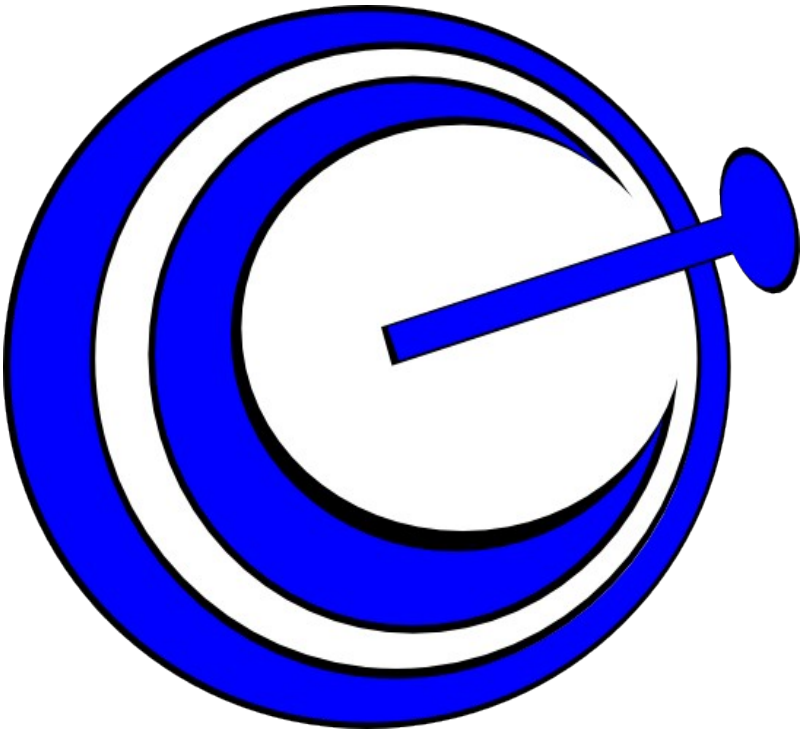


Avalpa Broadcast Server User Manual

document release	3.0
suited for OpenCaster version	3.1
date	18/04/11

Dedicated to Anna, Asli & Renzo

Thanks to:

Table of contents

General Index

Avalpa Broadcast Server User Manual....1
Table of contents....3
Introduction....5
  Avalpa Broadcast Server....5
  Avalpa Digital Engineering 5
  Use cases: some examples....5
  Demonstration....6
  OpenCaster....6
Copyright and warranty....7
  OpenCaster, the software....7
  Avalpa Broadcast Server, the manual....7
First Installation....8
  Login into the system....8
  Setting up a fixed IP address....8
  Configure multicast routing....8
Scope of the user manual and intended audience....10
  How to read the manual....10
  Interesting and basic readings....11
Avalpa Broadcast Server basic concepts and usage....12
  Your first transport stream goes broadcast....12
  Multiplexing a Single Program Transport Stream from local files....20
  Generating PSI/SI basic tables....22
  Tool dvbsnoop....26
  Transport stream bitrate....27
  IP network tools....28
Avalpa Broadcast Server for advanced datacasting....29
  EPG....29
  Teletext support....33
  DSMCC carousels and data casting....37
    DVB-SSU....37
  Interactive TV support....40
    Tool oc-update.sh....40
    MHP/MHEG5 signalling....41
    HbbTv signalling....45
    Update an object carousel with running applications....46
    Signalling Stream Events....48
  DSMCC-receive....53
  Multiplexing a Multiple Program Transport Stream from local files....54
  BISS-E and CSA output support....58
Avalpa Broadcast Server:Avalpa Web EPG GUI....60
  Reference....60

Avalpa Broadcast Server for audio/video playout....61
Introduction to ingestion....61
Extracting Program and Elementary Streams: ts2pes....63
Analysing mpeg2 video files: esvideoinfo....64
Tool vbv....66
Analysing audio elementary streams: esaudioinfo....67
How to encode mpeg2 digital video files with ffmpeg and mjtools....68
How to encode digital audio files with ffmpeg....69
How to capture DV input....71
How to generate silence....72
Audio and video real case ingestion study 73
Audio and video initial synchronization....74
Encoding with x264....75
Play out scheduling from command line....76
Re-multiplexing input transport streams....82
Tool tsorts 82
Appendix A: Acronyms, glossary and references....84
Appendix B: DVB-T transmission parameters and net bitrates....85
Appendix C: Related readings 86
Appendix D: Mpeg2 transport stream overview....87

Introduction

Avalpa Broadcast Server

Avalpa Broadcast Server is a single rack unit system featuring a collection of tools to generate, process, multiplex, play out and broadcast MPEG-2 transport stream content.

Transport stream (usual file suffixes are .TS, .TP, .TPS, .TPR) is a communication protocol for audio, video, and data which is specified in MPEG-2 (ISO/IEC standard 13818-1). An overview of the protocol is available on Appendix. Transport stream offers features for error correction for transport over unreliable media, and is used in broadcast networks such as DVB.

Avalpa Broadcast Server **inputs are usually non transport stream files stored on hard disk while output is a transport stream suitable for DVB headend integration through DVB-ASI or UDP and for stand alone broadcasting through RF modulation, VOD or IPTV**. For data broadacasting only is also possibile to output DVB packets over parallel port, on this interface jitter is unsuitable for video broadcasting.

The DVB-T/-S/-C output can be directly received by Set Top Boxes and Integrated Digital TeleVisions.

Latest version of the Avalpa Broadcast Server features also re-multiplexing features using as inputs DVB-T/-S/-C pci boards and/or udp multicast packets.

Avalpa Digital Engineering

Avalpa is an Italian company working in the digital television market since a long time with a quite broad range of products and services for broadcaster, great skills, creativity and an open mind attitude. Avalpa http://www.avalpa.com/ is the main developer of OpenCaster and wrote this manual too; Avalpa offering include also:

- Custom installations and maintenance of solutions
- Integration with others DVB systems
- Training, support and consultancy
- Further development of features and customization.

Contacts:

- a.venturi@avalpa.com CEO
- l.pallara@avalpa.com CTO
- info@avalpa.com for general questions or ordering info.

Use cases: some examples

Avalpa Broadcast Server can transmit standalone directly over a **local/residential/hotel television DVB network or integrated into a broadcaster DVB headend** and comes useful in many ways:

As a **data broadcasting headend equipment** for signalling table generation (for example for **LCN** numbering generation), electronic program guide (**EPG**) server, **data carousel** (**MHP, HbbTV, Ginga** and **MHEG5 DSMCC**) server, **teletext** and firmware upgrade (**OTA DVB-SSU**) server multiplexed with other services.

As a **standalone single box for scheduled pre-compressed Video or Video On-Demand digital television services with or without interactive support.** For vod service can manipulate pmt and/or **support CSA**.

As a **developer tool for a software house and laboratories (QA and R&D)** involved in digital television products Avalpa Broadcast Server can prototype advanced services directly on developers STB in the very same conditions of a real television environment, and pursue certifications like MHP or MHEG5 or test OTA.

Many others simpler scenarios are possible like: ASI to IP or IP to ASI converter, or IP to DVB-T/-C/-S modulator, IP to IP multiplexer, DVB-T/-C/-S to IP IRD and so on, in these use cases “alone” Avalpa Broadcast Server is usually an overkill compared to dedicated hardware soltutions unless you don't need both playout functions and re-multiplexing functions, in that case **Avalpa Broadcast Sever remultiplexing functions can avoid you additional hardware costs.**

Demonstration

You can download a transport stream multiplexed by OpenCaster showing many advanced features at: http://www.avalpa.com/the-key-values/15-free-software/59-opencaster-demo-roll The transport stream is ready to be broadcast with a DVB-T modulator using QAM 16, guard interval 1/4 and code rate 2/3,you can watch on your tv or stb. You can play it on pc by VLC or streamed on IP setting bitrate at 13271000 bps.

OpenCaster

OpenCaster is the GPL v2 open source software developed mainly by Avalpa People and its **the heart of Avalpa Broadcast Server**. Avalpa Broadcast Server is our internal reference system always under tests and runs over a **Linux Debian 6.0 Stable** system. Avalpa Broadcast Server can be requested as it is or you can contact us to integrate/customize it for your systems and solutions on the other hand **OpenCaster is available as standalone source code download for ts over UDP input/output usage on any pc** from Avalpa web site (www.avalpa.com) to anybody that wants to work on it, develop it further or use it as standalone software without dedicate hardware or others functions that doesn't require a complex system set-up and/or a DB or third parties hardware integration.

Avalpa Broadcast Server IS NOT ONLY what you get installing OpenCaster on your Linux pc but also:

1) AWE, **Avalpa Web EPG**, for event information insertion and generation based on Apache and MySQL, check http://www.avalpa.com/assets/freesoft/opencaster/AvalpaBroadcastWebEPGManual-v1.0.pdf for its manual

2) **Hardware boards for ASI and DVB- modulations for Avalpa Broadcast Server are carefully tested and choosen and their software is customized for recuded latency and improved performance** Avalpa Broadcast Server generated output has been successfully integrated on top of many third parties systems, among them it was already deployed with systems by Cisco/Scientific Atlanta, Ericsson/Tandberg, Eurotek, Harmonic/Scopus, Mitan, Screen Service, Wellav, DeltaCast, Sr-Systems...

3) **Support for installation and integration with third parties is far faster** as the hardware configuration is standard

Contact info@avalpa.com for information about system integration.

Copyright and warranty

OpenCaster, the software

Avalpa Broadcast Server, the manual

First Installation

Avalpa Broadcast Server after boot will look for a **dhcp address** if a ethernet connection is provided. If the network connection is not available you can connect monitor, keyboard and mouse and do login from the keyboard.

Login into the system

Software will usually run using Avalpa user so you can log in as “Avalpa” and the password is “Avalpa” for any need about digital television. You can also access the server as a super user using “root” user and “Avalpa” password for system set up like ip address.

If you are login from a Linux system you can simply use ssh, if you are login from a windows system you can use any ssh client like PuTTY: http://www.putty.org/

Setting up a fixed IP address

If you want to change the IP address you need to login as root and edit the file “/etc/network/interfaces”.

```
mc -e /etc/network/interfaces
```

This line enables dhcp:

```
iface eth0 inet dhcp
```

and can be replaced by a static ip address this way:

```
#iface eth0 inet dhcp
iface eth0 inet static
address 192.168.0.1
netmask 255.255.255.0
gateway 192.168.0.1
```

After the file is saved (F2 then F10) you have to restart the network issuing:

```
 /etc/init.d/networking restart
```

After restarting you can just exit:

```
exit
```

Configure multicast routing

Before doing any multicast it will be necessary to set up a route, for example:

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
```

To control it run from another pc:

```
ping -t 1 -c 2 224.0.0.1
```

it should show up your local multicast-capable hosts:

```
PING 224.0.0.1 (224.0.0.1) 56(84) bytes of data.
64 bytes from 192.168.2.203: icmp_seq=1 ttl=255 time=0.822 ms
64 bytes from 192.168.2.203: icmp_seq=2 ttl=255 time=0.724 ms
--- 224.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.724/0.773/0.822/0.049 ms
```

Scope of the user manual and intended audience

This manual deals with the philosophy and the principles behind the server design and try to smooth the path for the user to get quickly acquainted with the tools and how they are supposed to be used and interact each other.

The user manual first aim is to target technicians and specialists in broadcaster and software development companies who are looking for a reliable and accountable baseline for digital television advanced services. We really like to share most of this software as we believe, for a digital television market to grow, there's more need of different services then technologies so with a shared common proven set of basic tools, we can all focus resources and propose new contents and services on top of an openly available technology.

Then, the final goal, for this manual, is to show clearly that managing this digital television stuff is not that easy and that you, the user, should trust Avalpa if you think you should focus on your core business and leave to professionals to deal with the gory details of the technology. Who's better then Avalpa people with regard to skills, commitment and willingness to make win-win solutions with customers ?

How to read the manual

This manual is provided AS IS, and nothing is guaranteed.

We have written this stuff in good faith and we believe there are not a lot of mistakes.

If it happens you to catch one, please report to opencaster@avalpa.com so we can fix it for you and for you mates in this crazy adventure on digital television revolution. Wouldn't you like that other people would make the same choice. Right?

It is strongly suggested to don't skip any chapters of “Avalpa Broadcast basic concepts and usage” in the manual because we assume at every following chapter that the reader is confident with the content of these basic chapters.

Every chapter also states where to find tutorial files.

```
The line written in this format are command line to be typed as is in the terminal

Example:
tsrfsend sample.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

The following symbols will serve as follow:

Symbol	Description
	Pay attention to this note, it should be a hint for a smarter usage of the software

	Beware, you can get some undesirable effect if you don't understand instructions and follow them carefully
	Help! This part is not completely up to date or finished; maybe you can help us, would you like?

Then, in the paragraphs where there are practical exercises using materials to be found in the OCtutorials directories on the server, we list them in advance with such a graphical hint:

[OCTutorials/sample.ts]

Interesting and basic readings

We believe it's not healthy to run this software and read this book if you don't have already some idea of the digital television technology, so at least please read Appendix by prof. Antonio Navarro from Aveiro University for an introduction, for a more deeper overview have a look at “A Guide to MPEG Fundamentals and Protocols Analysis” freely available by Tektronix.

Good books to get into the details are: “Digital Video and Audio Broadcasting Technology” by W.Fisher and “Interactive TV standards” by Steve Morris,

The main specification OpenCaster refers to is:

ISO/IEC 13818-1 Information technology -- Generic coding of moving pictures and associated audio information: Systems

As the specifications are a bit dry and don't really explain the context we suggest you to read them after you are already quite confident with the overall picture, however it's quite useful to have it on your desk to check the exact meaning of every parameter used by OpenCaster that strictly follows the name used in the specification for every variable.

Avalpa Broadcast Server basic concepts and usage

Let's begin with the simplest way to stream something “out in the air”. In the next paragraphs we'll move from easy exercises to more complex ones, explaining in great detail what are trying to achieve and why we are doing that way.

Don't forget to login as **“avalpa” user and password “avalpa”**, after that you are ready for your first tutorial

Your first transport stream goes broadcast

[start-here]

A Transport Stream (TS) is made of packets, a packet is the minimum unit that cannot be splitted, has a fixed length of 188 bytes and always starts with a synch byte (0x47).

Each table or elementary stream in a transport stream is identified by a 13-bit PID. A demultiplexer extracts elementary streams from the transport stream in part by looking for packets identified by the same PID. In most applications, time-division multiplexing will be used to decide how often a particular PID appears in the transport stream.

Transport stream brings also a logic representation of the carried content, mainly with the concept of programs. A single program is described by a Program Map Table (PMT) which has a unique PID, and the elementary streams associated with that program have PIDs listed in the PMT. For instance, a transport stream used in digital television might contain three programs, to represent three television “channels”. Suppose each channel consists of one video stream, one or two audio streams, and any necessary metadata. A receiver wishing to decode a particular "channel" merely has to decode the payloads of each PID associated with its program. It can discard the contents of all other PIDs.

To make sure you can broadcast and receive a transport stream check your set up with the sample TS presenting only one video with one audio.

After login change into directory start-here:

```
cd start-here
```

If your server is equipped with RF output you will need to set modulation parameters:

```
tsrfsend sample.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

if your server is equipped with DVB-ASI from you will have:

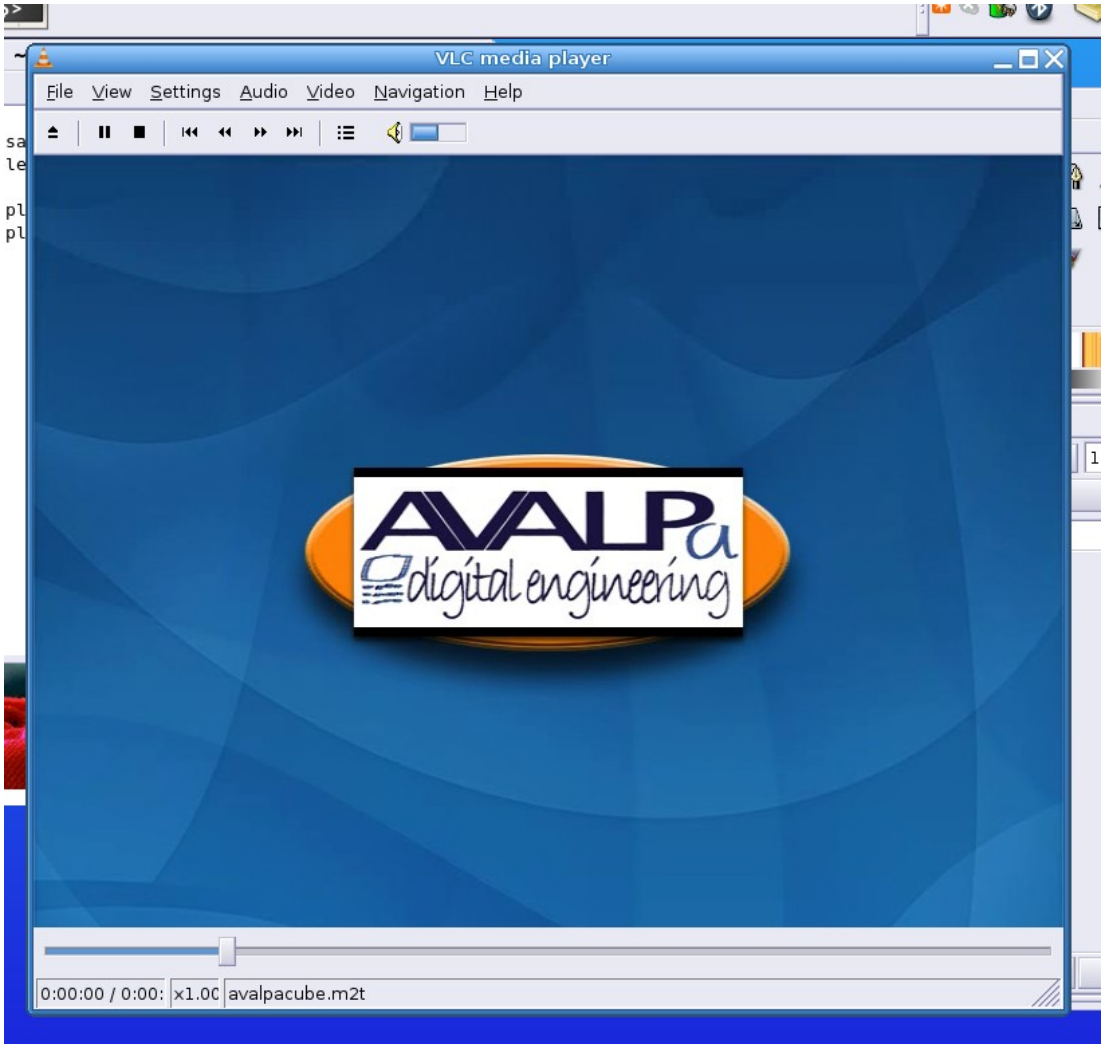
```
tsasisend sample.ts -r 13271000
```

In both cases or if you are using OpenCaster without DVB hardware you can also always output over IP networking with **tsudpsend,** a tool for broadcasting using multicast, to manage this you can try:

```
tsudpsend sample.ts 224.0.1.2 12345 13271000
```

The first problem we face is that sample.ts is too short and you won't have enough time to tune and search the service on the set top box so let's introduce a tool:

tsloop



This tool will put the transport stream file on loop generating a new "endless" stream where the original file sample.ts is repeated.

Tsloop is a very simple tool and will just fix the continuity counter at transport level so there is not a transport error. This is perfect for broadcasting data but not audio and video because at the end of the stream the “clock” of audio and video will go back in time: Tolerant decoders will detect a time skew on the stream clock and correct their internal clock so you won't see any glitch.

Many decoders, however, will have problems with such a simple set-up because they are expecting a progressive video and audio time line and a correct buffer management always progressive and forward-looking, we will address all these issues later on, for now to keep first steps easy remind that every 8 seconds there could be some artefacts due to the improper loop and to the modest decoder recovery skills.

VLC versions after 0.8.4 are reported to don't work any more with the tutorial files sample.ts and firstvideo.ts because of these reasons regarding poor clock management so even if VLC works correctly with a real set-up of OpenCaster where clock is properly stamped it will fail to show the tutorials until clock management is put in place. (Check Demonstration chapter for a transport stream that plays correctly on VLC for a longer time)

On the other Mplayer is able to correctly manage the clock issue:

```
mplayer -nocache -noidx udp://224.0.1.2:1234
```

Before using tsloop tool we need to introduce an important concept in OpenCaster operations: the “fifo” (“First In First Out”)

fifo connections are the way OpenCaster tools pass TS packets to each other, the packets are usually pulled out from the fifo but later on we will see more complex set-ups where pull and push live together.

The following command creates a fifo:

```
mkfifo myfirstfifo.ts
```

after you can execute tsloop and the play command together:

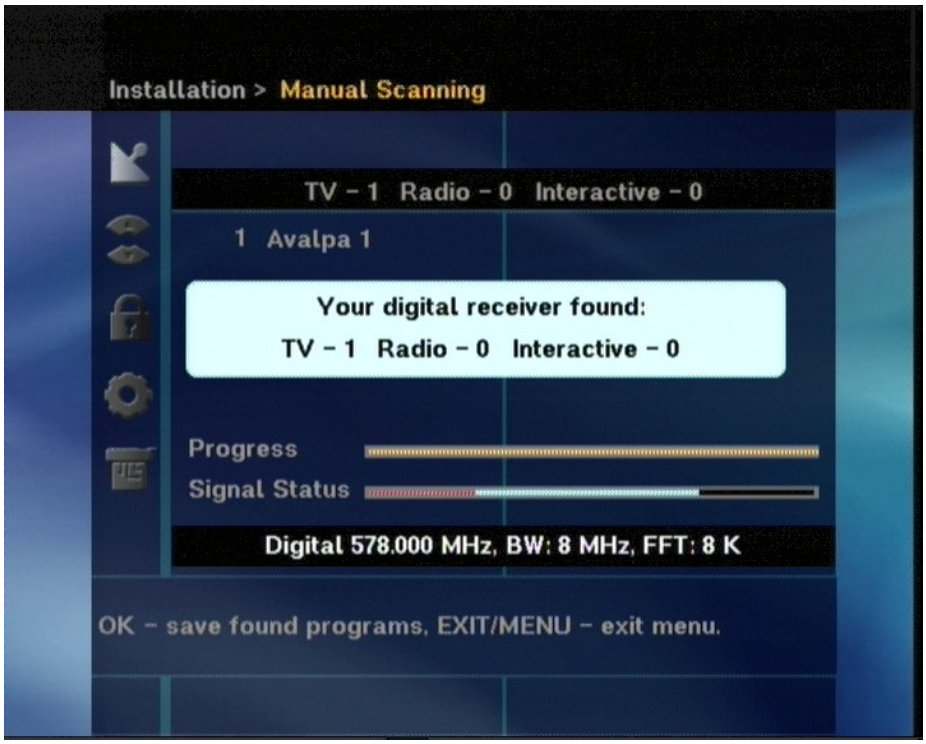
```
tsloop sample.ts > myfirstfifo.ts &
tsrfsend myfirstfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

The '&' at the end of tsloop command is necessary to put the process in background .

