to I frames, but not used throughout the code.
int	i_bframes	Contains the number of B frames to be used when encoding the GOP.
int	i_tolerance	The tolerance in bit rate between the fixed value and the obtained one
sout_cfg_t	p_cfg	Pointer to the configuration of the encoder.



Table 3: The picture\_t structure

<i>Type</i>	<i>Field Name</i>	<i>Description</i>
video_frame_format_t	Format	The properties of the current picture
uint8_t *	p_data	Not to be touched, contains the address to the memory location of the frame-buffer
void*	p_data_orig	Pointer before memory alignment, preserved for proper memory de-allocation
plane_t	p[ ]	Contains the addresses to the different planes of the image
int	i_planes	Contains the number of planes for current frame
int	i_status	Picture status signals (internally) if the picture is ready to be displayed, or reserved, etc.
int	i_type	Contains information if there is direct rendering etc.
vlc_bool_t	b_slow	Contains information if the picture is in a slow memory
int	i_matrix_coefficients	Contains the YUV encoding type for current frame
int	i_refcount	Contains references for current picture (not to be accessed directly, but through APIs)
mtime_t	Date	Display date for current image (i.e. the temporal instant when the image must be displayed)
vlc_bool_t	b_force	Force displaying current image, even if it is in the past (i.e. the decoder is not fast enough to display in proper time). Using this flag may broke audio/video synchronization.
vlc_bool_t	b_progressive	Signals if the image is a progressive or interlaced image
unsigned int	i_nb_fields	Number of displayed fields for current image
vlc_bool_t	b_top_field_first	Signals if the top field or the bottom one is the first to be decoded
picture_heap_t *	p_heap	The heap this image is attached to. All the pictures are stored in a heap in order to maintain them accessible and allocated when they are already displayed.
int	(*pf_lock)	Allows picture buffer locking before modifying memory areas
int	(*pf_unlock)	Unlocks previously allocated pictures
picture_sys_t *	p_sys	Private data of the current video frame
void	(*pf_release)	Releases the memory areas for the current frame
struct picture_t *	p_next	Pointer that references the next buffer in the FIFO queue of pictures



## 2.1. Video on Demand Server Module

The VLM module allows configuring the VLC as a VoD server for streaming of multiple multimedia sessions in parallel, given the user requirements. The VLM module may receive commands through the telnet interface, allowing a dynamic configuration of the server that allows inserting new multimedia contents and deleting old ones.

VoD session parameters include specifications of input, output and configuration options. Input may come from any input device; hence even a V4L-enabled camera may be used as an input. Output specifies the plug-in used to save the input contents. Different output devices may be files, where you can store the contents, transcoders, which allows compressing the input contents in a specific format or network output devices that can be connected to the transcoders in order to send the content over the network. The output device may be configured through option parameters that will be passed to the plug-in when it is opened.

The syntax of the VLM commands is given in the following<sup>1</sup>

- help : Displays an exhaustive command lines list
- new (name) vod|broadcast|schedule [properties] : Create a new vod, broadcast or schedule element. Element names must be unique and cannot be "media" or "schedule". You can specify properties in this command line or later on by using the setup command.
- setup (name) (properties) : Set an element's property.
- show [(name)|media|schedule] : Display current element states and configurations.
- show (name) : Specify an element's name to show all information concerning this element.
- show media : displays a summary of media states.
- show schedule : displays a summary of schedule states.
- del (name)|all|media|schedule : Delete an element or a group of elements. If the element wasn't stopped, it is first stopped before being deleted.
- del (name) : Delete the (name) element.
- del all : Delete all elements
- del media : Delete all media elements.
- del schedule : Delete all schedule elements
- control (name) [instance\_name] (command) : Change the state of the (instance\_name) instance of the (name) media. If (instance\_name) isn't specified, the control command affects the default instance. See Control Commands for available control commands.
- save (config\_file) : Save all media and schedule configurations in the specified config file. The config file path is relative to the directory in which vlc was launched. If the file exists it will be overwritten. Note that states, such as playing, paused or stop, are not saved. See Configuration Files for more info.
- load (config\_file) : Load a configuration file. The config file path is relative to the directory in which vlc was launched. See Configuration Files for more info.

The commands may be entered as a configuration file on the command-line at the starting of the VLC, during the execution of the VLC through the text interface, or through the network via the telnet interface.

---

<sup>1</sup> The commands are taken from URL <http://www.videolan.org/doc/streaming-howto/en/ch05.html>



## 2.2. SVC Encoder Plug-in

### 2.2.1. Internal Interface between SVC Encoder Plug-in and VLC

In order to ease the streaming process, the VLC encoder plug-in can instruct the remaining VLC modules about some characteristics of the produced bit stream. In this section, we describe the plug-in characteristics in order to easily and efficiently interface the encoder with the remaining modules in the streaming server.

One of the requirements of the RFC3984 (and the amendments for scalability [6] now under discussion) is the transmission of the SPS and PPS parameters by means of reliable channels. VLC allows sharing some preliminary data by means of the `es_format_t` structure. Each module has an incoming `es_format_t` structure, named `fmt_in`, that contains information about incoming data, and delivers to the next module in the chain a `fmt_out` structure contains information about the outgoing data. The `es_format_t` structure contains two fields – a `p_extra` char pointer and an `i_extra` integer – that may be used to convey some extra information about the format. We use them to convey information to the `rtp.c` module about the SPS and PPS data, in the same format as they would be placed in the AVC Configuration Box for the File Format. The `rtp.c` module will receive the information from the `fmt_in` structure and will format it in an appropriate way (i.e. through the `sprop-parameter-sets=`), in order to deliver it by means of the RTSP protocol.

The NAL units will be delivered to the streaming server appropriately formatted for ease the sender's task. For that reason, the encoder will output in the `block_t` structure the bit stream relative to a single frame it encoded at the highest temporal, spatial and quality layer. Hence the `block_t` will contain in general one or more NAL Units. To ease the sender task, the encoder will save the NAL units in the same format they are saved in the chunk of data saved in the file format. Hence, some initial bytes will be pre-pended to all the NAL units in the `block_t`, containing information about the size in bytes of the current NAL unit. The number of initial bytes is contained in a `avcC_length_size` field that is delivered to the `rtp.c` module initially by means of the AVC Configuration Box data delivered by means of the `p_extra` field. In ASTRALS, a 4-byte length field will always precede each NAL unit.

The encoding parameters are read by the plug-in from a configuration file. Path and filename will be passed by a command line parameter `--sout-hhi4-cfg-File=file://<filepath>/<filename>`.

A configuration file template is found below; some more are included in the software package (which is deliverable D3.5.2).

### 2.2.2. SVC Encoder Configuration through Command Line Interface

The server is started by a configuration script which defines one or more setups that can be called by a client through RTSP. A sample call of the VoD configuration file looks as follows:

```
./vlc -vvv -l telnet --telnet-password videolan --rtsp-host 0.0.0.0:5554 --vlm-conf  
mediaset_vod_spatial.cfg
```

The parameters of this call are explained below:

- l telnet: use telnet as command interface for the server
- telnet-password videolan: provide password (example: videolan) for the telnet connection
- rtsp-host 0.0.0.0:5554: IP      Address and port a client has to connect to  
(IP address 0.0.0.0 is automatically replaced by the default  
network address of the host)
- vlm-conf mediaset\_vod\_spatial.cfg: A configuration file that defines all other settings.

An example configuration file is shown below.



Example VOD configuration file (mediaset\_vod\_spatial.cfg):

```
new mediaset vod
  setup mediaset input
/home/astrals/vlc/rel_astrals_02052007/install/sequences/test_CIF6Hz.avi
  setup mediaset output #transcode{vcodec=h264,vb=384,scale=1}
  setup mediaset option venc=hhi_h264svcenc_svc
  setup mediaset option sout-HHI4-cfg-File=mediasetQCIF6Hz.cfg
  setup mediaset option sout-rtp-port-video=1232
  setup mediaset option sout-udp-caching=800
  setup mediaset option sout-rtp-sdp=file:./mediaset.sdp
  setup mediaset option mtu=1472
  setup mediaset option file-caching=3000
  setup mediaset option loop
  setup mediaset option noaudio

  setup mediaset enabled
```

Basic parameters to be set within the VoD configuration file are (see also chapter 2.1):

- Codec to be used (in the example: option `venc=hhi_h264svcenc_svc`)
- Codec configuration file (option `sout-HHI4-cfg-File=mediasetQCIF6Hz.cfg`, more details see below)
- Port number to be used (option `sout-rtp-port-video=1232`)
- Cache size(s) (option `sout-udp-caching=800 [ms]`)
- SDP file name (option `sout-rtp-sdp=file:./mediaset.sdp`)
- Play once or loop (option `loop`)

### 2.2.3. SVC Encoder Setup through Configuration Files

The SVC encoder is configured by means of configuration files which are read by the encoder kernel once the encoder is started because changes of any configuration parameter would influence the internal state of the encoder in a way that it has to be restarted anyway. Besides that limitation, we regard the flexible usage and adaptation of a bit-stream without any need to interfere with the encoder as a main advantage of Scalable Video Coding. Nevertheless, different setups can be chosen according to the use cases. A number of configuration files can be prepared which activate different numbers of layers, scalable spatial resolution or frame rate, and different image quality.

The syntax and semantics of the configuration file is similar to the JSVM reference software in order to facilitate testing of the real-time plug-in against a proven offline encoder. That is why some redundant parameters are included in the examples below which are ignored by the real-time plug-in but would be needed for offline tests and interpreted by the JSVM software. Parameters that are valid for all layers are given in a "main" configuration file.

Main configuration file, referring to configuration files for each layer:

```
# Main Configuration File

##### GENERAL #####
OutputFile      mediaset_hhi.264 # Bitstream file (ignored by the plug-in)
FrameRate       6.25           # Maximum frame rate [Hz]
FramesToBeEncoded 1500         # Number of frames (at input frame rate)
#               # (ignored by the plug-in)

##### MCTF #####
```



```

GOPSize          8          # GOP Size (at maximum frame rate)
IntraPeriod      16         # Intra Period
NumberReferenceFrames 1     # Number of reference pictures
BaseLayerMode    1          # Base layer mode (0:Scalable,
                          # 1:AVC compatible, 2:AVC w subseq SEI)

##### MOTION SEARCH #####
SearchMode       4          # Search mode (0:BlockSearch, 4:FastSearch)
SearchFuncFullPel 3         # Search function full pel
                          # (0:SAD, 1:SSE, 2:HADAMARD, 3:SAD-YUV)
SearchFuncSubPel 2         # Search function sub pel
                          # (0:SAD, 1:SSE, 2:HADAMARD)
SearchRange      32         # Search range (Full Pel)
BiPredIter       0          # Max iterations for bi-pred search
IterSearchRange  8          # Search range for iterations (0: normal)

##### LOOP FILTER #####
LoopFilterDisable 0         # Loop filter idc (0: on, 1: off, 2:
                          # on except for slice boundaries)
LoopFilterAlphaC0Offset 0   # AlphaOffset(-6..+6): valid range
LoopFilterBetaOffset 0     # BetaOffset (-6..+6): valid range

##### LAYER DEFINITION #####
NumLayers        2          # Number of layers
LayerCfg         avc_mediaset_QCIF_12.5hz_qp36.cfg # Layer conf file
LayerCfg         svc_mediaset_QCIF_12.5hz_qp30.cfg # Layer conf file

##### ADAPTIVE GOP STRUCTURE #####
UseAdaptiveGOP   0          # Use adaptive GOP structure (0: no, 1: yes)
AGSMODEDecision  0          # Mode decision (0: AGS Coding, 1: Mode Dec)
AGSGOPModeFile   ags_mode.dat # GOP mode file
UseRealtimeConstraints 1    # Restrict encoder proc time (0:off, 1:on)
RapPeriod        2          # eff RAP period = RapPeriod * IntraPeriod
RapScalableSei   true       # Whether to emit a scalable SEI at RAPs
RapSpsPps        true       # Whether to emit SPS and PPS at RAPs
  
```

Basic parameters to be set within the main SVC encoder configuration file are:

- FrameRate (in the example: 6.25 Hz)
- GOPSize (example: 8 frames)
- NumLayers (example: 2 layers)

Distinct parameters for each single layer are given in a layer configuration file as shown below.