The tsloop command will wait or the second command to "pull" the data out of the **myfirstfifo.ts**.

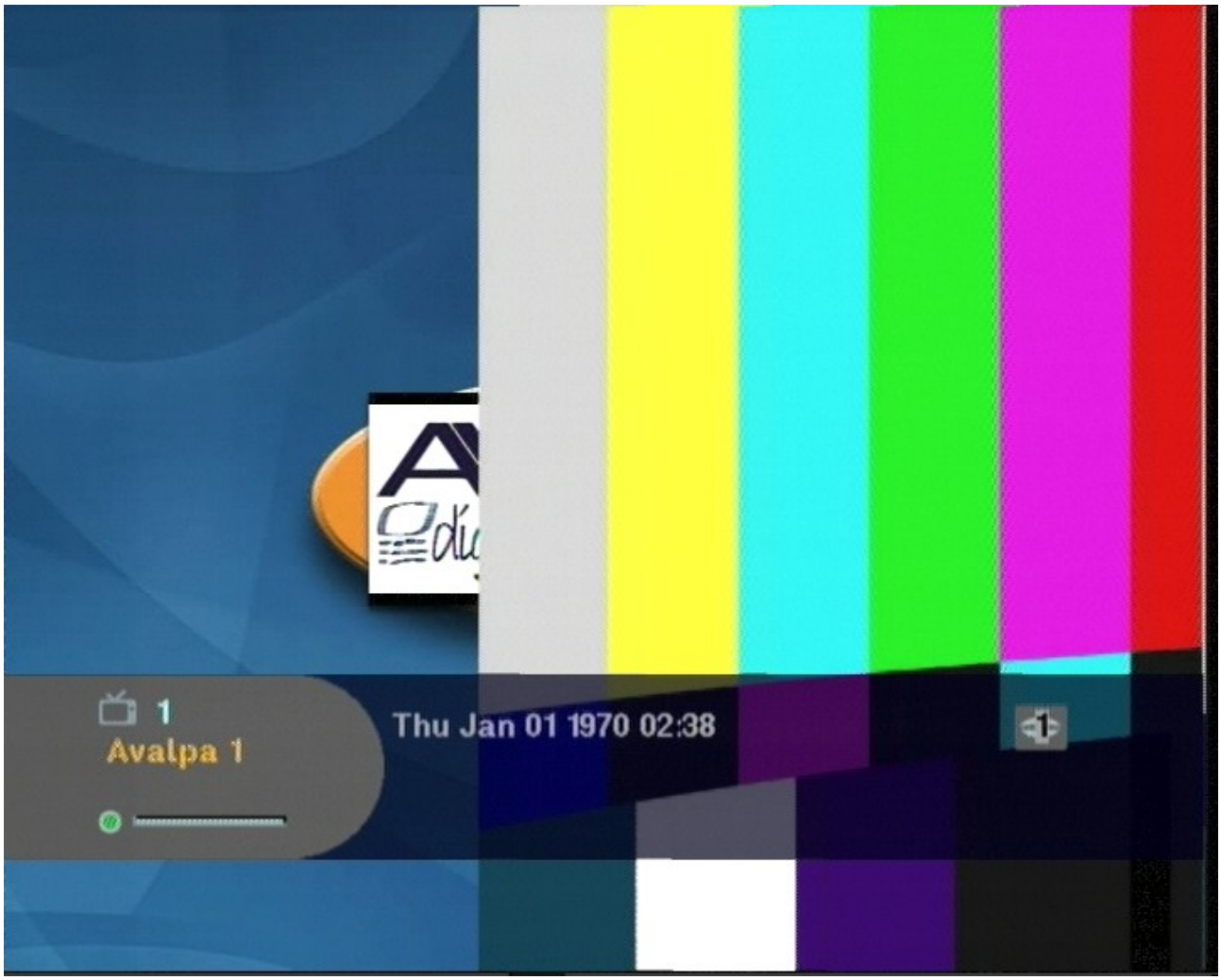
You will be able to receive the stream with a dvb-t decoder tuning to 578 MHz, the following pictures show the procedure step by step. Set top box manual scan menù reports a signal is locked:

the scan finds 1 service: **Avalpa 1**



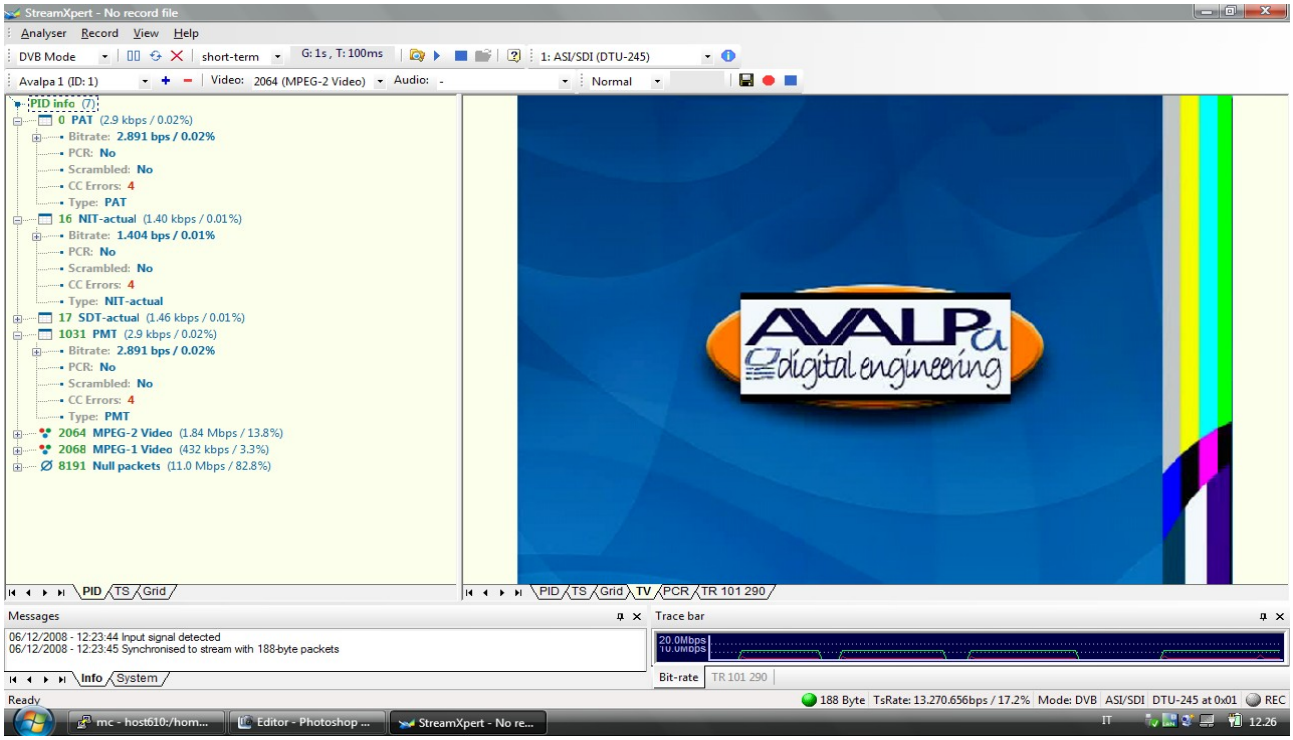
Check the modulation parameters that the decoder is using actually match the one from the command line, the signal status strength is also high.

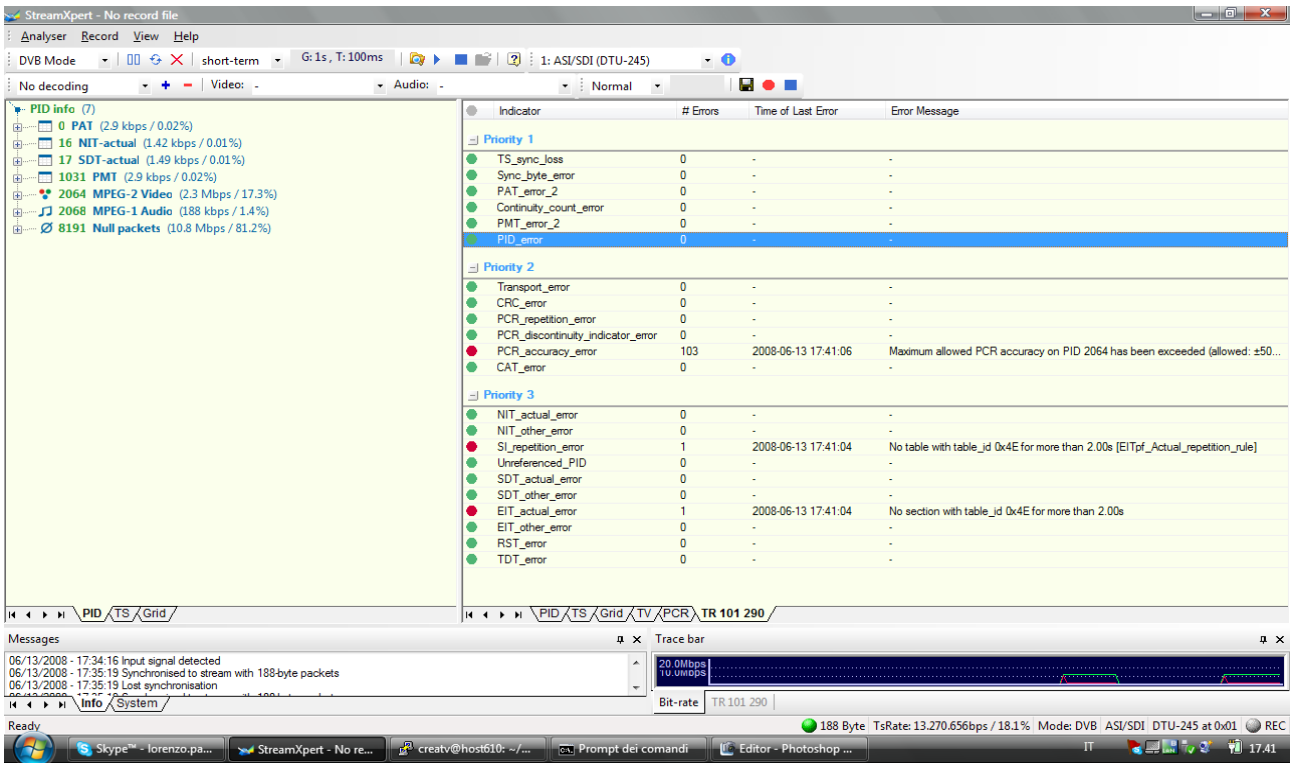
let's check the service Avalpa 1:



you will probably notice a lot of artefacts and the colour bars won't turn smoothly over Avalpa logo, let's check what's happening with some analyser tool.

We will show, as example, some screenshots from StreamXpert analyzer by Dektec, here they come:





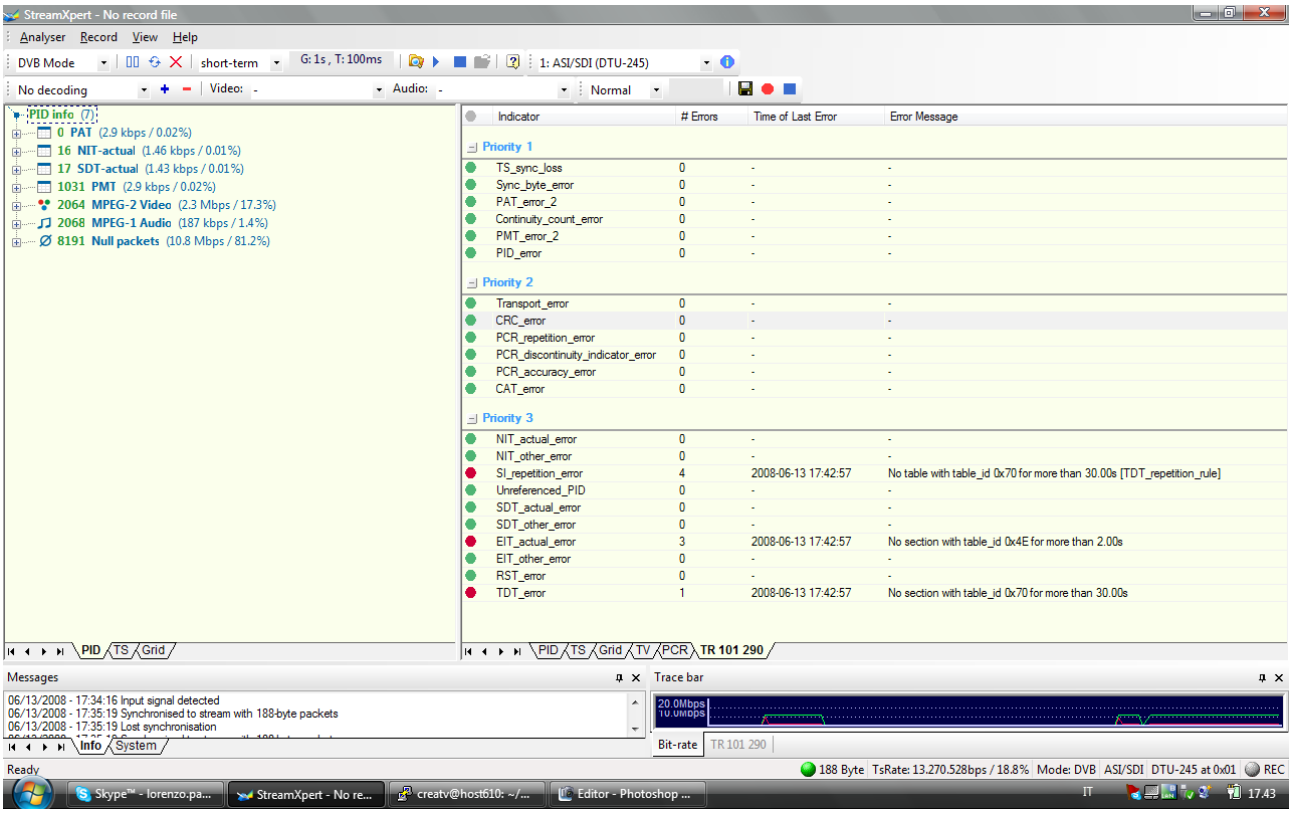
something seems wrong with the PCR, that's the timing information of the transport stream, there are two reasons for this:

- we need to properly manage the loop of the video to avoid the embedded time info to go backward in time
- sample.ts was a result of a multiplexing without PCR correction,

both problems can be addressed using “tsstamp” and another fifo connection, so here we go:

```
mkfifo mysecondfifo.ts
tsloop sample.ts > myfirstfifo.ts &
tsstamp myfirstfifo.ts 13271000 > mysecondfifo.ts &
tsrfsend myfirstfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

(13271000 if you recall was the bit rate we are modulating)

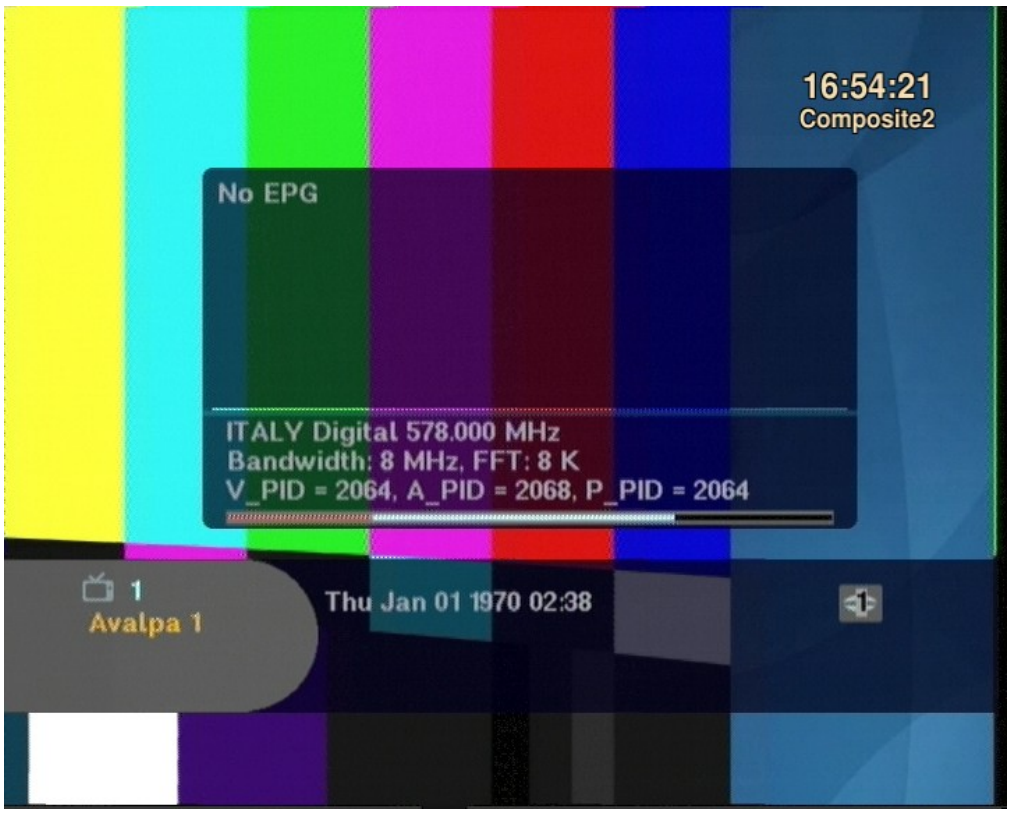


As you can see PCR is better now and an error can occur at maximum at the end of the loop, some red lights about Electronic Program Guide are reported, however many decoder will still have problem to decode this stream after the first loop, because a transport stream needs to respect not only transport constraints but also many constraints at elementary stream level on how the end and the start of the streams actually chain together. This kind of constraints are related to the inner elementary streams themselves and are not shown in this analyser that is focused on the transport stream level. We will

address them later on.

As we want to give away tutorial examples for OpenCaster without huge video file downloads **we have prepared some “loop-able” transport streams for the next chapters but don't forget that usually a transport stream is not seamless loop-able on any decoder without some more processing of its elementary streams, advanced chapters will show how to deal with this issue.**

Pressing the info button on the decoder we figure out what were the last complains of the analyser tool:



the Electronic Program Guide info are missing and time information looks odd! We will fix them later, in the mean time notice that the decoder is so smart to remind us of audio and video PIDs, they will come handy soon...

Multiplexing a Single Program Transport Stream from local files

[first-file-mux]

To broadcast a transport stream from audio and video frames you need to encode audio and video.

We will discuss later how to encode audio and video because it's a more advanced topic, for now let's focus on multiplexing.

The mandatory standard for free view requires the following signalling tables: PAT, PMT, SDT and NIT for a DVB stream.

Now that you got all the streams from the sample tutorial, you need to mux them together before output them, tscbrmuxer is the tool for this task.

tscbrmuxer gets as input a .ts file and its own bitrate, for example:

```
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts > muxed.ts
```

N.B. you need to stop tscbrmuxer with a signal (CTRL-C) or it will go on filling your hard disk, that's because input files are looped and muxed.ts is not a fifo.

The resulting muxed.ts will have a total bit rate sum up of the the bit rate of all the ts input files that is: 2496916 bit per second.

The bit rate of the video stream and of the audio stream comes from the encoder settings, so for now just take them as reported, for the other bit rates will introduce some concepts here.

Bit rates are very important, MPEG networks are usually constant bit rate so you will often need to output files at the correct bit rate or you will have a countless number of side effects.

How do we know the bit rate of the signalling streams? MPEG, DVB and all the others standards specify the bit rates, you just need to pay attention to a detail:

PSI bit rates are expressed as section per second, for example a PAT has to be broadcasted every 500 ms, but a PAT, as all the other sections, can be 1 ts packet up to 23 ts packets depending on its content, so bit rate needs to be changed according to the sections size.

Set the bit rate to the maximum allowed is not a wise choice because 99% of the signal tables are usually 1 or 2 packets, so it would result in a big waste of bandwith.

Plan carefully your bit-rates when starting the muxing so you won't need to stop it later on, bit-rates can't be changed without artefacts and for the decoders the best way to do this is stop all the muxing and restart.

There is still another point to face before you can broadcast the multiplexed .ts, the bit rate of the resulting muxed.ts is not a bit-rate that matches a DVB modulation scheme. How can we fix that?

An option is to just make some data stream faster or to increase the video bit rate but there is also another choice: add a "NULL PID" filling stream.

```
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts b:10774084 null.ts > sample.ts
```

A null stream is a stream made of empty packets (0xFF) with fixed PID 8191, to get the required bit rate you need to add a null.ts to the multiplexing with the missing bit rate to reach the required one.

In the example there is a massive insertion of null packets, more than 10Mbps but this is just an example!

Noticed that also sample.ts from the first tutorial was generated in the same way, just to make thing easier to start.

To output the final result use tscbrmuxer instead of tsloop from the previouse example:

```
mkfifo myfirstfifo.ts
mkfifo mysecondfifo.ts
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts b:10774084 null.ts> myfirstfifo.ts &
tsstamp myfirstfifo.ts 13271000 > mysecondfifo.ts &
tsrfsend mysecondfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

On the other hand if you are going to broadcast a TS over an UDP network, where there are no fixed bit rates, you will be better to avoid sending Constat Bit Rate null packets so the final batch will be:

```
mkfifo myfirstfifo.ts
mkfifo mysecondfifo.ts
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts > myfirstfifo.ts &
tsstamp myfirstfifo.ts 2496916 > mysecondfifo.ts &
tsudpsend mysecondfifo.ts 224.0.1.2 7001 2496916
```

Now you could try to tune this program on the decoder as for the sample.ts the outcome should be the same audio/video. If you are broadcasting on an IP network, any modern pc with VLC will be able to display the stream with an input like: udp://@224.0.1.2:7001 , most of the IPTV set top box should also easily manage it.