Example layer configuration file (avc\_mediaset\_QCIF\_12.5hz\_qp36.cfg):

```

# Layer Configuration File

##### INPUT / OUTPUT #####
InputFile        mediaset_qcif12.5.yuv # Input file
ReconFile        rec.yuv # Reconstructed file
SourceWidth      176 # Input frame width
SourceHeight     144 # Input frame height
FrameRateIn      6.25 # Input frame rate [Hz]
FrameRateOut     6.25 # Output frame rate [Hz]
SymbolMode       1 # 0=CAVLC, 1=CABAC
  
```



UpdateStep	0	# Update Step for MCTF
ClosedLoop	1	# Closed-loop ctl (0,1:at highest rate, # 2: at lowest and highest rate)
#===== DECOMPOSITION (MCTF) =====		
UpdateStep	0	# Update Step (0: no 1: yes)
AdaptiveQP	0	# QP selection (0: standard, 1: adaptive)
UseIntra	1	# Intra mode usage (0: off, 1: on)
FRExt	0	# FREXT mode (0:off, 1:on)
#===== CODING (MCTF) =====		
#MaxDeltaQP	1	# Max. absolute delta QP
QP	36.0	# Quantization parameters
NumFGSLayers	0	# Number of FGS layers ( 1 layer -> DQP = 6 )
InterLayerPred	0	# Inter-layer Prediction (0: no, 1: yes, 2:adaptive)
BaseQuality	3	# Base qual lvl (0, 1, 2, 3) (0: no, 3, all)
DecodeLoops	0	# Dec loops (0:single, 1:mult for LP, 2:mult)

Basic parameters to be set within layer configuration files are:

- SourceWidth and SourceHeight (in the example: QCIF)
- FrameRateIn (example: 6.25 frames/s)
- Quantizer Parameter QP (example: 36)
- InterLayerPrediction (example: no)

We have defined some practical parameter sets, as the demand for processing power will be influenced dramatically by some features. We reach about 6 frames per second, spatial scalability between QCIF and CIF, and one quality enhancement layer.

## 2.3. Other library parts implemented within WP3

### 2.3.1. SVC File Format

Support for the ISO Base Media File Format [3] and its extension for AVC [4] had already been available, but not for SVC. HHI has implemented a library for SVC media file creation and access. An application has been made available to ST so that SVC files conformant to the most recent drafts of the AVC File Format standard can be used by the VLC server and SVC decoders.

In a first step, we used a basic implementation which can add an SVC track containing the whole SVC bit stream. When we read a bit stream generated by that application, only complete Access Units (AU) can be accessed. The server uses very similar interfaces to read from file or to receive data from the encoder plug-in, because in both cases it needs to parse a complete AU containing all layers of the scalable bit stream. It is also capable of adding “hint” tracks in a way that the media can be streamed directly (e.g. by the Darwin Streaming Server).

The first basic implementation of the media file format tool and an update supporting Layered Multicast through multiple hint tracks are available on the FTP site as Linux application. Both have been compiled under Windows as well.



### **2.3.2. Signalization**

From the decoder side, RTSP [5] protocols are fully interoperable with Internet streaming servers. A VLC server - VLC client interoperability test works properly.

As we are taking in consideration a scenario where there is a uni-directional flow of data, we can also use RTP/UDP and announcement as specified in "8.2.3 Usage in Declarative Session Description" where SDP over RTSP announces the session parameters (no Offer/Answer handshake), RFC 3984. Please note that RFC 3984 also recommends using an out-of-band transport for SPS and PPS, even if in-band SPS and PPS transmission is possible (but not recommended, as UDP packets may be lost, resulting in an undecodable sequence).

The implemented SDP session description contains the required information for streaming unicast and multicast sessions. The SDP text contains instruction about the packetization schemes and the SPS and PPS of the H.264/SVC sequence. The following paragraphs will give a detailed description of the implementations in the server.

About the unicast session management implementation, we based our solution according to the description provided in section 2.2.1 of this document. We used the p\_extra field in the fmt structure to deliver to the RTP plug-in the ConfigurationRecord as read from the FileFormat. A function called RtspConfigureH264Decoder() uses this information to generate a base64 encoded string containing all the data for the configuration, to be exported to the client using the sprop-parameter-sets= option.

About the layered multicasting implementation, we used a similar mechanism to handle the communication between the different modules. We used the p\_extra field to deliver the information contained in the SDP box to the RTSP module, in order to compose the SDP text. In the following paragraphs we will explain the details of the implementation for the MPEG-4 File Format plug-in and to the RTP plug-in in order to compose a SDP session description from the information coming from the File format.

We implemented the decoding of the Hint track from scratch. The original behavior of the File Format plug-in is to decode the media tracks and send their contents to the following modules, ignoring hint tracks. We added the --mp4-rtp-hint option to the File Format plug-in. When this option is present on the configuration of the plug-in, the original behavior is overridden in favor of the hint track decoding. Only the hint tracks are considered, while the tracks without hinting information are disabled. The contents of the SDP box we are considering is decoded and sent to the RTP module for the composition of the final SDP session descriptor. The plug-in uses a fourcc identifier 'rtp' that distinguishes normal decoders from hint tracks decoding. In this way other modules will know that the information they are receiving is coming from the hint track and will behave accordingly.

The RTP plug-in will receive the information coming from the File Format and compose the final SDP. The SDP information coming from the File Format is saved in the id->psz\_fmtp field for each streamed hint track and then used in the SDPGenerateHinted() function that creates the final SDP.

Finally we tested this solution using SAP session announcements. Regarding the source code of the SAP announcer, we made a small bug correction, removing an erroneous "#else" statement from line 450 of file src/stream\_output/sap.c.

### **2.3.3. Layered Multicast**

In IETF, there seems to be a great interest for supporting layered-multicast, i.e. splitting a scalable multimedia compressed bitstream over several parallel RTP sessions. ASTRALS WP3 proposes to use layered multicast so that different devices can join more or less RTP sessions according to their capabilities. Additionally, we would like to configure the wireless interface so that different ports at the same IP address can be served with different modulation. Thus, there will be some graceful degradation of quality in case the wireless connection gets worse due to increasing C/N (higher attenuation, bigger distance).

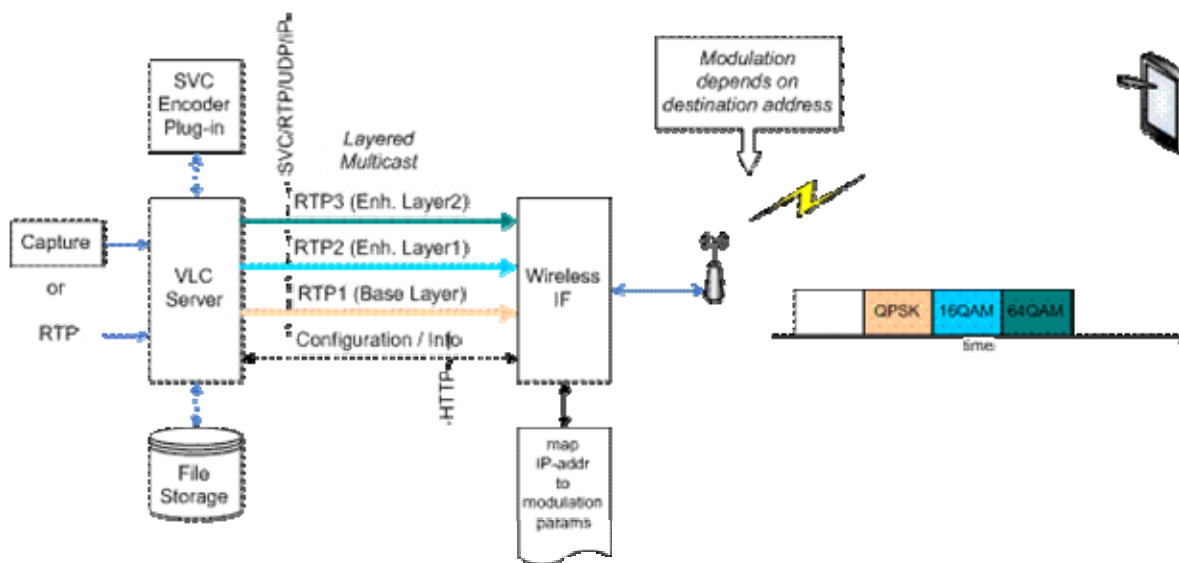


Figure 2: Interface between VLC Server and Wireless Interface for Layered Multicast

The layered multicast implementation is based on packets formatted by the MPEG-4 File Format hint tracks. The solution alleviates the server from parsing the bit-stream to put the different NAL units on different multicast sessions. The external file format tool can parse the bit-stream and generate hint tracks that give information on how to create the RTP packets without the constraint of performing the task in real-time and reducing the load of the server.

The MPEG-4 File Format tool produces the RTP packet payload. The file format gives also information about the value assumed by some of the RTP header fields as the P, X, M bits the timestamp relative values and the RTP sequence number values. Therefore, implementation of the function `rtp_packetize_hint()` – that streams the packets over the network – is rather simple. The function extracts the information coming from the file format from the first 12 bytes of the packet and assembles it according to the RTP specifications. Hence it creates the RTP header and copies the payload directly from the incoming buffer.

## 2.4. Basic and Extended Use Cases

We had to find a balance between interesting use cases and feasibility in terms of processing power or software features. Extended use cases can be envisioned but may not be available before the exploitation phase of the project. As long as we use SVC, the encoder should always generate several layers, and either the network restricts or the client selects the number of layers used.

In principle, two different ways of adaptation are possible:

- Adaptation of image resolution, frame rate and image quality according to the users needs.
- Automatic adaptation of the bit rate to some given requirements of the network which could be done by a media aware network element (MANE). There has been no manpower assigned to such a task.

The basic use case will be handled by starting VLC from command-line with appropriate parameters.

### 2.4.1. File Input

File input will be used in several use cases including demonstration of advanced SVC features such as real-time decoding of 4CIF (TV format) video, which would be too demanding in terms of real-time encoding with limited processing power, and for the erosion storage feature.

### 2.4.2. Cascading

It might be interesting to run a (real-time) encoder on a high-end PC. In this case we need appropriate network architecture to support cascaded servers. The first one is sending the encoded video using a simple codec (it might as well be a camera with its own network interface). The second server would be trans-coding that stream using SVC (cf. section 2.2.2).

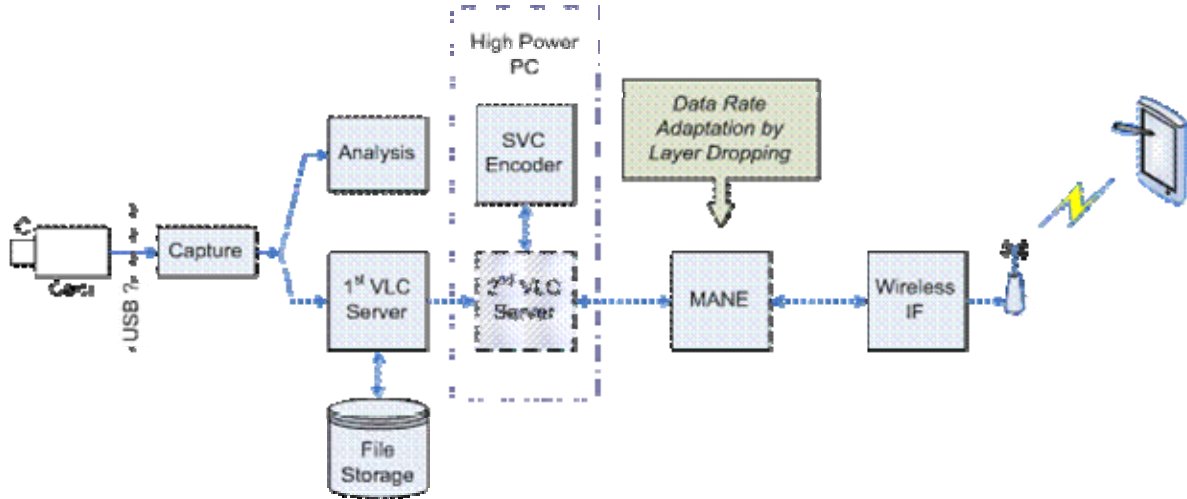


Figure 3: Cascading VLC

### 2.4.3. Live Camera Input

The basic idea behind this use case is to show a video surveillance set-up that makes use of some SVC features. We can imagine the use of temporal scalability (low frame rate until something happens) or spatial scalability (high resolution images if something has happened).

The server should be able to record the complete bit stream into a file, preferably while it is streamed to the network.

It has to be tested whether the same input device can be used in parallel by the event detection (WP6, cf. D6.1) and still be streamed directly. We assume that either event detection or streaming system can connect to the camera device driver.



### 3. Interfaces with other WPs

In this section we analyze the possible use-cases of the VLC inside the ASTRALS project and how it can be dynamically configured by other applications running on the system. We made several tests to prove that the VLC may be easily configured in order to insert new contents, dynamically configure the plug-ins it uses also from the external devices. It is possible to initiate new VoD session, specifying parameters for the involved plug-ins and this make possible to configure them at any time, just giving a new set of initial parameters. Using a network interface, it is also possible to specify new commands in order to change some selected parameters on-the-fly during the streaming session.

An overview of the VLC server and the connected plug-ins is given by Figure 1. Live contents will be captured by a video camera, which is a task that is dealt with in WP6. Thanks to the V4L plug-in of the VLC, the captured data are directly available to the SVC video encoder. Transrating is done by a Transrater plug-in developed in WP4. Only the server itself and the SVC encoder are integral parts of this workpackage. Though, interfacing with all plug-ins is part of the server implementation.

The SVC video encoder reads its parameters on the opening phase, so it is not possible to modify them during the streaming session. However, the VLC allows closing and opening streaming session multiple times without shutting down the server completely. Therefore it is possible to close the SVC encoder plug-in, change its configuration parameters and restart it with the new sets of parameters without rebooting the VLC server completely. A similar approach may be used for the AVC transrater. In this case, some of the parameters may be changed by means of VLC internal variables. The QP setting parameter will be changed on-the-fly during the streaming session, without needing to close the AVC transrater and start a new streaming session.