If you want to record your output you can use the tool tsdoubleoutput:

```
mkfifo myfirstfifo.ts
mkfifo mysecondfifo.ts
mkfifo mythirdfifo.ts
touch outputcopy.ts
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts > myfirstfifo.ts &
tsstamp myfirstfifo.ts 2496916 > mysecondfifo.ts &
tsdoubleoutput mysecondfifo.ts outputcopy.ts > mythirdfifo.ts &
tsudpsend mythirdfifo.ts 224.0.1.2 7001 2496916
```

Pay attention outputcopy.ts file will grown and fill your hard disk! However we will see that you often want to check your output when you are trying new set-ups so it is very useful.

Generating PSI/SI basic tables

[psi-generation]

Let's change our transport stream, for example we want to change the service name from "Avalpa 1" to "OpenCaster 1".

The name of the service is in the SDT table so we will need to generate a new sdt.ts.

sdt.ts is generated by a tools "sec2ts" thats encapsulate sections into ts packets, it works this way:

```
sec2ts 17 < firstsdt.sec > firstsdt.ts
```

sdt.sec is the section of the sdt, 17 is the well known pid number for the sdt stream.

To generate the sdt.sec we need to use a different language called python, have a look at firstsdt.py, here is reported the main part:

```
[...]

# Service Description Table (ETSI EN 300 468 5.2.3)

sdt = service_description_section(
    transport_stream_id = 1,  # demo value, an official value should be demanded to dvb org
    original_network_id = 1,  # demo value, an official value should be demanded to dvb org
    service_loop = [
        service_loop_item(
        service_ID = 1, # demo value
        EIT_schedule_flag = 0, # 0 no current even information is broadcasted, 1 broadcasted
        EIT_present_following_flag = 0, # 0 no next event information is broadcasted, 1 yes
        running_status = 4, # 4 service is running, 1 not running, 2 starts in a few seconds,3 pause
        free_CA_mode = 0, # 0 means service is not scrambled, 1 means at least a stream is scrambled
            service_descriptor_loop = [
                service_descriptor(
                    service_type = 1, # digital television service
                        service_provider_name = "Avalpa",
                        service_name = "Avalpa 1",
                ),
            ],
        ),
    ],
)
```

This python script can be executed with

```
./firstsdt.py
```

and will generate sdt.sec.

You can edit firstsdt.py to change the **service_name** value and then execute it again to generate the new sdt.sec.

```
mc -e firstsdt.py
```

After you located and changed the service name, save the file (F2) exit (F10) and enter the following command:

```
./firstsdt.py
```

this command will generate a new firstsdt.sec, while:

```
sec2ts 17 < firstsdt.sec > firstsdt.ts
```

will generate a new firstsdt.ts

What will happen to the old one? if you the old firstsdt.ts was being multiplexed the tscbrmuxer will get the new firstsdt.ts as soon as the previous one was sent for the last time.

So get back to "Multiplexing a Single Program Transport Stream" set up, replace firstsdt.ts while the processes are running and rescan the service on the box.

As you have probably noticed firstsdt.py has a complex structure, that's because it maps 1:1 the original specifications and allows to use all of its signalling.

If you look at "ISO/IEC 13818-1" and "EN 300 468" you will find the description of every single table and field, OpenCaster implements the most important of them.

Have a look at firstconfig.py for a complete example of signalling, a single python script describes all of the signalling tables you are using, you can split it in more files but a single file is probably easier to edit until you don't need to manage a very complex setup .

Take your time to understand it properly and compare it with the specifications of the single tables.

```
[...]

#
# Shared values
#

avalpa_transport_stream_id = 1 # demo value, an official value should be demanded to dvb org
avalpa_original_transport_stream_id = 1 # demo value, an official value should be demanded to dvb
org
avalpa1_service_id = 1 # demo value
avalpa1_pmt_pid = 1031

#
# Network Information Table
# this is a basic NIT with the minimum desciptors, OpenCaster has a big library ready to use
#

nit = network_information_section(
    network_id = 1,
    network_descriptor_loop = [
        network_descriptor(network_name = "Avalpa",),
    ],
    transport_stream_loop = [
        transport_stream_loop_item(
            transport_stream_id = avalpa_transport_stream_id,
                original_network_id = avalpa_original_transport_stream_id,
                transport_descriptor_loop = [
                    service_list_descriptor(
                        dvb_service_descriptor_loop = [
                            service_descriptor_loop_item(
                                service_ID = avalpa1_service_id,
                                service_type = 1, # digital tv service type
                            ),
                        ],
                    ),
                ],
            ),
        ],
    version_number = 1, # you need to change the table number every time you edit, so the decoder
```

```
will compare its version with the new one and update the table
    section_number = 0,
    last_section_number = 0,
    )

#
# Program Association Table (ISO/IEC 13818-1 2.4.4.3)
#

pat = program_association_section(
    transport_stream_id = avalpa_transport_stream_id,
    program_loop = [
        program_loop_item(
            program_number = avalpa1_service_id,
            PID = avalpa1_pmt_pid,
        ),
        program_loop_item(
            program_number = 0, # special program for the NIT
            PID = 16,
        ),
    ],
    version_number = 1, # you need to change the table number every time you edit, so the decoder
will compare its version with the new one and update the table
    section_number = 0,
    last_section_number = 0,
    )

#
# Service Description Table (ETSI EN 300 468 5.2.3)
# this is a basic SDT with the minimum desciptors, OpenCaster has a big library ready to use
#

sdt = service_description_section(
    transport_stream_id = avalpa_transport_stream_id,
    original_network_id = avalpa_original_transport_stream_id,
    service_loop = [
        service_loop_item(
            service_ID = avalpa1_service_id,
            EIT_schedule_flag = 0, # no current even information is broadcasted, 1 broadcasted
            EIT_present_following_flag = 0, # no next event information is broadcasted, 1 yes
            running_status = 4,# service is running,1 not running,2 starts in a few seconds, 3 pause
            free_CA_mode = 0, # service is not scrambled, 1 means at least a stream is scrambled
            service_descriptor_loop = [
                service_descriptor(
                    service_type = 1, # digital television service
                    service_provider_name = "Avalpa",
                    service_name = "Avalpa 1",
                ),
            ],
        ),
    ],
    version_number = 1, # you need to change the table number every time you edit, so the decoder
will compare its version with the new one and update the table
    section_number = 0,
    last_section_number = 0,
    )


#
# Program Map Table (ISO/IEC 13818-1 2.4.4.8)
# this is a basic PMT the the minimum desciptors, OpenCaster has a big library ready to use
#

pmt = program_map_section(
    program_number = avalpa1_service_id,
    PCR_PID = 2064,
    program_info_descriptor_loop = [],
    stream_loop = [
        stream_loop_item(
            stream_type = 2, # mpeg2 video stream type
```

```
            elementary_PID = 2064,
            element_info_descriptor_loop = []
        ),
        stream_loop_item(
            stream_type = 3, # mpeg2 audio stream type
            elementary_PID = 2068,
            element_info_descriptor_loop = []
        ),
    ],
    version_number = 1, # you need to change the table number every time you edit, so the decoder
will compare its version with the new one and update the table
    section_number = 0,
    last_section_number = 0,
    )

[...]
```

firstconfig.py ends with system commands invoking sec2ts so they directly generates .ts files without the need to execute manually sec2ts so execute them to generate the psi .ts files.

Tool dvbsnoop

[start-here]

dvbsnoop http://dvbsnoop.sourceforge.net/ is a very useful software to analyse transport stream, and it is provided in Avalpa Broadcast Server.

You can use it to check most the signalling in DVB networks and also Avalpa Broadcast Server own output.

To parse PAT from a transport stream file:

```
dvbsnoop -tsraw -s ts -tssubdecode -if sample.ts -N 2 0
```

Dvbsnoop will show you information about the PAT:

```
PID:  0 (0x0000)  [= assigned for: ISO 13818-1 Program Association Table (PAT)]
Guess table from table id...
PAT-decoding....
Table_ID: 0 (0x00)  [= Program Association Table (PAT)]
section_syntax_indicator: 1 (0x01)
(fixed): 0 (0x00)
reserved_1: 3 (0x03)
Section_length: 17 (0x0011)
Transport_Stream_ID: 1 (0x0001)
reserved_2: 3 (0x03)
Version_number: 1 (0x01)
current_next_indicator: 1 (0x01)  [= valid now]
Section_number: 0 (0x00)
Last_Section_number: 0 (0x00)

Program_number: 1 (0x0001)
reserved: 7 (0x07)
Program_map_PID: 1031 (0x0407)

Program_number: 0 (0x0000)
reserved: 7 (0x07)
Network_PID: 16 (0x0010)

CRC: 614848810 (0x24a5d92a)
```

Transport stream bitrate

[start-here]

To learn a bit rate of a transport stream file you can use **tspcrmeasure** in this way:

```
tspcrmeasure sample.ts 4000000
```

Where 4000000 is your initial guess of the bit rate, the tool needs a guess because it will compare the input guess value with the real bit rate giving information about the differences.

After the first execution you can try again with the values suggested by the tool.

Usually you should just verify that "instant bit rate prints" are quite constant that means the transport stream is not corrupted and can be processed.

Sometimes bit rate is not constant because null packets have been removed for storage purpose, in this case null pids need to be reinserted before use.

For a better understanding of PCR have a look at Tektronix's "A Layman's Guide to PCR Measurements"

A typical print of tspcrmeasure is:

```
pid 2064, new pcr is 614805912, pcr delta is 676290, (25.047778 ms), indices delta is 41548 bytes,
( 25.568000 ms), pcr accuracy is -0.0005202222, instant ts bit rate is 13269999.5564033184
```

The first value is the pid number of the pcr/video stream.

The second value is the pcr value itself.

The third value is the pcr difference from the previous pcr value and in brackets there is the time passed counting the pcr ticks.

The fourth value is the difference in bytes from the pcr position into the file and in brackets there is the time passed with the guessed bitrate on input.

The fifth value is the accuracy of the predicted pcr.

The last value is the actual bit rate measure between pcr ticks and ts bytes.

If the transport stream brings PCR information the tool will be able to do the computations otherwise the bit rate information need to be known out-of-band.

IP network tools

[start-here]

OpenCaster 2.0 for the first time featured some tools to send a transport stream over an IP network and their usage is very simple to exploit. As in every real word problem, there are some issues you should be concerned of.

Let's see an example:

```
tstcpreceive 7001 > received.ts
Binding the socket...
```

This command will start a process waiting for an incoming connection on port 7001 that can be made from another pc:

```
tstcpsend sample.ts 192.168.1.2 7001 3000000
```

Where the parameters are the transport stream to send, the destination ip address and port and the bit rate of the transfer.

The first thing to keep in mind is that **the bit rate is the transport stream bit rate and not the ip packets bit rate** . As far as there is enough bandwidth the file will be transferred and due to the TCP properties we won't miss any packet.

Another option is **to use UDP packets instead of TCP but this implies that same packet could be lost or repeated** so there is no guarantee that a transport stream received from an UDP stream is going to be error-free and so it's a risk to use UDP as a “contribution link” between OpenCaster tools without implementing some recovery strategy. **The main reason to send a transport stream over UDP is to enable multicast** for end user distribution; the decoder receiving the stream will implement the necessary algorithms of error-concealment to manage packet loss while decoding the stream in the same way it recovers from broadcast errors for weak signal conditions.

Why not RTP?

Maybe we will support RTP in the future, for now it is not available.

On some scenarios can make sense to pay for more memory in the receiver than pay for network bandwidth so to reduce network usage NULL packets are removed from the transport streams and this can lead to migrate a Constant Bit Rate transport stream into a Variable Bit Rate transport stream. This happens if NULL packets are not uniform inserted as shown in the chapter: “Multiplexing a Single Program Transport Stream”.

Instead of NULL packets a time stamp is added at sending time and NULL packets are re-generated by the receiver while analyzing RTP packets in the additional memory buffer. OpenCaster tipical usage scenarios are local networks and/or DVB headends so we are more concern about keep jitter and latency lower as possible than network cost, that's why for now we strictly supports UDP only.

Avalpa Broadcast Server for advanced datacasting

EPG

[eit]

An interesting signalling is the EIT, Event Information Table, these tables are received by the box and used to generate the Electronic Program Guide.

Have a look at eitconfig.py, it is a template, so you'll need to change the date and the time from the python config to your current time so the decoder will show the events as they are coming in a short time.

```
[...]

#
# Event Information Table (ETSI EN 300 468 5.2.4)
#

eit = event_information_section(
    table_id = EIT_ACTUAL_TS_PRESENT_FOLLOWING,
    service_id = avalpa1_service_id,
    transport_stream_id = avalpa_transport_stream_id,
    original_network_id = avalpa_original_transport_stream_id,
    event_loop = [
        event_loop_item(
            event_id = 1,
            start_year = 108, # since 1900
            start_month = 6,
            start_day = 10,
            start_hours = 0x00, # use hex like decimals
            start_minutes = 0x00,
            start_seconds = 0x00,
            duration_hours = 0x23,
            duration_minutes = 0x00,
            duration_seconds = 0x00,
            running_status = 4,#service is running,1 not running,2 starts in a few seconds,3 pause
            free_CA_mode = 0, # service is not scrambled, 1 means at least a stream is scrambled
            event_descriptor_loop = [
                short_event_descriptor (
                    ISO639_language_code = "ita",
                    event_name = "epg event name",
                    text = "this is an epg event text example",
                 )
             ],
         ),
    ],
    version_number = 1,
    section_number = 0,
    last_section_number = 1, # pay attention here, we have another section after this!
    last_segment_section_number = 1,
    )


eit_follow = event_information_section(
    table_id = EIT_ACTUAL_TS_PRESENT_FOLLOWING,
    service_id = avalpa1_service_id,
    transport_stream_id = avalpa_transport_stream_id,
    original_network_id = avalpa_original_transport_stream_id,
    event_loop = [
       event_loop_item(
          event_id = 2,
```

```
          start_year = 108, # since 1900
          start_month = 06,
          start_day = 10,
          start_hours = 0x23,
          start_minutes = 0x30,
          start_seconds = 0x00,
          duration_hours = 0x12,
          duration_minutes = 0x00,
          duration_seconds = 0x00,
          running_status = 4,# service is running,1 not running,2 starts in a few seconds,3 pause
          free_CA_mode = 0, # service is not scrambled, 1 means at least a stream is scrambled
          event_descriptor_loop = [
              short_event_descriptor (
                  ISO639_language_code = "ita",
                  event_name = "epg event name 2",
                  text = "this is the following text example",
              )
          ],
       ),
    ],
    version_number = 1,
    section_number = 1, # this is the second section
    last_section_number = 1,
    last_segment_section_number = 1,
    )
[...]
```

To add EPG to your set up you will need to add the firsteit.ts to your muxing and it would be wise also to signal a time info. That's what tstdt tool is for.

tstdt is a tool that set the time of the service, this is just a hint for the decoders user clock and doesn't affect any low level buffer synchronization issue, the time is read from the pc clock.

The usage is very simple:

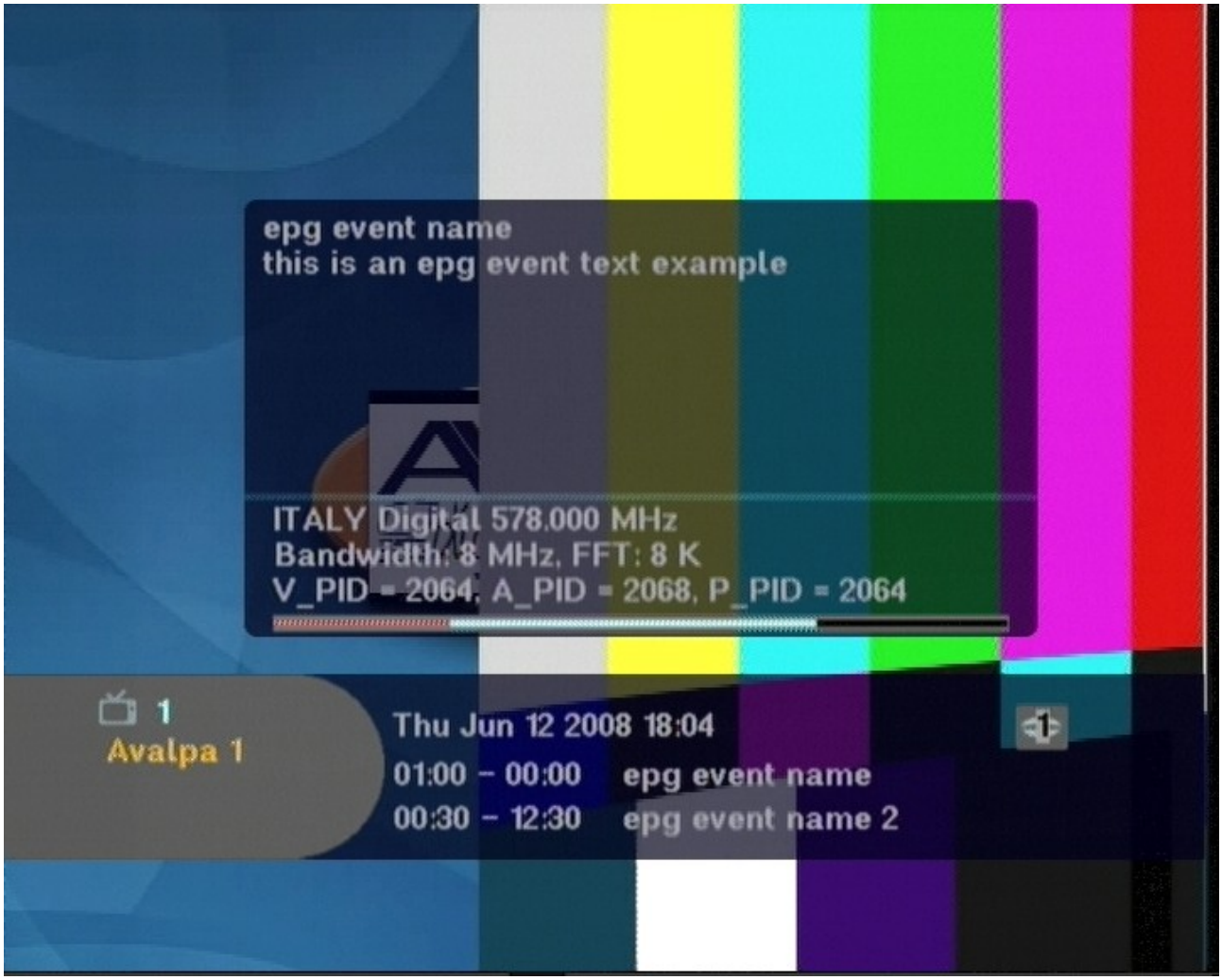
```
tstdt inputs.ts > output.ts
```

The input ts needs to already have stub tdt packets inserted at multiplexing time so the tool will replace them with packets at the current time.

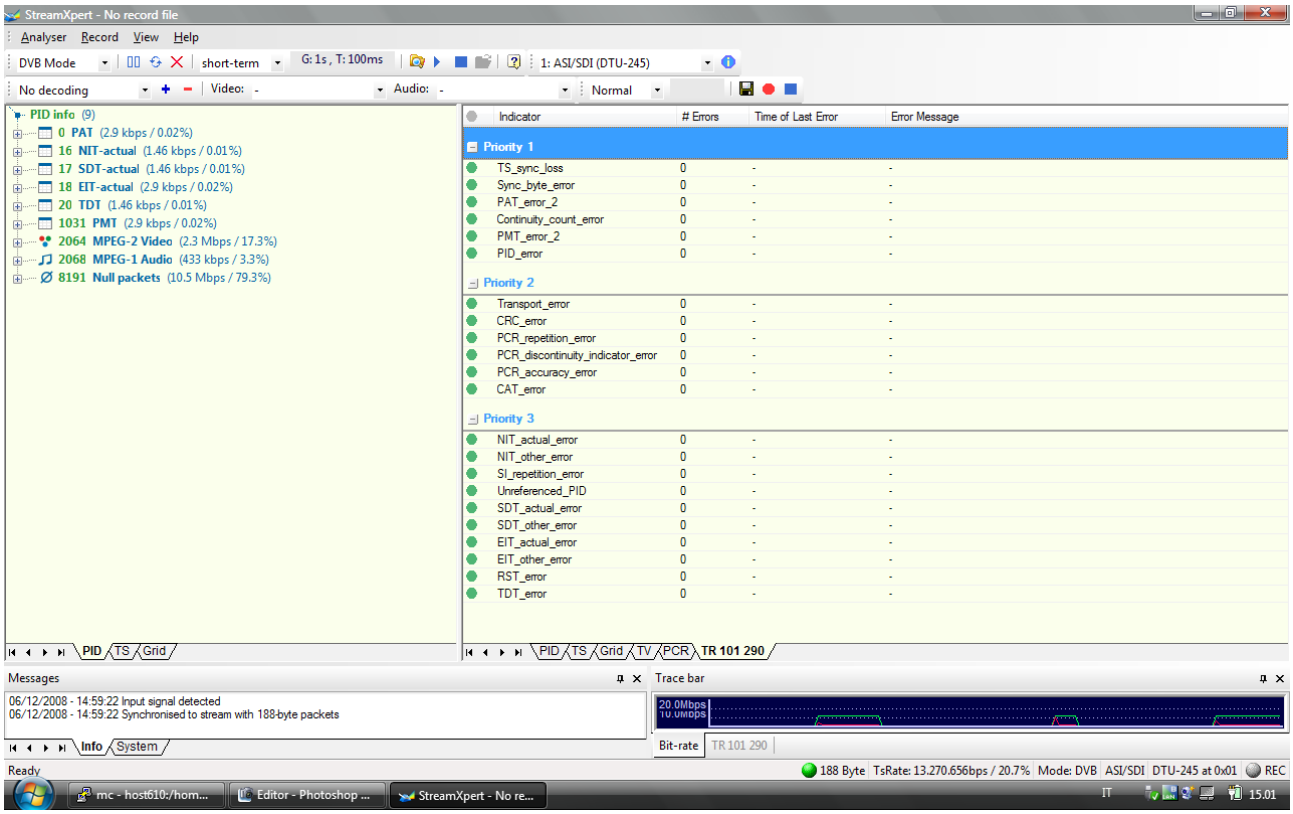
Pay attention that if you are processing the transport stream faster than real-time I.e. with a rate higher then the own bit rate you are going to write into the hard disk a tdt time information quite meaningless.

```
mkfifo fifomuxed.ts
mkfifo fifotdt.ts
mkfifo fifotsstamp.ts
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts b:2000 firsteit.ts b:2000 firsttdt.ts b:10770084 null.ts >
fifomuxed.ts &
tstdt fifomuxed.ts > fifotdt.ts &
tsstamp fifotdt.ts 13271000 > fifotsstamp.ts &
tsrfsend fifotsstamp.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

Let's have a look at the decoder informations:


epg event name
this is an epg event text example
ITALY Digital 578.000 MHz
Bandwidth: 8 MHz, FFT: 8 K
V_PID = 2064, A_PID = 2068, P_PID = 2064
1
Avalpa 1
Thu Jun 12 2008 18:04
01:00 - 00:00 epg event name
00:30 - 12:30 epg event name 2

and to the analyser too:



All green lights! This is as we like it most, as you can see there are no more errors because the last red lights were about missing EIT and TDT, tables.

Teletext support

[txt]

This tutorial describes support for digital teletext generation and multiplexing. To describe how does teletext works and its packets layout is out of this document but we will address how teletext packets are encapsulated into transport stream.

Once a receiver decoded the packets has two choices not exclusive: modulate the teletext signal into an analog video output and/or render the teletext as on screen graphics, however when the video output is HDMI, the only way to support teletext for the decoder is on screen graphics because up to now HDMI is not supporting in band teletext.

Teletext packets are generated by pythons scripts presented later, let's suppose we have some teletext packets stored in one file, we will present two utilities that take care of the forthcoming process: txt2pes and pesdata2ts:

```
txt2pes pages.txt 15 3600 1800 > txtpage.pes &
```

Txt2pes will generate PES packets from TXT packets, the first parameter 15 is how many TXT packets should be inserted in a single PES packet.

This information is related to the PES packet time information and also bit rate. Teletext has maximum number of packet per frame that is actually split in halft per field.

3600 is the number of PTS clocks for the first PES packet while 1800 is the increment of the PTS clock every PES packet, 1800 is actually 50 PES packet per second, so the result is the command line states that every 15 TXT packet a new PES packet will be generated and every PES packet matches a video field.