The described approach fits all the necessities we foresee for the ASTRALS project. In fact the big advantage of the SVC approach is the possibility of adapting the quality and bandwidth just dropping layers. Therefore, we do not see the necessity of a dynamic re-configuration of the server on-the-fly. However, having the possibility of adapting the parameters without rebooting the whole VLC is a desired feature. For the AVC transrater instead, a dynamic QP configuration is a desired feature, as network bandwidth fluctuation may occur and we have no other means of adapting the bandwidth, but changing the QP parameter. In this case, even restarting the plug-in would not be a desired feature because it would mean quitting a streaming session and starting a new one, which would not be very appealing from the end-user point of view. The following section describes how these features can be achieved.

#### 3.1. The Transrater Plug-in Interface

ASTRALS also integrates a transrater in the VLC server. Transcoding and transrating services were originally supported by the VLC code. The original VLC idea was to fully decode one image or sound samples and then re-encode them in another (or the same) format, maintaining proper synchronization. In this solution, we can transcode sound and frames from any supported format (for decoding) to any supported format (for encoding) or adapt the bitrate maintaining the same coding format. However, shortcuts in transcoding may allow handling this process with less computational resources.

In the following sections we describe the details of the implementation of the Provision transrating plug-in. Section 3.1.1 explains how this plug-in interfaces the VLC, section 3.1.2 explains which parameters can be changed and finally section 3.1.3 explains how the former parameters can be changed during the streaming session.

##### 3.1.1. The Transrater plug-in interface

In this section we describe the Provision transrater plug-in. The VLC software architecture is very flexible and allows inserting transcoding and transrating plug-ins as additional modules in the



original structure without modifications. Some examples are already present in the original code, as form of C code in modules modules/stream\_out/transrate/transrate.c and modules/stream\_out/transcode.c. Basically, the modules are added to the VLC as “stream out” objects. Provision inserted a new transrating plug-in named pv.c that was added to the VLC folder modules/stream\_out/transrate/. We describe the plug-in in the following.

The Open( ) and Close( ) functions will handle the memory and data initialization and de-allocation. The Add( ) function will handle the request of transrating the bitstream, queuing the request in the module, so that it is possible to transrate multiple bitstream at the same time:

```
sout_stream_id_t *Add( sout_stream_t *p_stream, es_format_t *p_fmt )
```

The returned sout\_stream\_id\_t structure contains a handler for the current stream (basically, it is an opaque pointer containing the handler of the current stream), while sout\_stream\_t \*p\_stream structure contains the handler of the incoming stream (see Table 4). The es\_format\_t \*p\_fmt structure contains the format of the incoming audio/video (see Table 5), while the outgoing audio/video format is determined according to the parameters passed during the Open( ) function.

Table 4: The sout\_stream\_t structure

<i>Type</i>	<i>Field Name</i>	<i>Description</i>
module_t *	p_module	Contains the basic information for dynamically choose the right module. The user should not touch this field
sout_instance_t *	p_sout	Contains information about the outgoing stream, for proper locking and handling
char *	psz_access	Name of the access for delivering transcoded data. A module_Need() function call will find the proper available module
sout_cfg_t *	p_cfg	The configuration structure for the module.
char *	psz_name	Name for current module, to be used to handle parsing of parameters from command line, etc
sout_access_out_sys_t *	p_sys	Contains the internal variables for current module
int	(*pf_seek)	Pointer to function to seek in the current module
Int	(*pf_read)	Pointer to function to read in the current module
Int	(*pf_write)	Pointer to function to write in the current module

Table 5: The es\_format\_t structure

<i>Type</i>	<i>Field Name</i>	<i>Description</i>
Int	i_cat	The category for current Elementary Stream (could be video, audio, etc)
vlc_fourcc_t	i_codec	The “four character” identifier for the current codec
int	i_id	-1 is invalid (to be marked), >=0 valid identification number for the stream
int	i_group	-1 is stand-alone, >=0 is part of a group (program)
int	i_priority	-2 non selectable by users, -1 not selected by default, >=0 priority of the stream



<i>Type</i>	<i>Field Name</i>	<i>Description</i>
char *	psz_language	The language for current stream (may not be set)
char *	psz_description	A description for current stream (may not be set)
audio_format_t *	Audio	Contains the format of current audio data
video_format_t *	Video	Contains the format of current video data
subs_format_t *	Subs	Contains the format for the sub-titles of current ES
unsigned int	Bitrate	The bit rate for the current stream
Vlc_bool_t	b_packetized	Signals if the data is packetized properly or need a packetizer
Int	i_extra	The number of bytes for current auxiliary information for properly initialising the module
void *	p_extra	The pointer to auxiliary information for proper initialization

The `transrate_video_process()` is the main function called by the plugin:

```
int transrate_video_process( sout_stream_t *p_stream, sout_stream_id_t *id,
    block_t *in, block_t **out )
```

It receives a chunk of bit-stream corresponding to a frame (`block_t *in`) as input, performs the transrater operations, and if it succeeds, returns the trasformed bit-stream (`block_t **out`) depending on the parameters contained in the `sout_pv_t` structure:

```
typedef struct
{
    int endoffile;
    int target_bits_per_frame;
    int byte_count;

    //for decoder setup
    NalParser NALParser;
    NalUnit NAL unit;
    JPC_H264Decoder H264Dec;
    int fnum;

    //for encoder setup
    PictureEncoder in_img;
    PictureEncoder recon_img;
    JPC_H264Encoder encoder;
    H264EncoderOptions encOptions;
    PictureEncoder *p_in_img ;
    PictureEncoder *p_recon_img;
    JPC_H264Encoder *p_encoder;
    int frame_number;
    char *p_pv_buffer;
    int i_pv_buffer;
    int setup_encoder_done;
    int quant;
    int mb_in_slice;
} sout_pv_t;
```



### 3.1.2. The Transrater plug-in parameters

The Provision transrater can adapt the bit-rate of a H.264/AVC previously encoded bit-stream in real-time. We make it possible to change some of the parameters of the transrating during a streaming session, depending on the network conditions.