```
pesdata2ts txtpage.pes 1978 > txt.ts &
```

Pesdata2ts, the second tool we are using, will just encapsulate PES packets in TS packet with the given PID, in this case is 1978.

To know the bit rate for the teletext transport stream math is easy, after starting to execute txt2pes it will report about PES packet size, for example:

“pes packet size without 6 byte header is 1144 ”

this means:

1144 + 6 = 1150 bytes per packet

1150 / 184 (TS packet payload) = 7 TS packet every PES packet

7 * 188 * 8 -> 10528 per PES

if PES goes at 3600 means: 10528 * (90000 / 3600) -> 263200 bps

For the previous example we have:

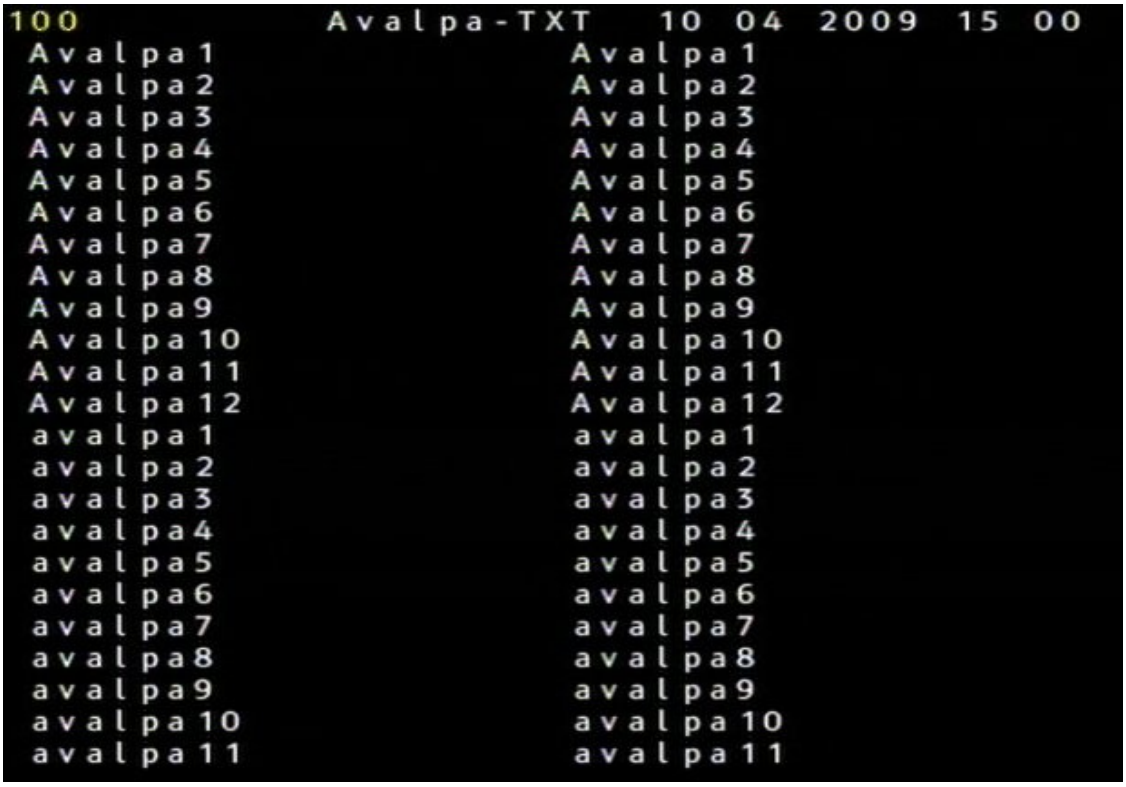
730 + 6 -> 736 / 184 -> 4 * 188 * 8 -> 6016 * (90000 / 1800) -> 300800 bps

So the final processes layout assuming the teletext packets are already generated and

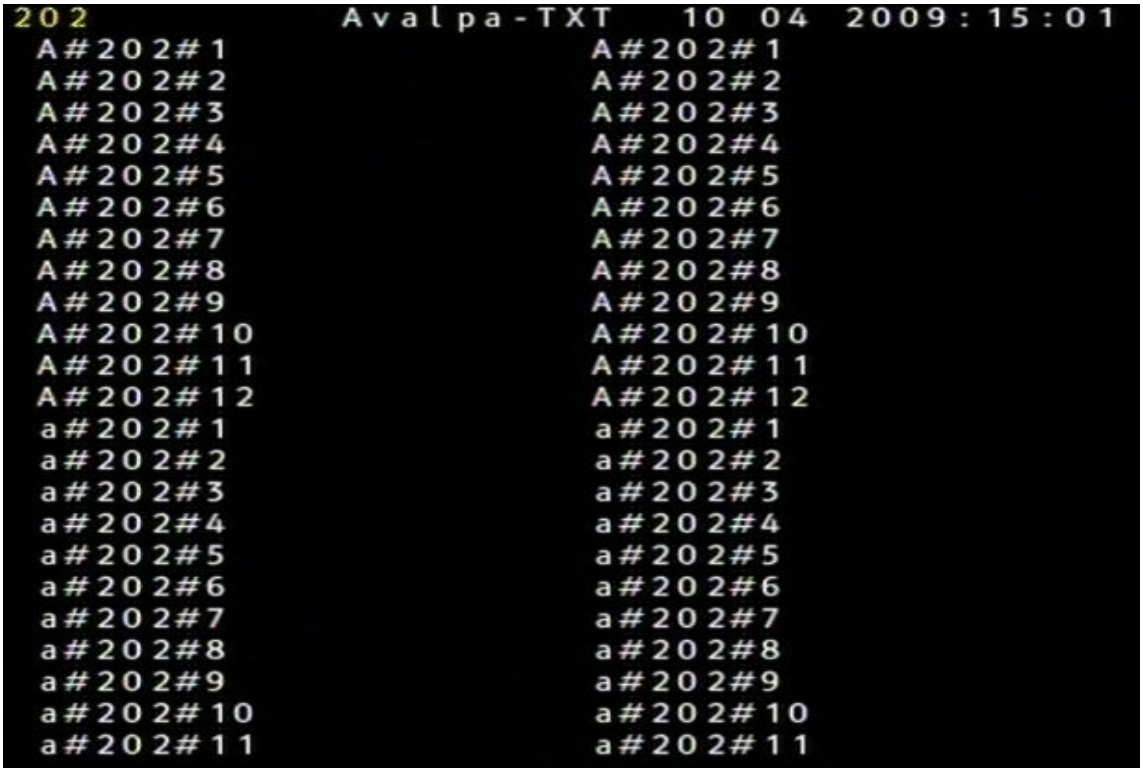
stored in pages.txt file is:

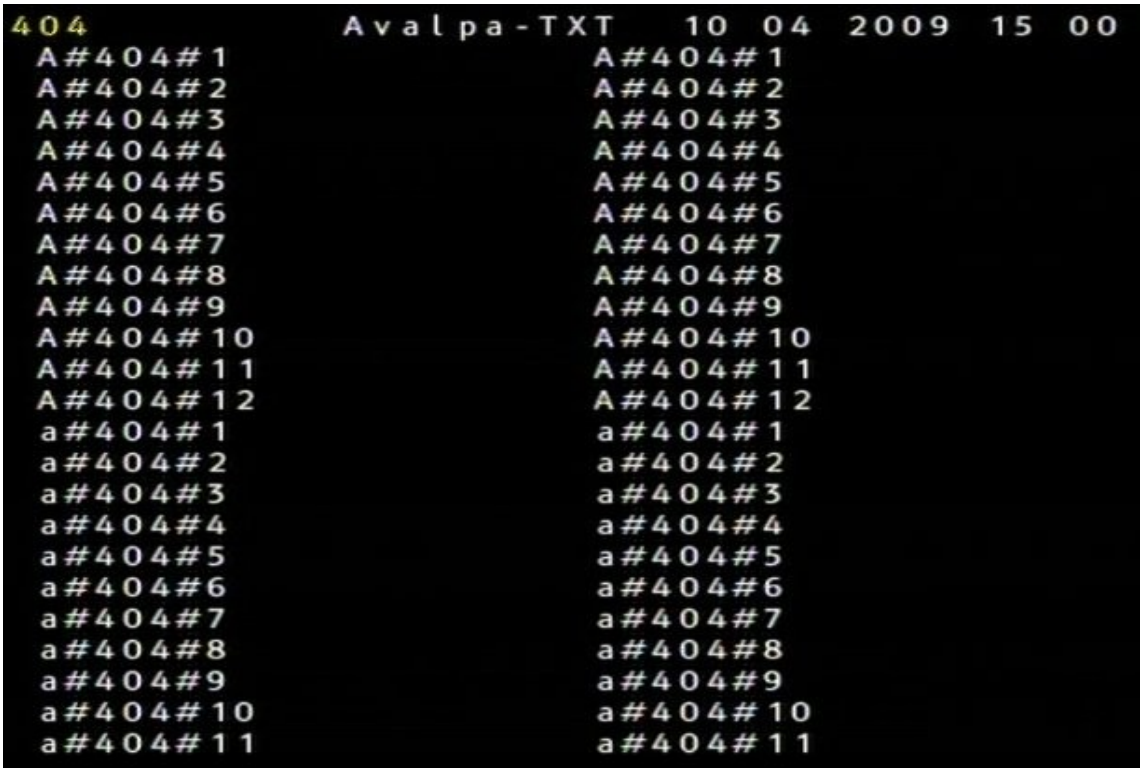
```
txt2pes pages.txt 15 3600 1800 > txtpage.pes &
pesdata2ts txtpage.pes 1978 > txt.ts &
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:28008 pat.ts b:28008 txtpmt.ts b:26500
sdt.ts b:1400 nit.ts b:300800 txt.ts b:10398284 null.ts > muxed.ts &
tsstamp muxed.ts 13271000 > stamped.ts &
tsrfsend stamped.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

The tutorial is transmitting 4 pages: 100, 202, 302 and 404 Page 100 has also fast link for the others pages connected to colour keys. The final look is:



About the teletext packet generation it can be done with python libraries but requires knowledge of the teletext itself. Basic information is presented here and will allow you to properly display text only pages.


202 Avalpa-TXT 10 04 2009:15:01
A#202#1 A#202#1
A#202#2 A#202#2
A#202#3 A#202#3
A#202#4 A#202#4
A#202#5 A#202#5
A#202#6 A#202#6
A#202#7 A#202#7
A#202#8 A#202#8
A#202#9 A#202#9
A#202#10 A#202#10
A#202#11 A#202#11
A#202#12 A#202#12
a#202#1 a#202#1
a#202#2 a#202#2
a#202#3 a#202#3
a#202#4 a#202#4
a#202#5 a#202#5
a#202#6 a#202#6
a#202#7 a#202#7
a#202#8 a#202#8
a#202#9 a#202#9
a#202#10 a#202#10
a#202#11 a#202#11


404 Avalpa-TXT 10 04 2009 15 00
A#404#1 A#404#1
A#404#2 A#404#2
A#404#3 A#404#3
A#404#4 A#404#4
A#404#5 A#404#5
A#404#6 A#404#6
A#404#7 A#404#7
A#404#8 A#404#8
A#404#9 A#404#9
A#404#10 A#404#10
A#404#11 A#404#11
A#404#12 A#404#12
a#404#1 a#404#1
a#404#2 a#404#2
a#404#3 a#404#3
a#404#4 a#404#4
a#404#5 a#404#5
a#404#6 a#404#6
a#404#7 a#404#7
a#404#8 a#404#8
a#404#9 a#404#9
a#404#10 a#404#10
a#404#11 a#404#11

Every pages is made of teletext packets, each packet is an unit. The first packet is the header line packets and looks like this:

```
                        EBUTeletext(
                                data_unit_id = 0x02, # 0x02 non-subtitle, 0x03 subtitles, 0xFF
stuffing
                                field_parity = 01, # which field odd or even?
                                line_offset = 0x7, # this is the first useful analog line to insert
the information
                                magazine = 0x04, # here we describe page 404
                                row = 0x00, # 24 (0x17) lines per magazine, 0 is the page header
                                page = 0x04,
                                subpage = 0x0000,
                                erase_page = 0, # no special request for this page
                                newsflash = 0,
                                subtitle = 0,
                                suppress_header = 0,
                                update_indicator = 1,
                                interrupted_sequence = 0,
                                inhibit_display = 0,
                                magazine_serial = 1,
                                country_code = 0x03,
                                chars = "    Avalpa-TXT  10 04 2009 15 00" # the header itself
                        ),
```

Packets belonging the the page will report only the magazine number 4, so this can lead to wrong pages is some pes packets are lost, anyway a typical teletext text line looks like:

```
                        EBUTeletext(
                                data_unit_id = 0x02,
                                field_parity = 01,
                                line_offset = 0x8,
                                magazine = 0x04,
                                row = 0x01, # the first teletext page row line, just after the header
                                chars = " A_404_1            " +
                                        " A_404_1            " ,
                        ),
```

DSMCC carousels and data casting

DSMCC is one of the standard to broadcast files over transport streams.

The most common uses are to broadcast decoder software upgrades DVB-SSU and/or applications for interactive television like MHP/OCAP(tru2way)/MHEG5/...

To implement carousels into OpenCaster architecture a directory has to be marshalled into a transport stream and the transport stream file need to be recreated every time the directory change, like PSI tables.

If the carousel is created properly you will just need to add it to all the other transport streams into your multiplexing loop.

To add more carousels you will need just to add them like any other transport stream file to the multiplexing.

DVB-SSU

[ssu]

The tutorial shows how to set-up a DVB-SSU update with OpenCaster. Many parameters regarding the client configuration are needed to be inserted so it's not expected to work out of the box for any decoder, it's just a skeleton useful for real use cases. Contact info@avalpa.com for more details.

To signal DVB-SSU you need to add a linkage descriptor to NIT, have a look to **ssu-data-generation.py**

```
nit = network_information_section(
        network_id = 1,
        network_descriptor_loop = [
            network_descriptor(network_name = "Avalpa",),
            linkage_descriptor(
                        transport_stream_id = avalpa_transport_stream_id,
                        original_network_id = avalpa_original_transport_stream_id,
                        service_id = 1,
                        linkage_type = 0x09,
                        OUI_loop = [
                                OUI_data (
                                        OUI = 0x15A, # any OUI
                                        selector_bytes = "",
                                )
                        ],
                        private_data_bytes = "",
                        ),
            ],
```

It is also necessary to add the dsmcc descriptor to the pmt:

```
pmt = program_map_section(
        program_number = avalpa1_service_id,
        PCR_PID = 8191,
        program_info_descriptor_loop = [],
        stream_loop = [
                stream_loop_item(
                        stream_type = 11, # data stream type
```

```
                        elementary_PID = avalpa1_dsmcc_pid,
                        element_info_descriptor_loop = [
                                stream_identifier_descriptor(
                                        component_tag = dsmccB_association_tag,
                                ),
                        ]
                )
        ],
```

For using more advanced DVB-SSU you will also need another descriptor in the pmt regarding a new table, the UNT:

```
pmt = program_map_section(
        program_number = avalpa1_service_id,
        PCR_PID = 8191,
        program_info_descriptor_loop = [],
        stream_loop = [
                stream_loop_item(
                        stream_type = 11, # data stream type
                        elementary_PID = avalpa1_unt_pid,
                        element_info_descriptor_loop = [
                                data_broadcast_id_descriptor(
                                        data_broadcast_ID = 0x000A, # DVB-SSU
                                        OUI_info_loop = [
                                                OUI_info_loop_item (
                                                        OUI = anyOUI,
                                                        update_type = 0x02, # with broadcasted UNT,
0x01 without UNT: it requires a stream_identifier_descriptor
                                                        update_versioning_flag = 0, # no version
change
                                                        update_version = 1, # increment this at
update change
                                                        selector_bytes = "",
                                                ),
                                        ],
                                        private_data_bytes = "",
                                ),
                        ]
                ),
                stream_loop_item(
                        stream_type = 11, # data stream type
                        elementary_PID = avalpa1_dsmcc_pid,
                        element_info_descriptor_loop = [
                                stream_identifier_descriptor(
                                        component_tag = dsmccB_association_tag,
                                ),
                        ]
                )
        ],
```

DVB-SSU carousel is simpler than interactive applications and just requires each file to be transmitted to be specified as a “module”, the last part of **ssu-data-generation.py** configuration file shows how to specify them, for a complete description of values check: ETSI TS 102 006

```
#
# DSMCC description
#

g1 = Group(
    PATH="DII-1.sec",
    transactionId = 0x80000002,
    downloadId    = 0x00000001,
    blockSize     = 4066,
```

```
    version       = 1,
    )
g1.set(
    compatibilityDescriptor = comp_desc1.pack(),
    modules = [
         Module(
            INPUT="ocdir1/RootRelook2100",
            moduleId = 0x0001,
            moduleVersion = 0x00,
            descriptors = [name_descriptor(name="a bin")],
        ),
         Module(
            INPUT="ocdir1/RootRelook2101",
            moduleId = 0x0002,
            moduleVersion = 0x00,
            descriptors = [name_descriptor(name="a bin")],
        ),

[…]


g2 = Group(
    PATH="DII-2.sec",
    transactionId = 0x80000004,
    downloadId    = 0x00000002,
    blockSize     = 4066,
    version       = 1,
    )
g2.set(
        compatibilityDescriptor = "hello",
        modules = [
         Module(
            INPUT="big-file.raw",
            moduleId = 0x0010,
            moduleVersion = 0x00,
            descriptors = [name_descriptor(name="another bin")],
            ),
    ],
    )

dsi = SuperGroup(
    PATH = "/",
    transactionId = 0x80000000,
    version       = 1,
    )
dsi.set(
    compatibilityDescriptor = "\000\000", # as specified in etsi 102 006
)
dsi.addGroup(g1)
dsi.addGroup(g2)
```

Finally a script will take care of convert all the generated sections to a transport stream file:

```
./ssu-update.sh datacarousel 2003 0
```

The script file gets the sections generated by **ssu-data-generation.py** a converts them to transport stream using sec2ts

The last part is the usual multiplexing carried out with these commands:

```
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts b:1000000 datacarousel.ts b:9774081 null.ts > fifomuxed.ts &
tstdt fifomuxed.ts > fifotimed.ts &
tspcrstamp fifotimed.ts 13271000 > fifostamped.ts &
tsrfsend stamped.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 514 &
```

Interactive TV support

Interactive television standards usually requires a data casting more complex than dvb-ssu so instead to describe modules one by one as shown in ssu-data-generation.py followed by ssu-update.sh an automatic tool will take care of all the processing:

Tool oc-update.sh

Usage:

```
oc-update.sh object_carousel_directory association_tag module_version dsmcc_pid carousel_id
[compress_mode] [padding_on] [clean_off] [DDB_size] [update_flag] [mount_frequency]
        - carousel_directory: the directory to marshal in an object carousel
        - association_tag aka common tag, referenced by PMTs and AITs, every carousel has one
        - modules_version, all the modules will have the same version, you need to change this to
notify to the box files are changed, goes from 0 to 15
        - pid, referenced by PMTs using this carousel
        - carousel_id, referenced by PMTs using this carousel, every carousel has its own, it is an
alternative for association_tag, they have the same function
        - compress_mode, 0: don't compress, 1:compress all, 2:smart compress, file with .solo
extension are set in an uncompressed module alone to allow use cases like quick image file update,
default is 2
        - padding_on, every section is padded, was useful with some buggy decoder, waste bandwith,
default off, unsupported since OpenCaster 2.4.8
        - clean_off, don't delete temp file, default off, used for debug
        - DDB_size, Use custom size for DDB payload, default = max = 4066
        - sets the Update flag in the TransactionID of DSI and DII to the value given (0 or 1)
        - mount_frequency, set how often insert DII/DSI/SGW to speed up carousel mount, default is
twice per carousel period
```

Here is an example, assuming ocdir1 is the directory you want to broadcast:

```
oc-update.sh ocdir1 0xB 0x5 2003 7 1 0 0 4066 0 2
```

will generate ocdir1.ts than can be muxed to all the other transport stream files, the parameters passed specifies that:

- the carousel directory to marshal is: ocdir1
- the association_tag and/or component_tag to use for PMT and AIT is: 0xB (11)
- the version of the modules and sections of the carousel is: 5
- the pid of the stream is: 2003
- the id of the carousel to use in the PMT is: 7
- the carousel will be compressed as much as possible (it can prevent update notify on buggy decoder)
- no padding
- delete temp files
- default block size is 4066 (maximum)
- the update flag is set off
- the head of the carousel (DSI/DII/SGW will be sent once at carousel start and once

at carousel middle)

Running again the command with a different version number will generate a new ocdir1.ts marshalling again the directory

```
oc-update.sh ocdir1 0xB 0x6 2003 7 1 0 0 4006 1 2
```

MHP/MHEG5 signalling

[mhp]

Now that we have learnt how to marshal a file system, the big thing is that in the file system we can bring to an interactive decoder the applications. We need to insert the Application Information Table signalling if we want the interactive decoders to be aware of applications and we also need to properly set up DSMCC references.