Several parameters of the transrater can be changed on the fly:

1. **Quantisation parameter (QP)**. The QP controls the output bit-rate of the transrater, i.e. the bit-rate increases as QP decreases. The actual bit-rate is sequence dependent.
2. **Requantization algorithm for intra frame** (0= no refine, 1=full recode)
3. **Requantization algorithm for inter frame** (0= no refine, 1=full recode, 2=refine).  
This parameter and the previous one control the transrating algorithm used by the plug-in. No refinement has the lowest complexity but provides the lowest rate-distortion performance. Full recode is almost equivalent to a full decode and re-encode, and has the highest complexity. The use of refinement provides a trade-off between complexity and rate-distortion performance. Turning on refinement or full recode will result in a big increase in complexity and may not run at full speed on the platform.
4. **Size of the search range for full pixel position** when refinement is ON (min=1, max = 67)
5. **Size of the search range for quarter pixel position** when refinement is ON (min=0, max = 3).  
For this parameter and the previous one, when refinement is used, the search range size provides a finer control on the computational complexity. Complexity increases with search range.
6. **Number of macroblocks in a slice** (min=1, max = Number of MB in frame).  
This parameters controls the number of macroblocks per slice. Using a smaller number of macroblocks per slice provides better resynchronisation and better robustness to channel losses, at the expense of compression efficiency.

Only H.264/AVC video format is supported by the transrater module.

To list the options supported by the transrater plug-in, it is possible to use the command line:

```
./vlc -p pv
```

H.264/MPEG4 AVC transrater by ProVision

```
--sout-pv-qpvalue <integer>   Quantization Parameter
--sout-pv-rqintra <integer>    Requantization Algorithm I frame
--sout-pv-rqinter <integer>    Requantization Algorithm P frame
--sout-pv-swfp <integer> Full pel search window for refine
--sout-pv-swqp <integer>   Quarter pel search window for refine
--sout-pv-mbinslice <integer> Number of Macroblock per slice
```

These options can be combined in order to obtain different bit-rates and to tune the complexity of the algorithm.

### 3.1.3. Configuration and Dynamic Adaptation of the Transrater

The initial configuration of the transrater can be performed in a similar way to the case dedicated to the SVC encoder (see section 2.1). The transrater initial plug-in parameters may be specified by



means of the setup sessionname option command. In this section, we focus more on the mechanism that allows us to change the parameters dynamically during the streaming session.

The VLC allows defining multiple interfaces. The main interface is specified by means of the --intf (-I) command line parameter. Extra interface may be started at the same time using the --extraintf command. In this way, the Remote Control (RC) and the telnet interface may be started at the same time during the same session. The RC interface allows through the TCP interface to send commands to the server.

We modified the RC interface to support new commands to dynamically configure the transrater plugin. In the following we describe the modifications to the code to support the new commands.

We added new variables in the VLC. The variable and its contents may be retrieved by the plug-in searching the variable name in the list of VLC variables. Being a command, the variable may register a callback function, here named SetTransraterValue(). This function is called each time one of the transrater parameters is modified using the remote control interface.

```
#ifndef ASTRALS
var_Create( p_intf, SOUT_CFG_PREFIX "qpvalue", VLC_VAR_STRING | VLC_VAR_ISCOMMAND );
var_AddCallback( p_intf, SOUT_CFG_PREFIX "qpvalue", SetTransraterValue, NULL );
var_Create( p_intf, SOUT_CFG_PREFIX "rqintra", VLC_VAR_STRING | VLC_VAR_ISCOMMAND );
var_AddCallback( p_intf, SOUT_CFG_PREFIX "rqintra", SetTransraterValue, NULL );
var_Create( p_intf, SOUT_CFG_PREFIX "rqinter", VLC_VAR_STRING | VLC_VAR_ISCOMMAND );
var_AddCallback( p_intf, SOUT_CFG_PREFIX "rqinter", SetTransraterValue, NULL );
var_Create( p_intf, SOUT_CFG_PREFIX "swfp", VLC_VAR_STRING | VLC_VAR_ISCOMMAND );
var_AddCallback( p_intf, SOUT_CFG_PREFIX "swfp", SetTransraterValue, NULL );
var_Create( p_intf, SOUT_CFG_PREFIX "swqp", VLC_VAR_STRING | VLC_VAR_ISCOMMAND );
var_AddCallback( p_intf, SOUT_CFG_PREFIX "swqp", SetTransraterValue, NULL );
var_Create( p_intf, SOUT_CFG_PREFIX "mbinslice", VLC_VAR_STRING | VLC_VAR_ISCOMMAND );
var_AddCallback( p_intf, SOUT_CFG_PREFIX "mbinslice", SetTransraterValue, NULL );
#endif
```

The SetTransraterValue() function searches iteratively among all the VLC objects the one that matches with the sout\_stream\_t \*p\_stream structure used by the transcoder plug-in. When it is found, depending on the parameter that caused the invocation of the callback function, the new value is properly changed in the \*p\_stream object. Finally, the transcoder plug-in should check the value of its parameters periodically (if possible, at the beginning of each frame) and modify its behaviour accordingly if any change occurs.

```
#ifndef ASTRALS
static int SetTransraterValue( vlc_object_t *p_this,
                              char const *psz_cmd,
                              vlc_value_t oldval,
                              vlc_value_t newval,
                              void *p_data )
{
    vlc_value_t    val;
    vlc_bool_t     b_found_var;
    intf_thread_t *p_intf = (intf_thread_t*)p_this;
    sout_stream_t *p_stream = NULL;
    sout_stream_t *p_stream2 = NULL;
    vlc_list_t     *p_list;
    vlc_list_t     *p_list_child;
    int            i_index = 0;
    int            i_index_child = 0;

```



```
p_list = vlc_list_find( p_this, VLC_OBJECT_SOUT, FIND_ANYWHERE );

if ( p_list )
{
    for ( i_index=0,b_found_var=VLC_FALSE;
          i_index < p_list->i_count;
          i_index++ )
    {
        p_stream = ( p_list->p_values[i_index].p_object);
        p_list_child = vlc_list_find( p_stream,
                                      VLC_OBJECT_GENERIC,
                                      FIND_CHILD );

        for ( i_index_child=0;
              i_index_child < p_list_child->i_count ;
              i_index_child++ )
        {
            p_stream2 = ( p_list_child->p_values[i_index_child].p_object);
            if ( (!var_Get( p_stream2, psz_cmd, &val ))
                && (b_found_var == VLC_FALSE) )
            {
                if ( strlen( newval.psz_string ) > 0 )
                {
                    newval.i_int = atoi( newval.psz_string );
                    b_found_var = VLC_TRUE;
                    var_Set( p_stream2, psz_cmd, newval );
                }
            }
            vlc_object_release(p_stream2);
        }
        vlc_object_release(p_stream);
    }
    if ( b_found_var == VLC_FALSE )
    {
        return VLC_EGENERIC;
    }
}
else
{
    return VLC_EGENERIC;
}
return VLC_ENOOBJ;
}
#endif
```

Finally, the contents of the variables may be changed from another application, sending a TCP packet at a pre-determined port. The source code is very similar to the one reported in section 1.2, a message is sent to the specific IP address and port that the VLC server is listening for incoming connections. The Transcoder plug-in should check regularly (e.g. with every frame) if the contents of the variables are changed and modify the internal state of the transrater accordingly.



## 4. Results

### 4.1. Tested Features

#### 4.1.1. SVC Encoder

We tested the VLC with the following command line

```
./vlc -vvv -l telnet --telnet-password videolan --rtsp-host 127.0.0.1:5554 --vlm-conf  
./mediaset_vod.cfg
```

where the telnet interface is active to allow issuing new commands. The --vlm-conf parameter allows reading the initial parameters from a configuration file.

The contents of the configuration file mediaset\_vod.cfg is shown below:

```
01 new mediaset vod  
02  setup mediaset input /home/celettol/sequence/mediaset_qcif12.5.avi  
03  setup mediaset output #transcode{vcodec=h264,vb=384,scale=1}  
04  setup mediaset option venc=hhi_h264svcenc_svc  
05  setup mediaset option sout-HHI4-cfg-File=./mediaset.cfg  
06  setup mediaset option sout-rtp-port-video=1232  
07  setup mediaset option sout-udp-caching=800  
08  setup mediaset option sout-rtp-sdp=file:./mediaset.sdp  
09  setup mediaset option mtu=1472  
10  setup mediaset option file-caching=3000  
11  setup mediaset option loop  
12  setup mediaset option noaudio  
13  setup mediaset enabled
```

The configuration file initiates a new VoD session, named mediaset (1), where the multimedia contents are acquired from an existing AVI file containing an uncompressed video track that is the base layer for the SVC encoding (2). The output is a transcoding plug-in that uses as an output standard H.264 (3), and specifically the SVC encoder (4), using the specified input parameters (5) for the enhancement layers. Lines (6)-(10) specifies the RTP session streaming parameters, and the stream is looped multiple times (11). The configuration phase is finished and the multimedia content is available to the public (13).

To play the stream, the following line is required:

```
./vlc -vvv rtsp://127.0.0.1:5554/mediaset
```

Using the telnet interface it is possible to specify new VoD session or delete the current one. For the new VoD session, we can change line (2) to use a live content, by means of a v4l:// command line. We can also change the SVC encoder parameter specifying a different configuration file on line (5).

#### 4.1.2. Transrater Plug-in

Here are some typical command line to test the plug-in (note that the line break in the second example is due to the limited width of this document and is not entered as a linefeed character):

```
vlc -l dummy input.mp4 vlc:quit --sout '#pv:std{access=file,mux=raw,dst="output.264"}'
```

```
vlc -l dummy -v --sout-pv-qpvalue 35 --sout-pv-mbinslice 396 ~/input.mp4 vlc:quit  
--sout '#pv:std{access=file,mux=mp4,dst="output.mp4"}'
```

```
vlc -l dummy -v input.mp4 vlc:quit --sout '#pv:std{access=udp,mux=ts,dst=127.0.0.1:1234}'
```



### 4.1.3. Remote Control of the Transrater Module

In order to use the remote control interface it is necessary to activate it when starting the VLC application with the `-l rc` or `--extraintf rc` command line parameter (note that the line break in the second example is due to the limited width of this document and is not entered as a linefeed character):

```
./vlc -vvv -l rc --sout-pv-qpvalue 35 mediaset.mp4
--sout '#pv:std{access=file,mux=mp4,dst="output.mp4"}'
```

The value of the transrater parameters can be changed on the fly by typing on the shell the name of the parameter followed by the new value (sout-pv-qpvalue parameter in this case):

```
[00000451] main stream output debug: adding a new input
[00000452] pv private debug: creating video transrating for fcc=`h264'
[00000456] main private debug: adding a new input
[00000456] mux_mp4 private debug: adding input
status change: ( New input: input.mp4 )
status change: ( audio volume: 256 )
status change: ( play state: 1 )
sout-pv-qpvalue 20          <=====
Transrater command sout-pv-qpvalue
old val for sout-pv-qpvalue=35
new val for sout-pv-qpvalue=20
sout-pv-qpvalue: returned 0 (no error)
[00000452] pv private debug: the new qpvalue is 20
```

The same thing can be done for all the other parameters:

```
sout-pv-rqintra
sout-pv-rqinter
sout-pv-swfp
sout-pv-swqp
sout-pv-mbinslice
```

In the same way, when streaming the transrater parameter content can be sent from a client to the server using the `vlcSendCommand_rc` function described previously.

### 4.1.4. Layered Multicast

We made some experiments for testing the layered multicast solution we implemented. We generated a test sequence and embedded it in a MP4 file format, containing several hint tracks corresponding to the different layers of scalability.

We streamed the bitstream with the following command line

```
./vlc --mp4-rtp-hint -vvv testJSVM8_7_lay.mp4 --loop
--sout '#rtp{dst=224.0.0.1,sdp= sap://224.0.0.255/test.sdp}'
```

The `--mp4-rtp-hint` option enables the implementation of the hint track decoding and allows streaming the parameters through the multicast address 224.0.0.1, using different ports for the



different layers. An SAP announcement allows the transmission of the SDP session parameters to the connecting clients.

The SDP contains the packetization-mode and sprop-parameter-sets fields that allow a proper configuration of the decoder. The layers dependency is signaled by means of draft-schierl-mmusic-layered-codec-04.txt specifications by means of the mid and lay identifiers.

```
v=0
o=- 1183456092281182 2 IN IP4 127.0.0.1
s=NONE
t=0 0
a=tool:vlc 0.8.4a
c=IN IP4 224.0.0.1/1
a=group:DDP 1 2
m=video 0 RTP/AVP 96
a=control:trackID=3
a=rtpmap:96 H264/90000
a=mpeg4-esid:2
a=fmtp:96 profile-level-id=4d400a; packetization-mode=2; init-buf-time=0;
sprop-parameter-sets=Z01ACprLFicg,aP4DGoA=;
a=mid:1
m=video 0 RTP/AVP 97
a=control:trackID=4
a=rtpmap:97 SVC/90000
a=mpeg4-esid:2
a=fmtp:97 profile-level-id=53000a; packetization-mode=2; init-buf-time=0;
sprop-parameter-
sets=Z01ACprLFicg,Z1MACksA1NZYsTk=,aP4DGoA=,aEvgZqA=,aGvgZiA=;
a=mid:2
a=depend:lay 1
```

At the client side, we need to explore the possible available channels. This is performed by means of the -S sap option, that enables the service discovery SAP service.

```
./vlc -vvv --codec hhi_h264svcdec_svc -S sap
```

When a SAP announcement is received, the SAP announcement contents are saved in the playlist of available channels, but no action is taken. The end-user may select a certain channel with a command in the control interface as

```
goto 0
```

so that the terminal will join the multicast channels and receive the multimedia data.