The complete basic workflow at decoder side for receiving applications is:

1) the decoder tunes the channel
2) the decoder parses the AITs signalled by the PMT
3) the decoder presents to the user the application list got from the AITs (it can be necessary to press the "app" button on the remote control)
4) the user selects an application
5) the decoder loads the DSMCC carousel referenced by the application descriptor or connect to the remote http server
6) the decoder executes the application

So let's have a look at mhpconfig.py on how to active this, first of all the PMT for the required referencing of AIT and DSMCC streams looks like this:

```
#
# Program Map Table (ISO/IEC 13818-1 2.4.4.8)
# this is PMT with DSMCC and AIT descriptor for MHP interactive applications
#

pmt = program_map_section(
    program_number = avalpa1_service_id,
    PCR_PID = 2064,
    program_info_descriptor_loop = [],
    stream_loop = [
        stream_loop_item(
            stream_type = 2, # mpeg2 video stream type
            elementary_PID = 2064,
            element_info_descriptor_loop = []
        ),
        stream_loop_item(
            stream_type = 3, # mpeg2 audio stream type
            elementary_PID = 2068,
            element_info_descriptor_loop = []
        ),
        stream_loop_item(
            stream_type = 5, # AIT stream type
            elementary_PID = 2001,
            element_info_descriptor_loop = [
                application_signalling_descriptor(
                    application_type = 1, # 1 DVB-J application, 2 DVB-HTML
```

```
                    AIT_version = 1,  # current ait version
                ),
            ]
        ),
        stream_loop_item(
            stream_type = 11, # DSMCC stream type
            elementary_PID = 2003,
            element_info_descriptor_loop = [ # a number of descriptors specifying DSMCC properites
                association_tag_descriptor(
                    association_tag = 0xB,  # this association tag identifys the carousel, it is
used also while generating the DSMCC with oc-update.sh and referenced by the AIT
                    use = 0,  # some default values follow, don't change them
                    selector_lenght = 0, # ...
                    transaction_id = 0x80000000, # ...
                    timeout = 0xFFFFFFFF, # ...
                    private_data = "",
                ),
                stream_identifier_descriptor(
                    component_tag = 0xB, # it is the same as the assocation tag, some decoders will
look for the component tag, others for the association tag, the same value should be used
                ),
                carousel_identifier_descriptor(
                    carousel_ID = 1, # carousel id number, it's a different number from
association/component tag, but it has a similiar purpouse: identifying the carousel
                    format_ID = 0, # no enhanced boot supported
                    private_data = "",
                ),
                data_broadcast_id_descriptor(
                    data_broadcast_ID = 240, # 240 is the code specifying this is DSMCC-MHP , 262
for mheg5
                    ID_selector_bytes = "", # for mheg5 you need selector bytes "\001\001\000\000",
                ),
            ]
        )
    ],
[...]
```

N.B. To add another AIT stream or another DSMCC you will have to add more stream_loop_items !

In mhpconfig.py you can also find an AIT example that will generate an firstait.ts transport stream file, let's have a look at it:

```
# Application Informaton Table (ETSI TS 101 812 10.4.6)
ait = application_information_section(
    application_type = DVB_J_application_type,
    common_descriptor_loop = [],
    application_loop = [
        # here we list only 1 application, adding another application loop item will signal a second
application in the sam AIT, you can signal applications in the same AIT or more AITs
        application_loop_item(
            organisation_id = 10,  # this is a demo value, dvb.org should assign an unique value
            application_id = 1001, # should be unique for every organisation id in the same program
            application_control_code = 2, # is PRESENT, the decoder will add this application to the
user choice of application, 1 is AUTOSTART, the application will start immedtiatly to load and to
execute, 3 is DESTROY, it will signal to the application to stop executing, is KILL, it will stop
execute the application
            application_descriptors_loop = [
                transport_protocol_descriptor(
                    protocol_id = MHP_OC_protocol_id, # the application is broadcasted on MHP-DSMCC
                    transport_protocol_label = 1, # carousel id
                    remote_connection = 0,
                    component_tag = 0xB, # carousel common tag and association tag
                ),
                application_descriptor(
                    application_profile = 0x0001, # Profile and MHP version
                    version_major = 1,
                    version_minor = 0,
                    version_micro = 2,
```

```
                    service_bound_flag = 1, # the application is expected to die on service change,
0 will wait after the service change to receive all the AITs and check if the same app is signalled
or not
                    visibility = 3, # the applications is visible to the user, 1 the application is
visible only to other applications
                    application_priority = 1, # is lowset, when more than 1 app. is executing
                    transport_protocol_labels = [1], # carousel Id
                ),
                application_name_descriptor(application_name = "Text input example"),
                dvb_j_application_descriptor(parameters = ["file://1/sample8"]), # xlet parameters
                dvb_j_application_location_descriptor(
                    base_directory = "/", # base directory, if set to "/hello" the xlet will act as
"/hello" is its root directory
                    class_path_extension = "", # an additiona classpath inside the carousel
                    initial_class = "Wizard",  # the starting class
                ),
            ],
        ),
    ],
    )
```

Now execute:

```
./mhpconfig.py
```

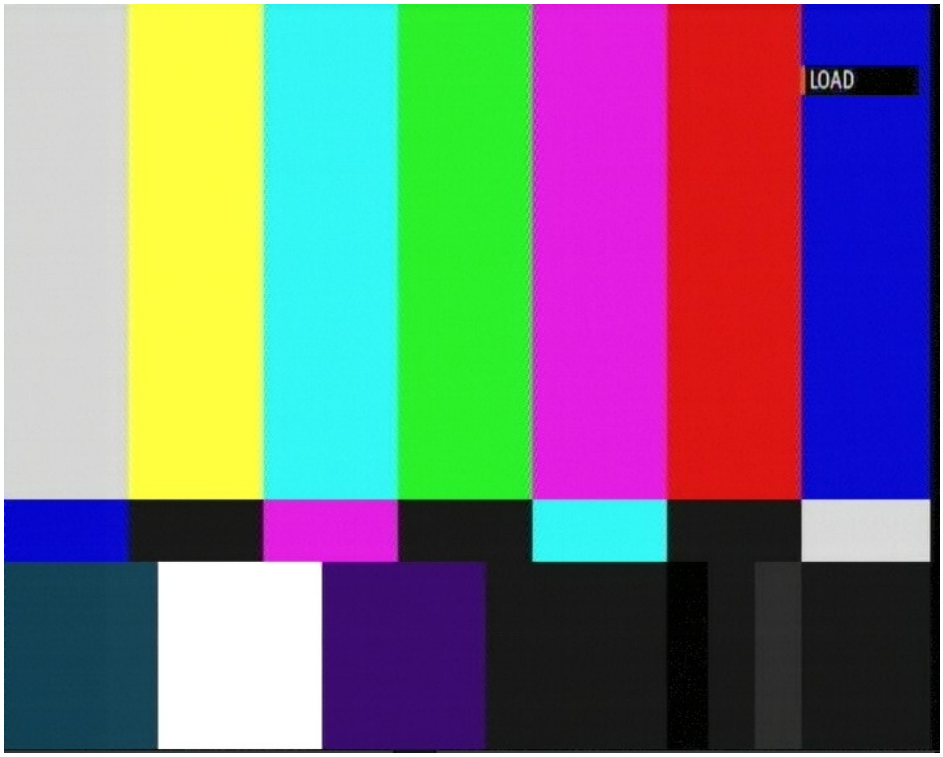
to generate firstait.ts and others signalling tables and execute

```
oc-update.sh ocdir1 0xB 5 2003 1 2 0 0 4066 1 2
```

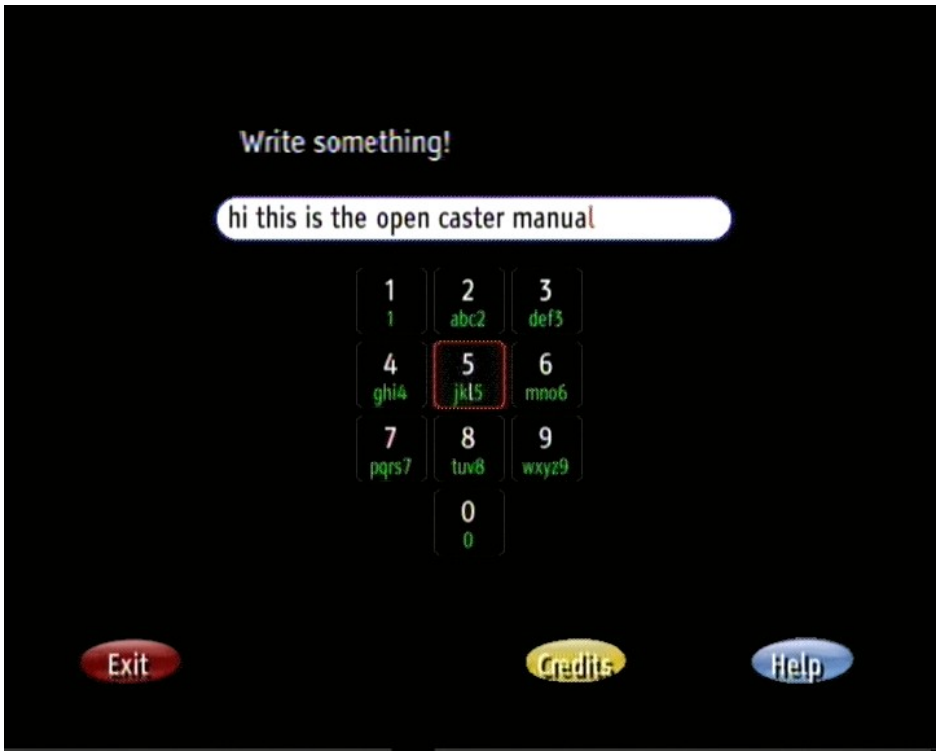
to generate ocdir1.ts so you are ready to multiplex your first Single Program Transport Stream with an interactive application:

```
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts b:1000000 ocdir1.ts b:2000 firstait.ts b:9772084 null.ts>
myfirstfifo.ts &
tsstamp myfirstfifo.ts 13271000 > mysecondfifo.ts &
tsrfsend mysecondfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

Now, to test it, proceed to the decoder tuning and select the example application:



Here you can see the xlet during execution.



HbbTv signalling

[hbbtv], [hbbtv-dsmcc]

An example of HbbTv signalling linking a http:// application is also available and the example is similar to MHP/MHEG5 but no DSMCC, let's have a look the the ait table:

```
# parameters reported into ths AIT to signalize a broadband application.
appli_name = "application Portail" #application name
appli_root = "http://192.168.1.1/portail/" #URL base of transport_protocol_descriptor
appli_path = "index.html"  #initial_path_bytes of simple application descriptor
organisationId_1 = 10  # this is a demo value, dvb.org should assign an unique value
applicationId_1 = 1001 # this is a demo value. This number corresponds to a trusted application
applicationId_2 = 1002

ait = application_information_section(
        application_type = 0x0010,
        common_descriptor_loop = [
              external_application_authorisation_descriptor(
                    application_identifiers = [[organisationId_1,applicationId_1],
[organisationId_1,applicationId_2]],
                    application_priority = [5, 1]
                    )
              ],
        application_loop = [
              application_loop_item(
                    organisation_id = organisationId_1,
                    application_id = applicationId_1,
                    application_control_code = 1, #AUTOSTART
                    application_descriptors_loop = [
                          transport_protocol_descriptor(
                                protocol_id = 0x0003,
                                URL_base = appli_root,
                                URL_extensions = [],
                                transport_protocol_label = 3,
                          ),
                          application_descriptor(
                                application_profile = 0x0000,
                                      #0x0000 basic profile
                                      #0x0001 download feature
                                      #0x0002 PVR feature
                                      #0x0004 RTSP feature
                                version_major = 1, # corresponding to version 1.1.1
                                version_minor = 1,
                                version_micro = 1,
                                service_bound_flag = 1, # the application is expected to die on
service change
                                visibility = 3, # the applications is visible to the user
                                application_priority = 1,
                                transport_protocol_labels = [3], # If more than one protocol is
signalled then each protocol is an alternative delivery mechanism. The ordering indicates the
broadcaster's view of which transport connection will provide the best user experience (first is
best)
                          ),
                          application_name_descriptor(
                                application_name = appli_name,
                                 ISO_639_language_code = "fra"
                          ),
                          simple_application_location_descriptor(initial_path_bytes = appli_path
),
                    ]
              ),
```

OCtutorail17 shows an example where the HbbTV application is sent over DSMCC

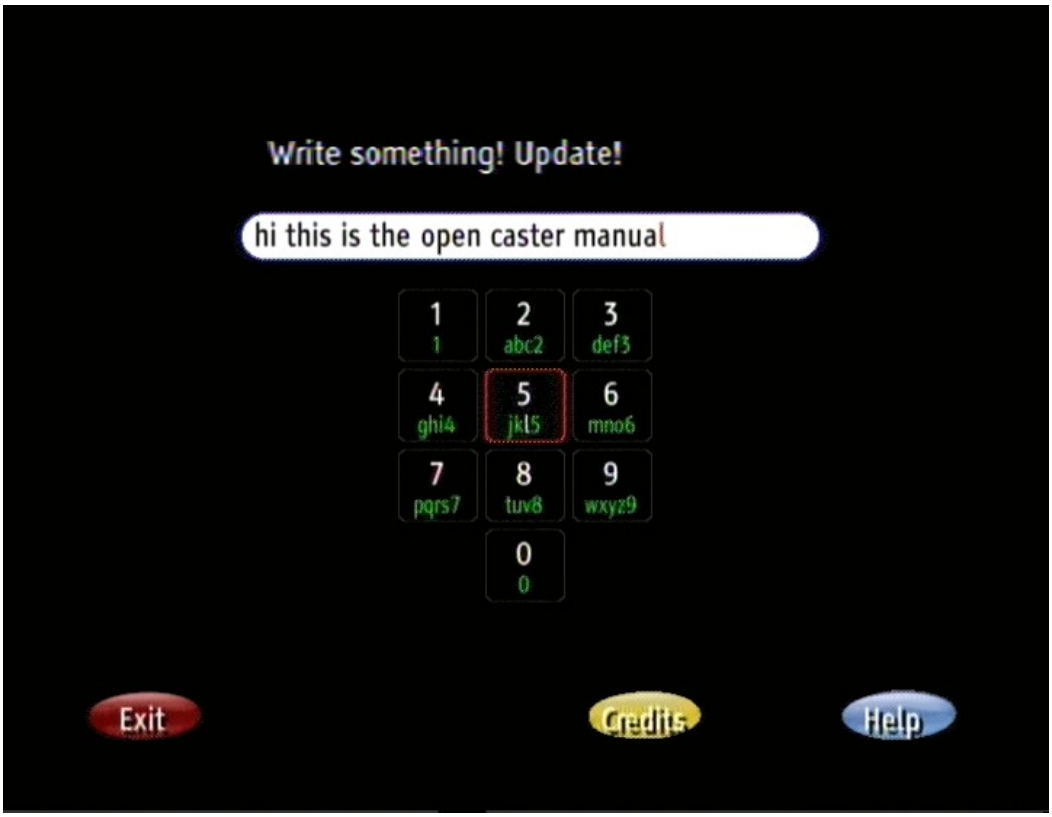
Update an object carousel with running applications

[mhp]

A usefull feature of DSMCC is that a running application can monitor a file on the DSMCC and be notified if the file is changed after it was transmitted and received at least once.The basic notion is that every file of the DSMCC is in a module, and every module is in a section, so we need to update the section version as for all the PSI tables. If the carousel was generated with version 5 as in the previous chapter, try to edit the text from ocdir1/change_text file, in our example we changed from “Write something!” to “Write something! Update!”. You can execute the example as in the previous chapter and after you change the file content you will just need to re run oc-update.sh with the new version:

```
oc-update.sh ocdir1 0xB 6 2003 1 2 0 0 4066 0 2
```

when the multiplexing is running and wait a few moments, here how it looks after update:



An interesting features of Linux kernel since 2.6.13 is the capability to notify filesystem changes to the application, so it is pretty easy to setup a bash script for auto-update using inotify-tools like this:

```
VER=1
while inotifywait -rq -e close_write ocdir1; do
        let "VER+=1"
        let "VER&=15"
        DATE=`date `
        echo "[$DATE]: updating ocdir1.ts filesystem to ver: $VER."
        oc-update.sh ocdir1 0xB $VER 2003 1 2 0 0 4066 0 2
done
```

Signalling Stream Events

[mhp-streamevents]

Have you ever asked how can you synchronize a running application to a video program on-going. How can you send a signal to an interactive application at an exact point in time, when that event is really happening on the video stream?

The answer is: Stream Event “do it now”!

This advanced topic chapter regards exactly how to signal interactive applications and Stream Event management.

Stream events are signals sent on a particular PID to the interactive application and they are actually defined as all the other tables, have a look at the Stream Event defined in mhpconfig2.py:

```
[...]
#
# Stream Event
#

ste = stream_event_section(
    event_id = 1,
    stream_event_descriptor_loop = [
        stream_event_do_it_now_descriptor(
            event_id = 1,
            private_data = "event 1 private data",
        ),
    ],
    version_number = 1,
    section_number = 0,
    last_section_number = 0,
    )
[...]
```

This event has id 1 and brings some private date in the payload: "event 1 private data".

This event is a Do It Now Event, the only kind of stream event supported by OpenCaster and actually the only one supported among all MHP decoders.

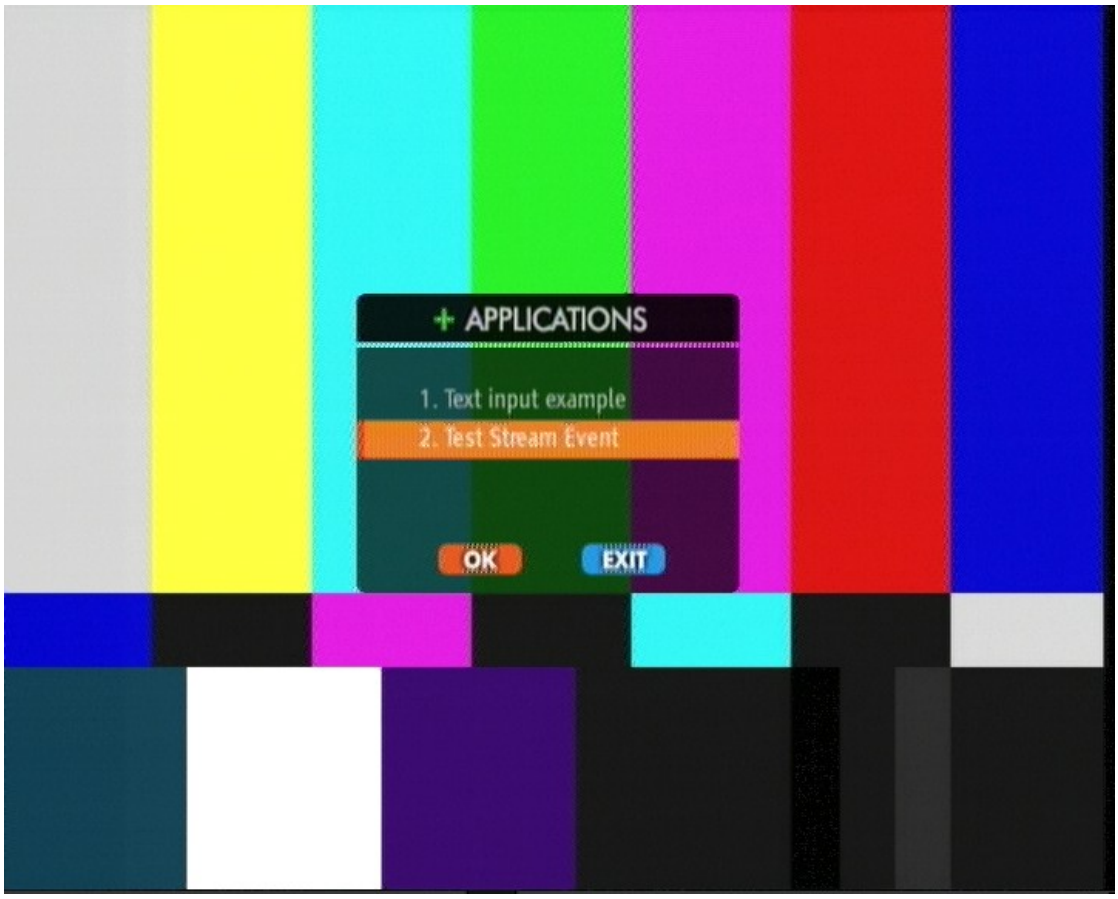
"Do It Now" means that as soon as the decoder receive the event it should be sent to the applications waiting for it.

Also HbbTv supports stream events with the same generation process.

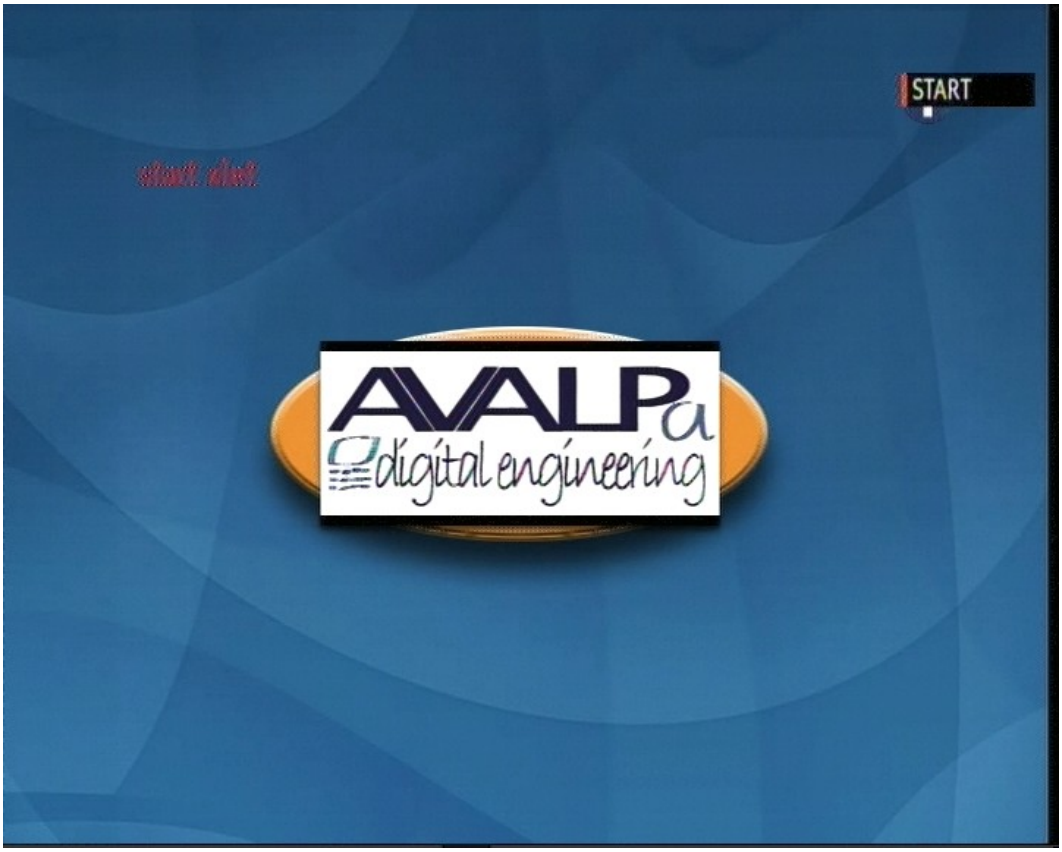
To have the demo running execute for mhp-streamevents:

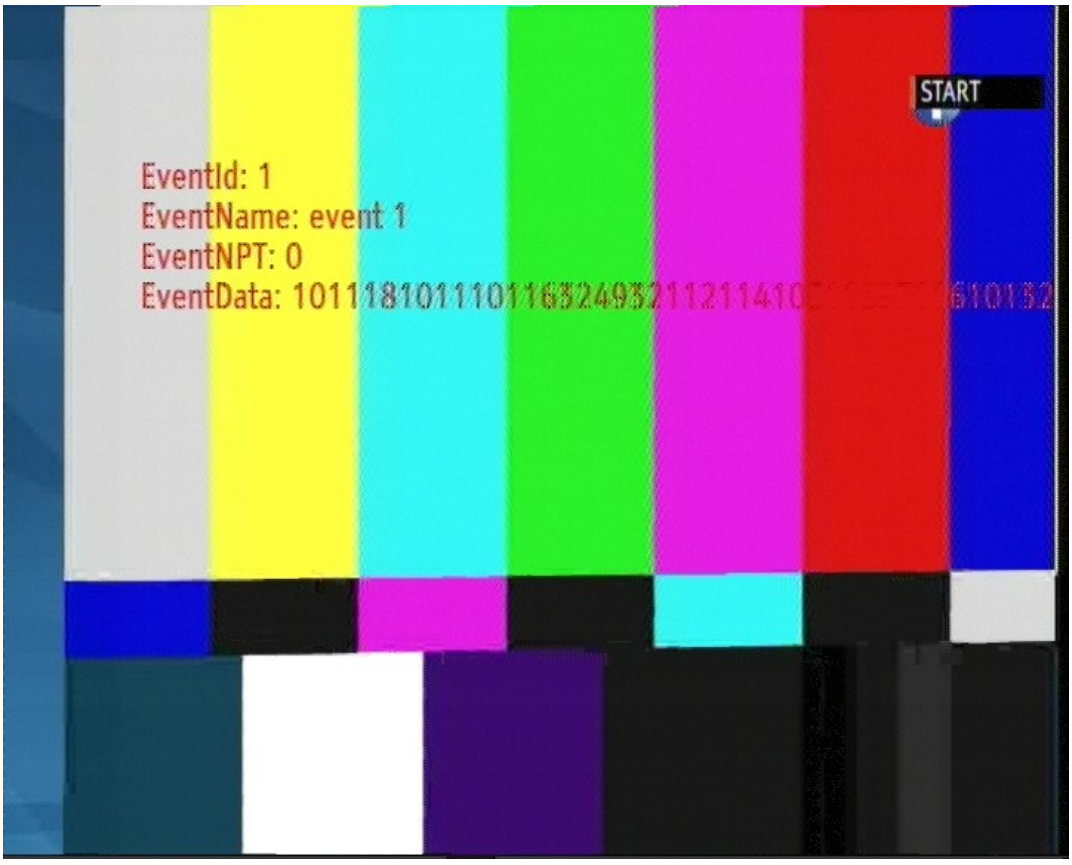
```
./mhpconfig2.py
oc-update.sh ocdir1 0xB 5 2003 1 2 0 0 4066 0 2
oc-update.sh ocdir2 0xC 5 2004 2 2 0 0 4066 0 2
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts b:1000000 ocdir1.ts b:1000000 ocdir2.ts b:2000 firstait.ts
b:2000 firstste.ts b:8770084 null.ts > myfirstfifo.ts &
tsstamp myfirstfifo.ts 13271000 > mysecondfifo.ts &
tsrfsend mysecondfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