## **4.2. SVC Encoder Performance**

The performance of the SVC video encoder depends on several parameters given by functional requirements of the use cases. Compression ratio and the image quality are limited by the available processing power and the CPU load generated by the operating system and other applications. In particular, the achievable frame rate scales with the video resolution and depends on the number of scalability dimensions and levels.

In a real-time streaming system, it is typically the encoder that limits the achievable frame rate at a given image resolution. Performance limitations of the decoder in the terminal device apply when pre-coded and stored content is streamed through a wide-band network.



The performance of the SVC encoder has been tested with various configurations. Some results for the encoder running on one Xeon 5080 core at 3.73 GHz, (1 core used), 533MHz FSB, 2MByte 2nd Level Cache, using the ASTRALS test sequence, are given in the following tables.

Table 6: Scalable Video Encoder Performance for 2 Spatial Levels (QCIF+CIF) at 5 Temporal Levels

spatial resolution	input frame rate	bit rate	Y-PSNR
176x144	0.7813	7.3336	37.2413
	1.5625	9.1930	37.0454
	3.1250	11.7336	36.8777
	6.2500	14.6952	36.6923
	12.5000	17.7432	36.5864
352x288	0.7813	57.0070	41.3159
	1.5625	68.0750	39.5644
	3.1250	79.0805	38.6898
	6.2500	93.0238	38.1706
	12.5000	105.9864	37.8489
Encoding speed: 61.916 ms/frame (> 15 fps)			



Table 7: Scalable Video Encoder Performance for 2 Spatial Levels (QCIF+CIF) with 3 Quality Levels at Temporal Levels

spatial resolution	frames / s	SNR level	bit rate [kbit/s]	Y-PSNR [dB]
176x144	0.7813	0.0000	3.6617	
		1.0000	7.0102	
		2.0000	12.3234	36.9089
	1.5625	0.0000	4.9094	
		1.0000	8.2805	
		2.0000	15.8648	38.2034
	3.1250	0.0000	6.4797	
		1.0000	9.7172	
		2.0000	18.9727	38.7531
	6.2500	0.0000	8.3563	
		1.0000	11.3500	
		2.0000	22.7421	38.9398
12.5000	0.0000	10.8720		
	1.0000	12.7952		
	2.0000	26.1320	38.9352	
352x288	0.7813	0.0000	64.2719	41.3953
	1.5625	0.0000	79.5367	39.4631
	3.1250	0.0000	93.4156	38.4290
	6.2500	0.0000	108.1405	37.8293
	12.5000	0.0000	120.8392	37.4529
Encoding speed: 71.072 ms/frame (> 12.5 fps)				



Table 8: Scalable Video Encoder Performance for QCIF with 3 Quality Levels at Temporal Levels

frame rate	SNR level	bit rate	Y-PSNR
1.5625	0.0000	8.7203	
	1.0000	22.7000	
	2.0000	45.8141	42.5527
3.1250	0.0000	10.6828	
	1.0000	24.6125	
	2.0000	49.4500	42.2343
6.2500	0.0000	13.0786	
	1.0000	27.2095	
	2.0000	54.7579	41.9770
12.5000	0.0000	15.9808	
	1.0000	29.5168	
	2.0000	60.5208	41.7638
25.0000	0.0000	19.8088	
	1.0000	31.9480	
	2.0000	65.2840	41.5665
Encoding speed: 34.365 ms/frame (> 25 fps)			



## 5. Conclusion

The software package which is described in this document consists of different libraries that can be used as plug-ins for the VideoLAN Client (VLC). A "tar" archive has been created which contains the source code of VLC version 0.8.4a and a number of required modules, as well as binaries provided by the project partners. The complete system has been tested successfully on different linux platforms, including the ASTRALS HW platform (D3.2). The SVC plug-in supports temporal, spatial and SNR scalability. It is one of the first implementations with live encoding capability.

The encoder is capable of encoding two spatial layers (QCIF and CIF resolution) at 15 frames per second on one Xeon CPU core clocked at 3.7 GHz.

Additionally, a special RTSP plug-in has been developed which provides the control protocol for the RTP session. It supports Layered Multicast, that means it opens a number of RTP sessions, each streaming a certain layer of the SVC bit-stream. This is presumably the first implementation of the Internet Draft *Signaling media decoding dependency in Session Description Protocol (SDP)* [7] supported by ASTRALS, which can be regarded as an emerging IETF standard.

For the configuration of plug-ins developed by other WPs, the streaming server exposes different control interface, of which the remote control interface will be used by WP5 for personalisation and control of the whole ASTRALS streaming system.

## 6. List of Software Releases

<i>Date</i>		<i>Status</i>	<i>Description</i>
08 Mar 2007	SCV_Codec <sup>2</sup>	Initial version	SVC encoder Linux plug-in for VLC
18 Apr 2007	SCV_Codec	Ver0.2	including bug fixes
23 Apr 2007	SCV_Codec	Ver0.3	including bug fixes
26 Apr 2007	SCV_Codec	Ver0.4	including bug fixes
18 May 2007	STMLibrary <sup>3</sup>	Initial version	RTSP library extensions for VLC
09 July 2007	ASTRALS_D3_5_2.tar.gz	D3.5.2	Software Package Deliverable

<sup>2</sup> The SVC\_Codec distribution consists of a number of files:

- Plug-in source files hhi\_h264enc\_svc.c, hhi\_h264dec\_svc.c,
- compiled shared objects libH264FullEncSvc\_Static.so and libH264FullDecSvc\_Static.so
- and appropriate header files h264enlib.h and h264declib.h.

Note that the SVC decoder is not part of this deliverable, but included for testing purposes. The appropriate deliverables D4.4.1 and D4.4.2 are due M20.

<sup>3</sup> The STMLibrary distribution consist of several files:

Compiled shared object libSTMLibrary.so and appropriate header file STMLibrary.h



## 7. References

- [1] MPEG4 Part 10: Advanced Video Coding, ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC
- [2] Joint Draft 10 of SVC Amendment, JVT-W201.doc
- [3] MPEG4 Part 12: ISO Base Media File Format, doc ref ISO/IEC 14496-12/2005
- [4] MPEG4 Part 15: SVC File Format, doc ref ISO/IEC 14496-15/PDAM2
- [5] RFC2326, Real-Time Streaming Protocol, <http://www.ietf.org/rfc/rfc2326.txt>
- [6] S. Wenger, Y.-K. Wang, T. Schierl, "RTP Payload Format for SVC Video," AVT Working Group, <http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-svc-01.txt>, March 2007
- [7] T. Schierl, S. Wenger, *Signaling media decoding dependency in Session Description Protocol (SDP)*, doc ref <http://www.ietf.org/internet-drafts/draft-schierl-mmusic-layered-codec-04.txt>, June 22, 2007