Now you can tune the decoder and look for the application "Test Stream Event":



If you execute the application, it will start and wait for the Stream Event:


START
AVALPa
digital engineering


START
EventId: 1
EventName: event 1
EventNPT: 0
EventData:

If you didn't notice the multiplexing is using also a firstste.ts file, this file was generated by ./mhpconfig2.py and contains the events themselves sent in a loop.

Obviously this is not the usual behaviour because you don't want to send stream event from the very beginning, so you will need to initially mux a null.ts instead of firstste.ts and replace the stream when it's the right time to send the event:

```
cp null.ts tempste.ts
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008 firstpat.ts b:3008 firstpmt.ts
b:1500 firstsdt.ts b:1400 firstnit.ts b:1000000 ocdir1.ts b:1000000 ocdir2.ts b:2000 firstait.ts
b:2000 firstste.ts b:8770084 null.ts > myfirstfifo.ts &
tsstamp myfirstfifo.ts 13271000 > mysecondfifo.ts &
tsrfsend mysecondfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
cp firstste.ts tempste.ts
```

to stop sending the event just put the null.ts back

```
cp null.ts tempste.ts
```

So far so good, but here it comes the difficult part: how the application register to wait for the event.

The application, in order to receive the events, has to start executing and register itself to the events.

To register to event the application needs a reference object placed in the object carousel so it's necessary to create a Stream Event Object into the file system you want to broadcast, that's possible with: steo.py

Have a look at the steo.py script, it will generate three files: .eid .ename .tap, if you put these three in a directory inside the object carousel like: test.event or example.event the carousel generator will create a Stream Event Object instead of a directory. This is a special in-band signaling.

```
[...]
tap = str_event_use_tap()
tap.set(
    id = 0,
    assocTag = 0xD, # association tag defined into PMT for the Stream Event
   )

taps = Taps (
   taps_count = 1,
   tap_loop = [ tap,],
  )

event_count = 3 # number of events
event_names = Event_names (
   eventnames_count = event_count,
   event_name_loop = [ "event 1", "event 2", "event 3"],  # name of the events
  )
event_ids = Event_ids (
        eventids_count = event_count,
        event_id_loop = [ 1, 2, 3,], # id of the events
    )
[...]
```

Here is how it looks like in ocdir2 the generated files .eid, .ename and .tap:

```
lorenzo@nb-lpallara:~/OpenCaster/example-config$ ls -la ocdir2/
total 20
drwxr-xr-x 3 lorenzo lorenzo 4096 2007-12-04 12:17 .
drwxr-xr-x 5 lorenzo lorenzo 4096 2008-06-10 02:06 ..
```

```
-rw-r--r-- 1 lorenzo lorenzo   21 2007-12-04 12:17 here_is_carousel2
drwxr-xr-x 2 lorenzo lorenzo 4096 2007-12-04 12:17 test.event
-rw-r--r-- 1 lorenzo lorenzo 3796 2007-12-04 12:17 Testste.class

lorenzo@nb-lpallara:~/OpenCaster/example-config$ ls -la ocdir2/test.event/
total 20
drwxr-xr-x 2 lorenzo lorenzo 4096 2007-12-04 12:17 .
drwxr-xr-x 3 lorenzo lorenzo 4096 2007-12-04 12:17 ..
-rw-r--r-- 1 lorenzo lorenzo    7 2007-12-04 12:17 .eid
-rw-r--r-- 1 lorenzo lorenzo   29 2007-12-04 12:17 .ename
-rw-r--r-- 1 lorenzo lorenzo    8 2007-12-04 12:17 .tap
```

When the applications querys the test.event object it will find out that can register to three different events: 1, 2, 3 as described by steo.py

These three event are sent over the PID specified into the PMT and into steo.py, look for the "component tag" both in steo.py and mhpconfig2.py

```
[...]
stream_loop_item(
    stream_type = 12, # Stream Event stream type
    elementary_PID = ste1_pid,
    element_info_descriptor_loop = [
        stream_identifier_descriptor(
            component_tag = 0xD,
        ),
    ]
 ),
[...]
```

How to receive stream events on a MHP application is out of the scope of this document, but you will find a java example application in the OpenCaster package.

The complete workflow after the application is started is:

1) the decoder executes the application TestSte
2) the application looks for "test" Stream Event Object generated by steo.py and placed into the DSMCC at generation time
3) the application subscribe to event 1
4) the application receives event 1 from the firstste.ts generated by mhpconfig2.py

DSMCC-receive

[mhp]

The dsmcc-receive tool will extract a dsmcc file system from a transport stream file.

```
mkfifo output.sec
mkdir outputdir
ts2sec ocdir1.ts 2003 > output.sec &
dsmcc-receive outputdir 100 2003 0xB < output.sec
```

The first command will extract sections from a given PID (2003 in the example) into a transport stream file, the second will output the dsmcc filesystem in the outputdir directory however to process the dsmcc it needs to know the PID (2003) and the component tag (0xB).

100 is a cache value, it specifies that it can store up to 100 sections even if it's not sure if they belong to the current dsmcc or not.

Multiplexing a Multiple Program Transport Stream from local files

[mpts]

To broadcast two or more programs at the same time (the very concept of a multiplex creating a “multi program transport stream” AKA MPTS), you basically add more audio.ts and more video.ts file to tscbrmuxer and signal the streams with the psi table, for a complete example download:

These python scripts ends with a system command invoking sec2ts so they directly generates .ts files without the need to execute manually sec2ts so execute them to generate the psi .ts files.

Audio and video are already multiplexed into mptsav*.ts files.

Have a look at mptsconfig.py, execute it with ./mptsconfig.py and then start muxing and playing: (remember you need a fifo: mkfifo myfirstfifo.ts)

```
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:2300000 secondvideo.ts b:188000
secondaudio.ts b:2300000 thirdvideo.ts b:188000 thirdaudio.ts b:3008 mptspat.ts b:3008 mptspmt1.ts
b:3008 mptspmt2.ts b:3008 mptspmt3.ts b:1400 mptsnit.ts b:1500 mptssdt.ts b:5792069 null.ts >
myfirstfifo.ts &
tsrfsend myfirstfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

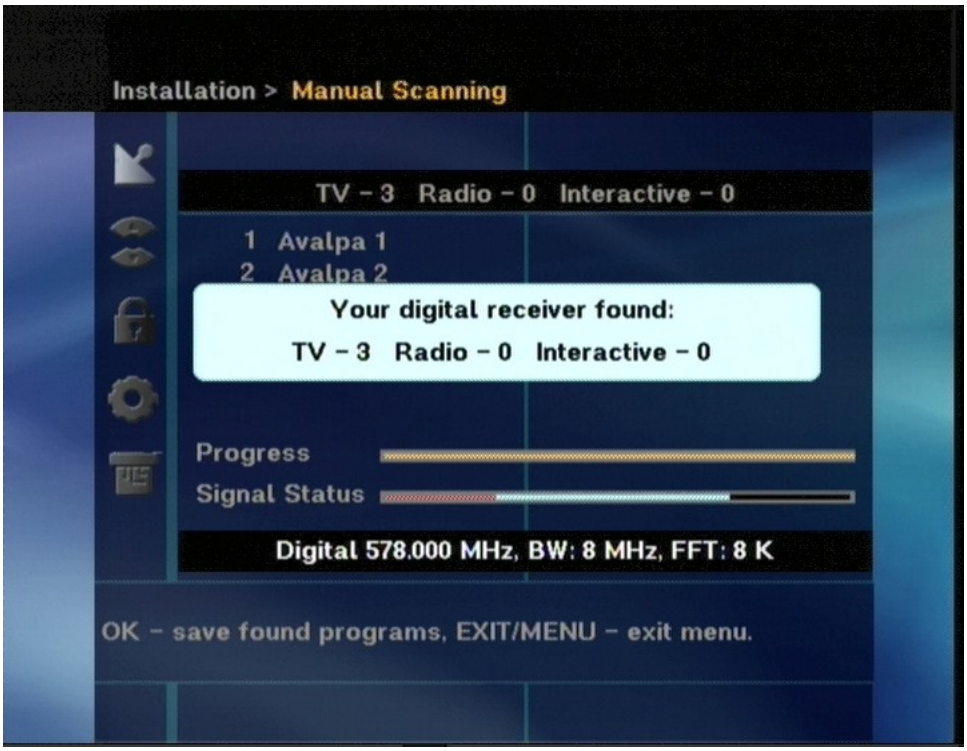
The clip files are very short and with a very small bit rate, many decoders can have problems decoding channel 2 and channel 3, channel 1 should play smooth on all decoders, this can also be a test on your mpeg2 decoder recovery capabilities.

A problem arises because the videos are looped and so, from the point of view of the decoder, the PCR goes back in time after the end of the first loop when we come back playing the same ts loop a second time; you can try to fix PCR using tsstamp like here:

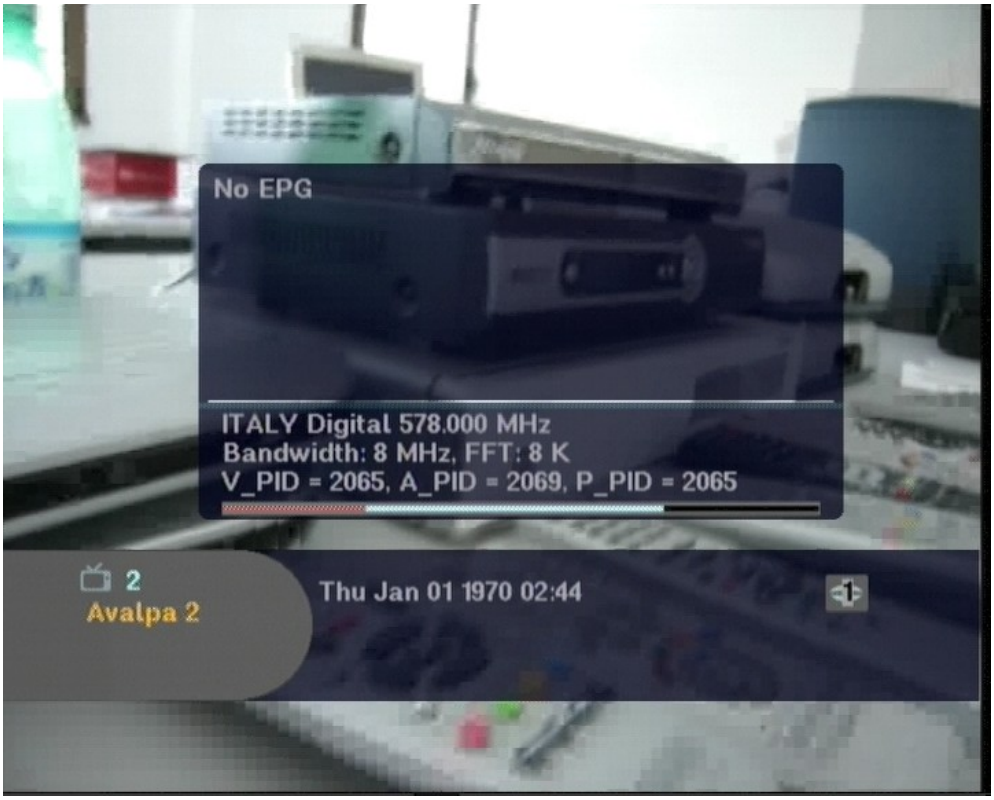
```
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:2300000 secondvideo.ts b:188000
secondvideo.ts b:2300000 thirdvideo.ts b:188000 thirdaudio.ts b:3008 mptspat.ts b:3008 mptspmt1.ts
b:3008 mptspmt2.ts b:3008 mptspmt3.ts b:1400 mptsnit.ts b:1500 mptssdt.ts b:5792069 null.ts >
myfirstfifo.ts &
tsstamp myfirstfifo.ts 13271000 > mysecondfifo.ts &
tsrfsend mysecondfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

Please note that input files are generated by OpenCaster itself, check re-multiplexing section for multiplex inputs from other systems.

Some screenshots from the decoder will follow:

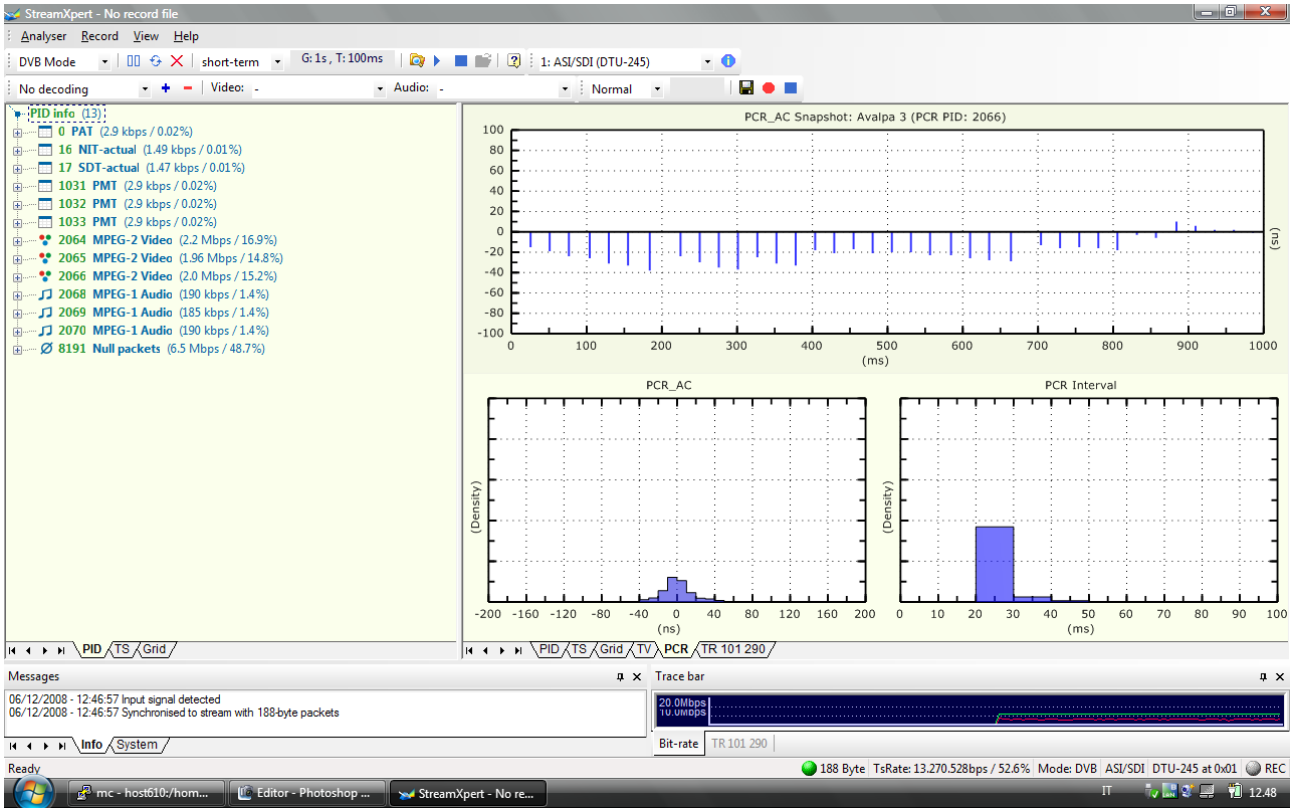

Installation > Manual Scanning
TV – 3 Radio – 0 Interactive – 0
1 Avalpa 1
2 Avalpa 2
Your digital receiver found:
TV – 3 Radio – 0 Interactive – 0
Progress
Signal Status
Digital 578.000 MHz, BW: 8 MHz, FFT: 8 K
OK – save found programs, EXIT/MENU – exit menu.

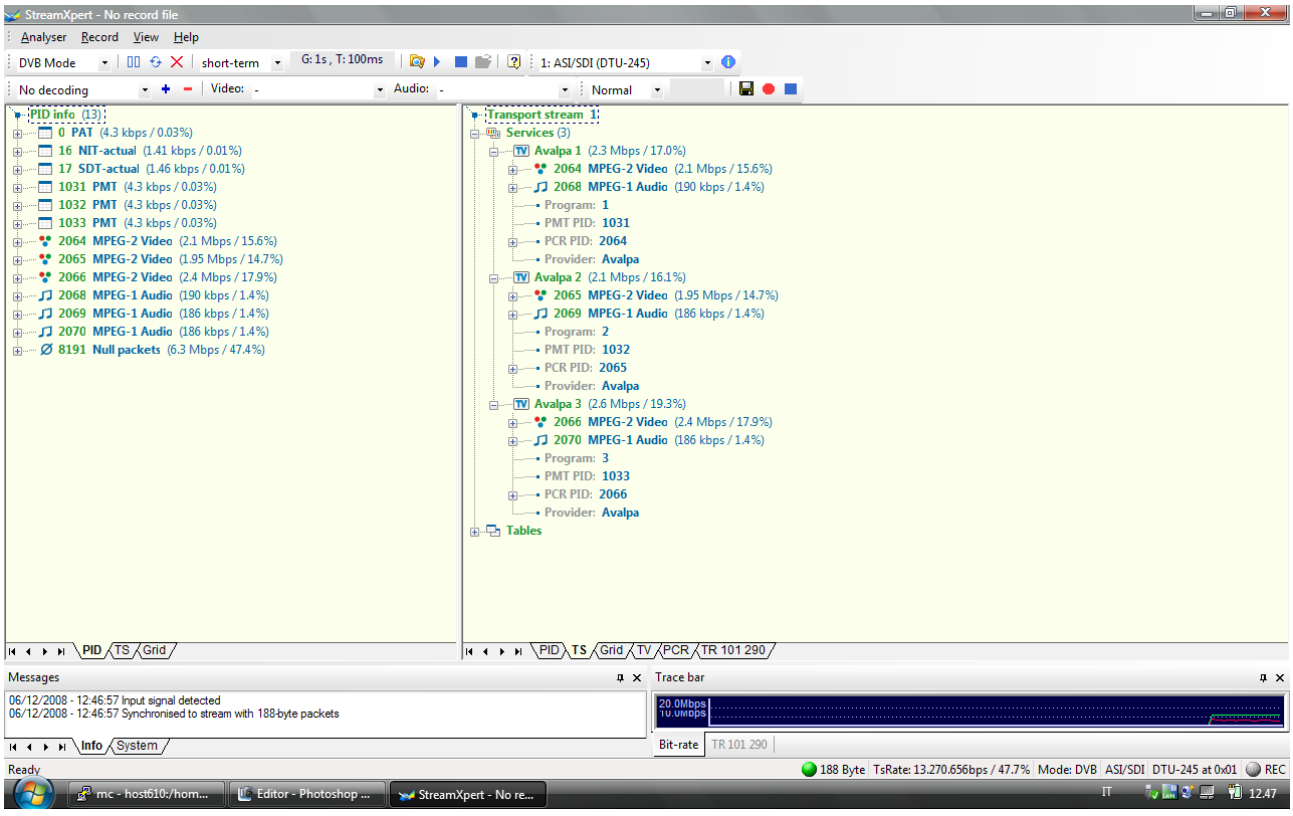

TV List
1 Avalpa 1
2 Avalpa 2
3 Avalpa 3



Finally let's check the analyser again for the services and their PCR:

Avalpa Broadcast Server user manual





BISS-E and CSA output support

[csa]

Avalpa Broadcast Server can scramble the transport stream using CSA and support BISS-E mode 1.

Tscrypt will scramble your transport stream, however it requires libdvbcsa provided by third parties and not installed on the server by default.

To install the software proceed as follow:

```
mkdir libdvbcsa
cd libdvbcsa
svn co svn://svn.videolan.org/libdvbcsa
cd libdvbcsa/trunk
autoconf
./configure
make
su (insert root password, default is Avalpa)
make install
exit
cd /home/OpenCaster/software/tools/tscrypt
make
su (insert root password, default is Avalpa)
make install
exit
```

Now you are ready to use tscrypt, the basic usage is:

```
tscrypt input.ts file.cw > crypted.ts
```

the input transport stream will be crypt using control words from file.cw, those are 8 bytes twice, odd and even keys, it is a perfect match for a biss cam.

The PSI signalling required is carried out in the pmt:

```
pmt = program_map_section(
        program_number = avalpa1_service_id,
        PCR_PID = 2064,
        program_info_descriptor_loop = [],
        stream_loop = [
                stream_loop_item(
                        stream_type = 2, # mpeg2 video stream type
                        elementary_PID = 2064,
                        element_info_descriptor_loop = [
                                ca_descriptor (
                                        CA_system_ID = 0x2600,
                                        CA_PID = 0x1FFF,
                                ),
                        ]
                ),
                stream_loop_item(
                        stream_type = 4, # mpeg2 audio stream type
                        elementary_PID = 2068,
                        element_info_descriptor_loop = [
                                ca_descriptor (
                                        CA_system_ID = 0x2600,
                                        CA_PID = 0x1FFF,
                                ),
                        ]
                ),
        ],
```

let's try some output:

```
# run once only
./bissconfig.py
tscrypt keys.cw firstvideo.ts > firstvideocrypted.ts
tscrypt keys.cw firstaudio.ts > firstaudiocrypted.ts
mkfifo myfirstfifo.ts
mkfifo mysecondfifo.ts

# always running
tscbrmuxer b:2300000 firstvideocrypted.ts b:188000 firstaudiocrypted.ts b:3008 firstpat.ts b:3008
firstpmt.ts b:1500 firstsdt.ts b:1400 firstnit.ts b:10774084 null.ts> myfirstfifo.ts &
tsstamp myfirstfifo.ts 13271000 > mysecondfifo.ts &
tsrfsend mysecondfifo.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

The control word used is 0x00 0x00 0x00 0x00 0x01 0x02 0x03 0x6

This won't be enough to support any available CAS system but OpenCaster is also able to signal ECM, EMM and CAT in the same way as described for the others PSI/SI shown in the previous examples so the support is quite complete.

The scrambling processing is quite computational expensive so pay attention to your cpu usage.

Avalpa Broadcast Server:Avalpa Web EPG GUI

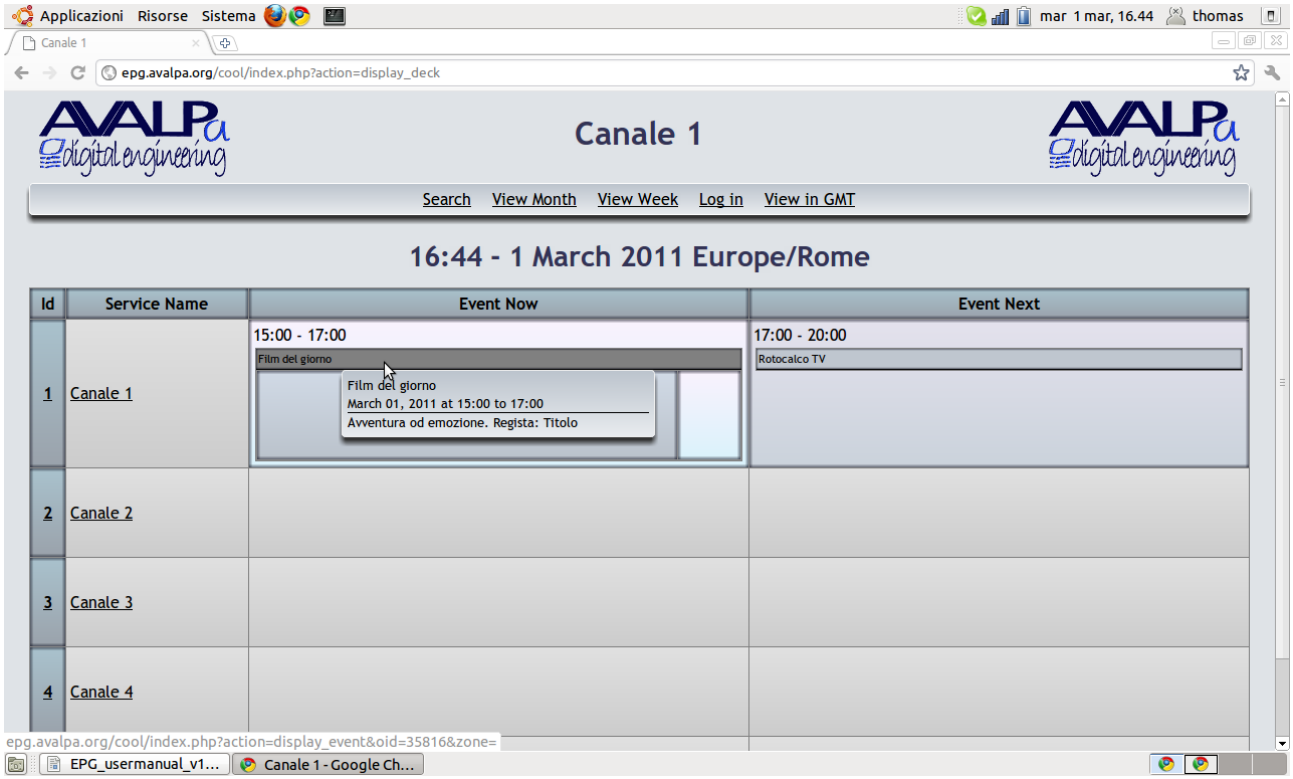
Reference

Already installed on the Avalpa Broadcast Server there is a Eletronic Program GUI generator than can configured by web. A complete manual is available in Italian and English translation is on the way, in the meanwhile a few topics are already covered.

Check out the guide at:

http://www.avalpa.com/assets/freesoft/opencaster/AvalpaBroadcastWebEPGManual-v1.0.pdf

Here is an actual screenshot:



Avalpa Broadcast Server for audio/video playout

Avalpa Broadcast Server enables you to set up a play out system of a sequence of pre-encoded files, the following chapters describe audio/video playout system, you can skip this section if you are interested in such usage

This shot shows the play out system in action, straight from a “budget” tv set, hence the interlacing, moire' and horizontal scanline artifacts!!

Many tv services are just the broadcasting of recorded events, video and films (think business television, hospitality and some satellite channels) . This context is a perfect match for Avalpa Broadcast Server. Using the following tools the broadcaster can avoid expensive real-time systems greatly reducing costs, parts, complexity and the storage needs for the running play out system.

To understand the play out system is necessary to understand the tools presented in this section, at the end of the chapter the simplest possible play out is realized with just a few command lines.

Introduction to ingestion

The most important issue about the ingestion of content is to validate it before usage so in the next chapters some analysis' tools are presented to make sure Avalpa Broadcast Server is filled with proper input from the encoders.

Current supported input formats for playout usage are:

- **MPEG2 video**
- **MPEG2 audio**
- **Dolby Digital aka ac3**
- **Dolby Digital Plus**
- **DTS (on test)**
- **h264 (on test)**

Most of **audio encoders** usually won't present a difficult challenge because for broadcasting usage their **bit rate is constant** and frame size is constant too so you will have only to get some parameters from the encoders' configuration and match them with Avalpa Broadcast Server's parameters.

Video Elementary Stream input, however, is a far more complex subject. For mpeg2 video the server makes available an analyser tool called mpeg2vbv (Video Buffer Verifier) that will report problem with the video input doing a decoder buffer simulation. H264 analyzer will be available soon.

Video bitrate can be both Variabile Bit Rate (VBR) or Constant Bit Rate (CBR) however in a digital video broadcasting architecture VBR makes sense only if multiplexers and encoders talk to each other and realize a statistical multiplexing otherwise using a VBR encoding without a statistical multiplexing would be just a waste of bandwidth. As far as we know there is not a standard about statistical multiplexing and every vendor has its own lock-in proprietary protocol between encoders, also **statistical multiplexing is valuable only on some context and requirement**. Working with pre-encoded files the choice of **CBR is a must especially for VOD services but also for scheduled services with small number of live programes**.

To get your video encoded there are many possibilities, we describe some of the most common:

An open source encoder library is available inside **libavcodec from ffmpeg**, you will be able to encode CBR video elementary stream as described in the chapters below without problem check later chapters for details. The **reference encoders** are also available from mpeg web site but it will probably be more cpu hungry and less effective, other software decoders are available both binary, open source, licensed and/or free.

MainConcept licenses software encoders with a CBR TS VBV compliant Video ES for a good price as standalone solution also on Linux workstations or it is available as a plug-in for Adobe Premiere and many other Non Linear Editing (NLE) softwares, even Adobe Premier Elements has a high quality MPEG2 video encoder option. A free trial download from their web site is also available with the only limitation of a watermark. Pay attention however GUIs are not always clear on how to generate a CBR video.

It is worthless to say that any professional broadcasting industrial encoder hardware will be also suitable, then some inexpensive pc hardware has been also reported to work good enough too if configured properly with up-to-date firmware.

Anyway in the following pages we will present examples with ffmpeg because it is installed and it has also a quite wide range of supported input formats making the tutorials easier.

Please note the **current ffmpeg version (0.6.1) has also a basic transport stream output but it is not suitable for CBR broadcast** so you cannot use with other OpenCaster tools, that is why we will always show example starting from pes or es input files.

Extracting Program and Elementary Streams: ts2pes

[first-file-mux]

Avalpa Broadcast Server comes with some tools to analyse Program and Elementary Stream, these are the basic operations needed to validate your input files before using them in the play out, this section will introduce you also the basic concepts you have to master if you want a smooth play out working without problems on any decoder.

ts2pes will extract from a transport stream the program elementary stream described with a PID, for example:

```
ts2pes firstvideo.ts 2064 > videooutput.pes
```

A program elementary stream files will be generated, usually audio or video.

pesinfo will report informations about the program elementary streams, usage is simple:

```
pesinfo videoutput.pes
```

Ouput is something like:

```
[...]
pes header: 00 00 01 e0, video stream number 0, header length: 5, Presentation Time Stamp is:
989100, 10.9900 sec.
pes header: 00 00 01 e0, video stream number 0, header length: 5, Presentation Time Stamp is:
992700, 11.0300 sec.
pes header: 00 00 01 e0, video stream number 0, header length: 10, Presentation Time Stamp is:
1007100, 11.1900 sec., Decode Time Stamp is: 996300, 11.0700 sec.
pes header: 00 00 01 e0, video stream number 0, header length: 5, Presentation Time Stamp is:
999900, 11.1100 sec.
[...]
```

so you can learn about presentation (PTS) and decoding time stamps (DTS) of the elementary stream of the program stream, for example you can see in the above print that decode time stamp is correctly a bit earlier than presentation time stamp for the same frame.

The time is printed as seconds and as 90Hz clock, we will discuss again PTS and DTS later on while discussing synchronization, remember that pesinfo will give you the necessary information.

pes2es tool will allow you to extract from a program elementary stream its elementary stream.

The tool takes as input a program elementary stream file and a stream header id, you can

read the header id using pesinfo as just shown in the example.

```
pes2es videooutput.pes 224 > video.es
```

will extract the stream 0xe0 (224), that's a typical video id while typical audio id is 0xc0 (192)

```
ts2pes firstaudio.ts 2068 > audiooutput.pes
pes2es audiooutput.pes 192 > audio.es
```

Analysing mpeg2 video files: esvideoinfo

[first-file-mux]

The command:

```
esvideompeg2info video.es
```

will print out information from the video frames header, here is a typical print of a Group Of Picture (GOP):

```
[...]
Sequence header: format: 720x576, 4:3, 25fps, bitrate: 6.00Mbs, vbv buffer size: 112, constrained:
no
Sequence header extension: profile is Main, level is Main
GOP header: measured size: 344267 bytes, bitrate from measured size (formula has rounds):
5737783.461583bps, drop: no, time code: 00:00:04 pictures:18, closed: no, broken: no
frame size: 20490
Postion 3525284, picture 119 start header: temporal reference: 2, picture coding type: I-Frame, vbv
delay: 22414
frame size: 78958
Postion 3604242, picture 120 start header: temporal reference: 0, picture coding type: B-Frame, vbv
delay: 16539
frame size: 17697
Postion 3621939, picture 121 start header: temporal reference: 1, picture coding type: B-Frame, vbv
delay: 18015
frame size: 18223
Postion 3640162, picture 122 start header: temporal reference: 5, picture coding type: P-Frame, vbv
delay: 19428
frame size: 30385
Postion 3670547, picture 123 start header: temporal reference: 3, picture coding type: B-Frame, vbv
delay: 19382
frame size: 18138
Postion 3688685, picture 124 start header: temporal reference: 4, picture coding type: B-Frame, vbv
delay: 20806
frame size: 23426
Postion 3712111, picture 125 start header: temporal reference: 8, picture coding type: P-Frame, vbv
delay: 21595
frame size: 29984
Postion 3742095, picture 126 start header: temporal reference: 6, picture coding type: B-Frame, vbv
delay: 21597
frame size: 17189
Postion 3759284, picture 127 start header: temporal reference: 7, picture coding type: B-Frame, vbv
delay: 23134
frame size: 20719
Postion 3780003, picture 128 start header: temporal reference: 11, picture coding type: P-Frame, vbv
delay: 24248
frame size: 30980
Postion 3810983, picture 129 start header: temporal reference: 9, picture coding type: B-Frame, vbv
delay: 24130
frame size: 22844
[...]
```

Some of these parameters are relevant because they will be useful for multiplexing, so

let's have a short description of relevant parameters in the next chapter.

In MPEG-2, three 'picture types' are defined. The picture type defines which prediction modes may be used to code each block.

'Intra' pictures (I-pictures) are coded without reference to other pictures. Moderate compression is achieved by reducing spatial redundancy, but not temporal redundancy. They can be used periodically to provide access points in the bitstream where decoding can begin.

'Predictive' pictures (P-pictures) can use the previous I- or P-picture for motion compensation and may be used as a reference for further prediction. Each block in a P-picture can either be predicted or intra-coded. By reducing spatial and temporal redundancy, P-pictures offer increased compression compared to I-pictures.

'Bidirectionally-predictive' pictures (B-pictures) can use the previous and next I- or P-pictures for motion-compensation, and offer the highest degree of compression. Each block in a B-picture can be forward, backward or bidirectionally predicted or intra-coded. To enable backward prediction from a future frame, the **encoder reorders the pictures** from natural 'display' order to 'bitstream' order so that the B-picture is transmitted after the previous and next pictures it references. This introduces a reordering delay dependent on the number of consecutive B-pictures.

The different picture types typically occur in a repeating sequence, termed a 'Group of Pictures' or GOP. A typical GOP in display order is:

```
B1 B2 I3 B4 B5 P6 B7 B8 P9 B10 B11 P12
```

The corresponding bitstream order is:

```
I3 B1 B2 P6 B4 B5 P9 B7 B8 P12 B10 B11
```

A regular GOP structure can be described with two parameters: N, which is the number of pictures in the GOP, and M, which is the spacing of P-pictures. The GOP given here is described as N=12 and M=2. MPEG-2 does not insist on a regular GOP structure. For example, a P-picture following a shot-change may be badly predicted since the reference picture for prediction is completely different from the picture being predicted. Thus, it may be beneficial to code it as an I-picture instead.

For a given decoded picture quality, coding using each picture type produces a different number of bits. In a typical example sequence, a coded I-picture was three times larger than a coded P-picture, which was itself 50% larger than a coded B-picture.

The number after the picture type is the temporal reference inside the single GOP.

On broadcasting environment an external additional constraint comes from the user acceptable “zapping” time because a decoder needs to wait an I picture to start displaying the video so the usual number of GOP size for broadcasting is 12, less than half a second at 25 frames per second: (12 * 1/25 = 0.48 sec.)

Closed gop have a different sequence, they start with an I picture with time reference 1 and allow them to be the start of an encoded video. This peculiar GOP feature allows also

GOP to be chained without re-encoding the video, we will get on this topic later on.

Tool vbv

Another interesting tool is "**mpeg2videovbv**", that's Video Buffer Verifier, the tool tries to simulate a mpeg2 video decoder buffer.

```
mpeg2videovbv video.es
```

will print underrun and overrun of the buffer if they happen or stay quiet if the video stream respects the limits reporting “found 0 errors” at the end.

If vbv reports error it is a hint that the video was encoded with a bit rate that's not enough constant.

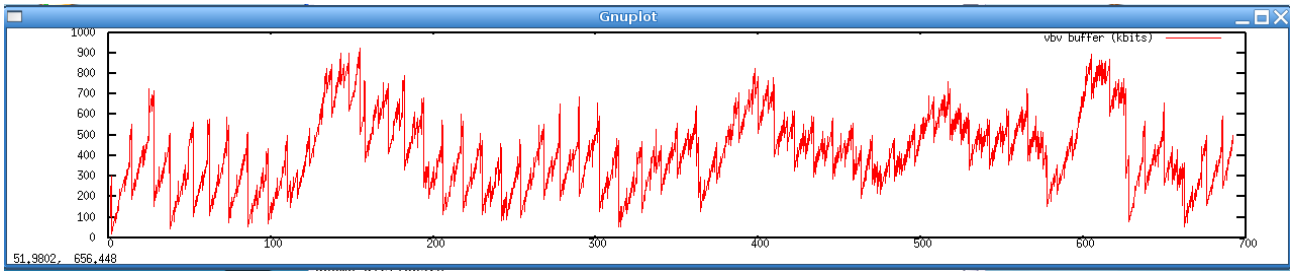
N.b. this tool is a simulator that tries to be strict, sometime a video stream happens to work properly on real decoders (as they should be far more tolerant) even if vbv reports errors.

It can also happen a video won't work properly even if vbv doesn't complain because buffer fullness is not the only important issue for a proper playback, the vbv tool fills the buffer at a constant bit rate (CBR) but on real transmission the model may look more like many small burst getting an average “cbr-ness”, if the burstiness is not properly managed a vbv complaint stream will fail to be decoded correctly.

Vbv information is also collected in a log file vbvData.dat that can be plotted with the command:

```
gnuplot plot.p
```

the plot.p file is available in the directory Opencaster.version/tools/vbv, the result looks like:



The Y is reporting the vbv buffer fullness while X is reporting the time in frame number. gnuplout let you zoom in an area using right click so you can analyse peaks and everything else.

Analysing audio elementary streams: esaudioinfo

[first-file-mux]

Avalpa Broadcast Server comes also with "**esaudioinfo**" tool to analyse mpeg audio elementary stream, usage is simple:

```
esaudioinfo audio.es
```

A typical output for every audio frame is already quite verbose:

```
[...]
audio header packet 1475, position:983812
audio version: MPEG Version 1 (ISO/IEC 11172-3)
audio layer: Layer II
protection bit: Not protected
bit rate index: 224kbps
sampling rate: 48000Hz
padding: Frame is not padded
audio frame from headers (formula can have rounds): 672 bytes, 5376 bits
channel mode: Stereo
copyrights: Audio is not copyrighted
original: Original media
emphasis: None
audio frame size from stream measured: 672 bytes, 5376 bits
[...]
```

MPEG-1 Audio Layer II is an audio codec defined by ISO/IEC 11172-3. While MP3 is much more popular for PC and internet applications, MP2 remains a dominant standard for audio broadcasting. Understanding mpeg audio compression features is beyond our interest but to multiplex audio you will need to know some of the information printed here: **bit rate, sampling rate and frame size**.

Sampling rate is the number of samples present in 1 second, that's why is expressed in Hz, while frame size is the dimension in bytes of all the audio frame. 48000 Hz is quite a common sample rate but also others are supported.

Knowing frame size and bit rate you can figure out how long the frame lasts and how many samples it brings with the sampler rate information.

Summary: to analyse an audio stream from a input.ts you will need the following commands:

```
[...]
ts2pes input.ts audiopid > audiooutput.pes
pes2es audiooutput.pes stream_id_usually_192 > audiooutput.es
esaudioinfo audiooutput.es
[...]
```

How to encode mpeg2 digital video files with ffmpeg and mjtools

Here is an example of video encoding done with ffmpeg:

```
"ffmpeg -i input.ext -an -vcodec mpeg2video -f mpeg2video -b 5000k -maxrate 5000k -minrate 5000k -bf 2 -bufsize 1835008 video.mp2"
```

-i input file, that can be video file supported by ffmpeg decoding

-an no audio, if audio is present in the input it will be ignored

-vcodec video codec, mpeg2video is the video codec we are looking for mpeg2 video

-f output format, this mpeg2video is the output file format we are looking, elementary stream video

-b, -maxrate, -minrate bitrate in kbps, that's depend on you, pay attention some version of ffmpeg use kbps as input, others bps so you need to add zeros

-bf number of b-frame for gop

-bufsize vbv buffersize, the version of ffmpeg used while writing this document is expecting the buffer size in bit, so it's vbv_buffer_size * 1024 * 16: 1835008, however in many encoders vbv for mpeg 2 video codec is often expressed in 16Kbits unit.

Be sure to check the output with esvideoinfo and vbv as explained in "Analysing video files", when you are done with encoding you will need to do encapsulation, ffmpeg actually has an option to output elementary stream encapsulated into transport stream but as of this writing it's totally broken. You can encapsulate the video elementary stream into program stream with esvideo2pes like this:

```
esvideompeg2pes video.mp2 > video.pes
```

You can analyse the output with:

```
pesinfo video.pes
```

You can encapsulate the video into a ts like in this example:

```
pesvideo2ts 2064 25 112 5270000 0 video.pes > video.ts
```

2064 is the video pid. 25 is the frame per second, 112 half vbv (on many decoders 224 is fine, others have half of it because DVD uses half and sometimes it happens also DVB decoder get the same limitation), 5270000 is the ts bit rate, it has to be bigger than the video bit rate, automatic minimum guess is still not available on the software, 15% more should be fine; the tool will adjust the pts and the dts to take into account first frame transmission delay.

To improve the quality and the compression of the video is suggested to use yuvdenoise tool, as shown below:

```
ffmpeg -i input.ext -an -s 720x576 -deinterlace -r 25 -aspect 4:3 -f yuv4mpegpipe - | yuvdenoise | ffmpeg -i - -an -vcodec mpeg2video -f mpeg2video -b 2000k -maxrate 2000k -minrate 2000k -bf 2 -bufsize 1343488 video.mp2
```

How to encode digital audio files with ffmpeg

Another example with ffmpeg:

"ffmpeg -i input.mpg -ac 2 -vn -acodec mp2 -f mp2 -ab 128000 -ar 48000 audio.mp2"

-i input file

-ac 2, stereo

-vn no video

-acodec mpeg2 audio layer 2

-f output format mpeg2 audio

-ab audio bit rate in bps

-ar is the audio sample rate

You can analyze the output with:

```
esaudioinfo audio.mp2
```

You can encapsulate the audio into ps with esaudio2pes like this:

```
esaudio2pes audio.mp2 1152 48000 768 -1 3600 > audio.pes
```

48000 sample rate, should be known or can be learn with esaudioinfo, a pts_step is 1152 / sample_rate * 90000, 1152 is fixed for mpeg2 layer 2 so for 48khz comes 2160

1152 audio frame size, you can read it with esaudioinfo, it's in bytes (NB this 1152 is by chance 1152 as the number of samples)

3600 first pts, this an important value for audio/video synchronization, read more below.

You can analyse the output with:

```
pesinfo audio.pes
```

You can encapsulate the audio into a ts with:

```
pesaudio2ts 2068 1152 48000 768 -1 0 audio.pes > audio.ts
```

2068 is the pid number

1152 is the number of sampler per frame

48000 is the sample rate

768 is the es frame size

-1 disable audio description header

0 the audio won't be on loop, again this is for future usage

constant bit rate.

To calculate the output bit rate steps are:

1 second is 90000 pts ticks, sample rate is 48000 hz, 1 frame is 1152 samples. so:

(90000 * 1152) / 48000 = 2160 is how many ticks is a frame

an audio pes frame is made for example of 384 byte so this means

384/184 = 2.08 TS packets, with padding on third packet so 3 packets * 188 bytes = 564 bytes = 4512 bit per pes audio frame.

4512 bit for 2160 ticks you convert with bps it comes:

(4512 * 90000) / 2160 = 188000 bps

When bit rate is not round, you should consider using the floor integer.

Beware of padding: if you check the math for audio bit rate at 160kbps you will find out that the TS bit rate is the same! Check OpenCaster README for smarter mpeg2 audio bit rates.

How to capture DV input

This chapter assumes you connected a digital video camera or a digital video player to the firewire of the server.

To record the DV you have to use the command:

```
dvgrab recorded.dv
```

But this way it will never stop recording, so to stop it you can use ctrl+c or use a duration time:

```
dvgrab -d 3:30:10 recorded.dv
```

will record for 3 hours, 30 minutes and 10 seconds, here is other time specification supported:

```
SMIL time value: XXX[.Y]h, XXX[.Y]min, XXX[.Y][s], XXXms, [[HH:]MM:]SS[.ms]

smpte=[[[HH:]MM:]SS:]FF
```

The resulting recordDV.avi file can be converted as explain in previous chapter into audio and video streams. The next chapter will show a complete ingestion from DVD data available on the net, you can replace the vob file with your DV file and execute the same steps

You can also connect DVGrab directly into ffmpeg and then directly to a playout:

```
dvgrab -format dv1 - | /home/avalpa/ffmpeg/ffmpeg -f dv -i - -acodec mp2 -ac 2 -ab 128000 -ar 48000
-f mp2 -y live.mp2 -s 720x576 -deinterlace -r 25 -aspect 4:3 -f yuv4mpegpipe -y - |
/home/avalpa/mjpegtools-1.9.0/yuvdenoise/yuvdenoise | /home/avalpa/ffmpeg/ffmpeg -i - -an -vcodec
mpeg2video -f mpeg2video -b 2000k -maxrate 2000k -minrate 2000k -bf 2 -bufsize 1343488 -y live.mpv
```

How to generate silence

Sometimes it is useful to generate silent audio tracks, for this purpose a simple example using sox and ffmpeg is given:

```
dd if=/dev/zero of=silence.raw bs=1152 count=4
sox -t raw -r 48000 -s -w -c 2 silence.raw silence.wav
ffmpeg -vn -f s16le -ab 128k -ar 48000 -ac 2 -i silence.wav -acodec mp2 silence.mp2
```

In this case 1152 is the number of samples of a single mpeg2 audio frame so in this way we can control how many frames are generated using count.

Number of samples, number of channels and sample rate will tell you also how long it lasts.

Audio and video real case ingestion study

[encoding/logo_tv.png]

Let's begin with a sequence of commands that will eventually give you some encoded files that can be used with Avalpa Broadcast Server. Of course this is just an example, it's better to stick with it for the beginners, otherwise you are on your own:

```
wget http://video.blendertestbuilds.de/topdir/ED/movie_only_pal.iso
```

wget will download the .iso of Elephant Dreams short and the following command will encode audio and video adding an external logo and enabling the initial synchronization.

```
sudo mount -o loop movie_only_pal.iso /mnt/
ffmpeg -vn -ab 128k -ar 480000 -i /mnt/VIDEO_TS/VTS_01_1.VOB -acodec mp2 -ac 2 ed.mp2
ffmpeg -i /mnt/VIDEO_TS/VTS_01_1.VOB  -an -vf 'movie=OpenCaster/tutorials/encoding/logo_tv.png [wm];
[in][wm] overlay=0:0:0:0 [out]' -f mpeg2video -vcodec mpeg2video -b 2600k -maxrate 2600k -minrate
2600k -bf 2 -bufsize 1835008 -aspect 4:3 ed.m2v
sudo umount /mnt
esvideo2pes ed.m2v 1> ed.video.pes 2> ed.pes.length
esaudio2pes ed.mp2 1152 48000 384 -1 3600 > ed.audio.pes
```

The file ed.pes.length is generated by esvideo2pes and it's the length of the video in PTS ticks, in this case we will get that ed.pes.length is 57286800, ticks of 90Khz so 1 frame is 90000/25 frame per second = 3600 pts ticks and 57286800/3600 = 15913 frames / 25 fps = 636.52 seconds of video

We need to compare this with the audio length to ensure the audio is a little shorter than the video, this is the only requirement to chain two videos encoded as just described!

ed.audio.pes size is 10530284, 1 frame is 1152 sample and sample rate is 48000 so 10530284 / (384+14) = 26458 frames * 1152 / 48000 = 634.992 seconds of audio

In this case audio is shorter so there is no problem, otherwise:

```
esaudio2pes ed.mp2 1152 48000 384 -1 3600 57286800 > ed.audio.pes
```

would have cut the audio before the end of the video. To have some video clips for the playout you should repeat the same commands for another short Big Buck Bunny (the second release still courtesy from the Blender Foundation), available from: http://www.archive.org/details/BigBuckBunny

To match our set up in the next chapters you should encode `VTS_05_1.VOB` and `VTS_02_1.VOB`

Audio and video initial synchronization

The first PTS value is important for synchronization between audio and video, that's because it is necessary to synch the presentation of the first video frame and the first audio frame.

First of all let's make sure with pesinfo that PTS of the first video frame matches the PTS of the first audio frame, better if they both are 3600 ticks of the 90 KHZ clock that's 0.04 ms that's 1 video frame at 25 frame per second, if you properly run esaudio2pes and esvideo2pes from the previous instructions you should have a video.pes starting like this:

```
pes header: 00 00 01 e0, video stream number 0, header length: 10, Presentation Time Stamp is: 3600,
0.0400 sec., Decode Time Stamp is: 0, 0.0000 sec.
pes header: 00 00 01 e0, video stream number 0, header length: 10, Presentation Time Stamp is:
14400, 0.1600 sec., Decode Time Stamp is: 3600, 0.0400 sec.
```

And an audio.pes starting with:

```
pes header: 00 00 01 c0, audio stream number 0, pes size 392, header length: 5, Presentation Time
Stamp is: 3600, 0.0400 sec.
pes header: 00 00 01 c0, audio stream number 0, pes size 392, header length: 5, Presentation Time
Stamp is: 5760, 0.0640 sec.
pes header: 00 00 01 c0, audio stream number 0, pes size 392, header length: 5, Presentation Time
Stamp is: 7920, 0.0880 sec.
```

Encoding with x264

This section will show how to use x264 to encode h264 video instead of mpeg2 video. The current example will show how to broadcast a 1080p24 video.

The video format is actually a non common choice for RF broadcasting and not standardized for DVB but it suites local networks use cases, VOD, and it is the simpler example on how to start as it doesn't require telecine techniques:

```
ffmpeg -i big_buck_bunny_1080p_h264.mov 'movie=OpenCaster/tutorials/encodingHD/logo_tv.png [wm];[in]
[wm] overlay=0:0:0:0 [out]' -f yuv4mpegpipe - | ./x264 --level 4.0 --nal-hrd cbr --vbv-bufsize 2000
--bitrate 7200 --keyint 12 --vbv-init 0 -o bb.x264.hd.video.pes

pesvideo2ts 2066 24:38 b7200000 7600000 0 bb.x264.hd.video.pes > video.ts

ffmpeg -i big_buck_bunny_1080p_h264.mov -vn -acodec ac3 -ar 48000 -ab 448k audio24p.ac3
```

since we are dealing with HD, let's also put 5.1 audio!

```
esaudio2pes audio24p.ac3 1536 48000 1792 -1 3750 > audio.pes

pesaudio2ts 2069 1536 48000 1792 0 audio.pes > audio.ts
```

it will also require to change psi to signal correctly the service in a few places:

for nit/sdt:

```
      service_type = 0x19, # avc hd digital tv service type
```

for pmt:

```
        stream_loop = [
                stream_loop_item(
                        stream_type = 0x1B, # avc video stream type
                        elementary_PID = 2066,
                        element_info_descriptor_loop = []
                ),
                stream_loop_item(
                        stream_type = 6, # private
                        elementary_PID = 2069,
                        element_info_descriptor_loop = [
                                ac3_descriptor(
                                        component_type_flag = 0,
                                        bsid_flag = 0,
                                        mainid_flag = 0,
                                        asvc_flag = 0,
                                        additional_info = "",
                                ),
                        ],
                ),
        ],
```

finally tscbrmuxer rates are:

```
[..] b:7600000 video.ts b:470000 audio.ts [...]
```

Play out scheduling from command line

[first-file-mux]

Here is the first example of audio and video play out script, we assume you already encoded the file as explained in “Audio and video real case encoding with ffmpeg”, the file names will be ed.video.pes, ed.audio.pes, ed.pes.length and similar for the other DVD encodings: bb.video.pes bb.audio.pes bb.pes.length bb2.video.pes bb2.audio.pes and finally bb2.pes.length

```
mkfifo video.ts
mkfifo audio.ts
mkfifo muxed.ts
mkfifo stamped.ts
pesvideo2ts 2064 25 112 2900000 1 ed.video.pes bb.video.pes bb2.video.pes > video.ts &
pesaudio2ts 2068 1152 48000 384 1 ed.audio.pes bb.audio.pes bb2.audio.pes > audio.ts &
tscbrmuxer b:2800000 video.ts b:188000 audio.ts b:3008 firstpat.ts b:3008 firstpmt.ts b:1500
firstsdt.ts b:1400 firstnit.ts b:10174084 null.ts > muxed.ts &
tspcrstamp muxed.ts 13271000 > stamped.ts &
tsrfsend stamped.ts -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf 578
```

This will output three files with audio and video connected seamlessly, please notice some points about why this works:

- ed, bb and bb2 have all the same bandwidth both audio and video and all the bandwidths are CBR
- the first GOP of every video is closed
- pesvideo2ts controls vbv fullness all the time and also on the switch between streams.

At every connection some interesting informations are printed:

```
pesaudio2ts: closing ed.audio.pes...  closed
pesaudio2ts warning: missed ed.audio.pes.length file
pesaudio2ts: opening bb.audio.pes...  open
pesaudio2ts sync: bb.audio.pes new presented audio frame will be at 57164400, 63
5.1600 sec., last presented audio frame was at 57161520, 635.1280 sec.
pesvideo2ts: closing ed.video.pes...  closed
pesvideo2ts: opening bb.video.pes...  open
pesvideo2ts sync: bb.video.pes new presented video frame is at: 57290400, 636.56
00 sec., decode time stamp is at: 57286800, 636.5200 sec.
```

Basically the prints report about the tools closing the previous files and opening the new files however something fishy is going on because we read that:

```
bb.audio.pes new presented audio frame will be at 57164400, 635.1600 sec.
bb.video.pes new presented video frame is at: 57290400, 636.5600 sec.
```

We are out of synchronization! The first value is the time in PTS ticks (90 KHZ) the second is the first time in seconds (57164400 / 90000).

This problem is also signalled by a warning message:

```
pesaudio2ts warning: missed ed.audio.pes.length file
```

we specified that the audio file must be generally shorter than the video file however for pesaudio2ts to know about how much short it is the audio compared to the video it needs additional information files: ed.audio.pes.length, bb.audio.pes.length and bb2.audio.pes.length that are the same files we generated while encapsulating the video files to pes in the previous chapter so just rename them as:

```
mv bb.pes.length bb.audio.pes.length
mv bb2.pes.length bb2.audio.pes.length
mv ed.pes.length ed.audio.pes.length
```

and restart the whole processes, after the first video you will correctly read that both pesaudio2ts and pesvideo2ts report the exact PTS that is going to be the first frame. of the incoming video:

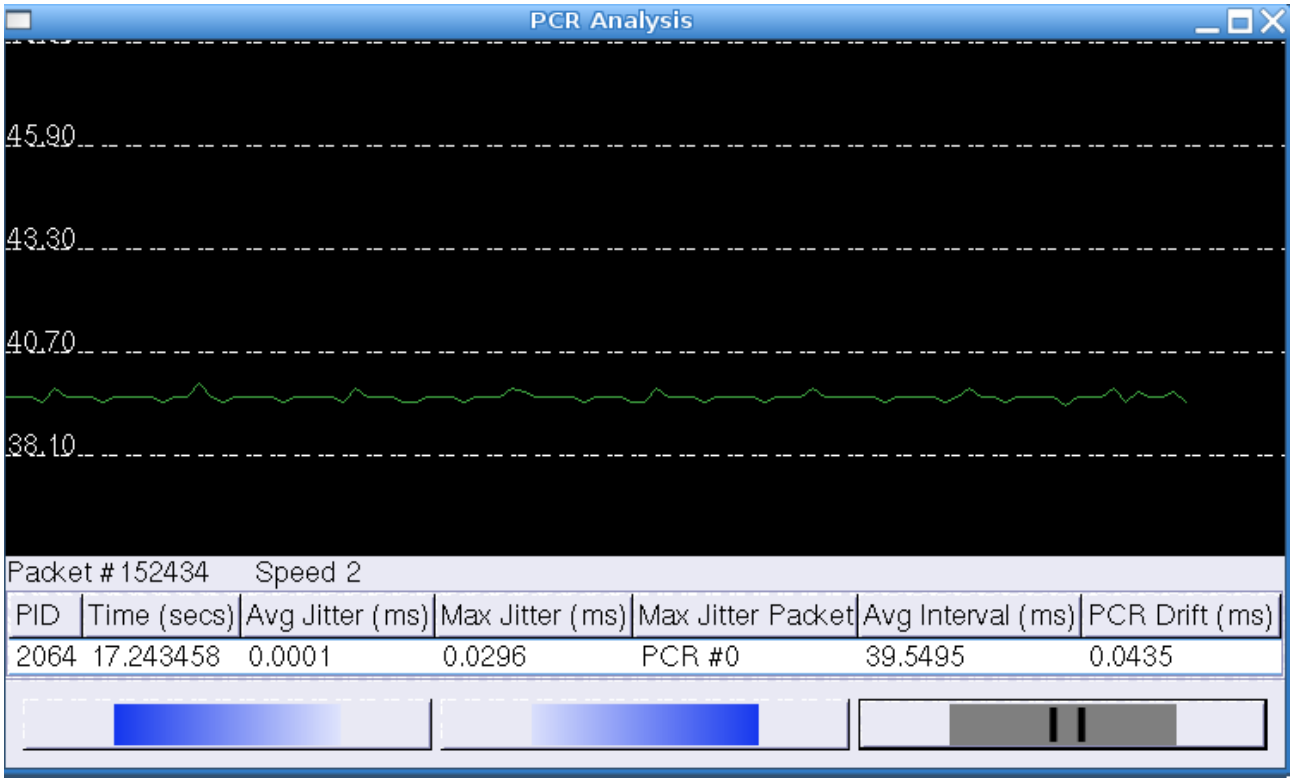
```
pesaudio2ts: closing ed.audio.pes...  closed
pesaudio2ts: opening bb.audio.pes...  open
pesaudio2ts sync: bb.audio.pes new presented audio frame will be at 57290400, 636.5600 sec., last
presented audio frame was at 57161520, 635.1280 sec.
pesvideo2ts: closing ed.video.pes...  closed
pesvideo2ts: opening bb.video.pes...  open
pesvideo2ts sync: bb.video.pes new presented video frame is at: 57290400, 636.5600 sec., decode time
stamp is at: 57286800, 636.5200 sec.
```

As you can read now:

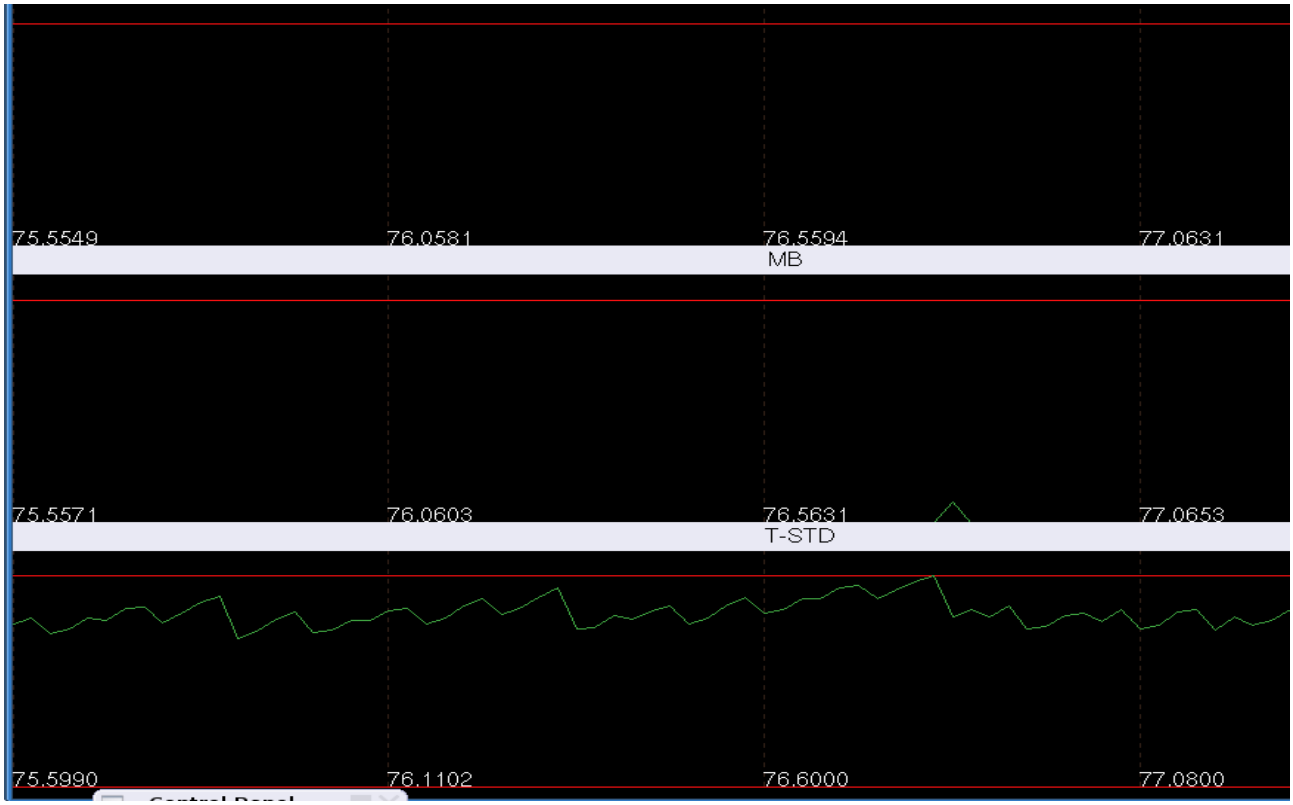
```
bb.audio.pes new presented audio frame will be at 57290400, 636.5600 sec.
bb.video.pes new presented video frame is at: 57290400, 636.5600 sec.
```

To recover from an out of synchronization after it is signalled you will need to do some mathematics and add at the current.audio.pes.length value the missing time so that while passing to the new stream from the current the synch is gained.

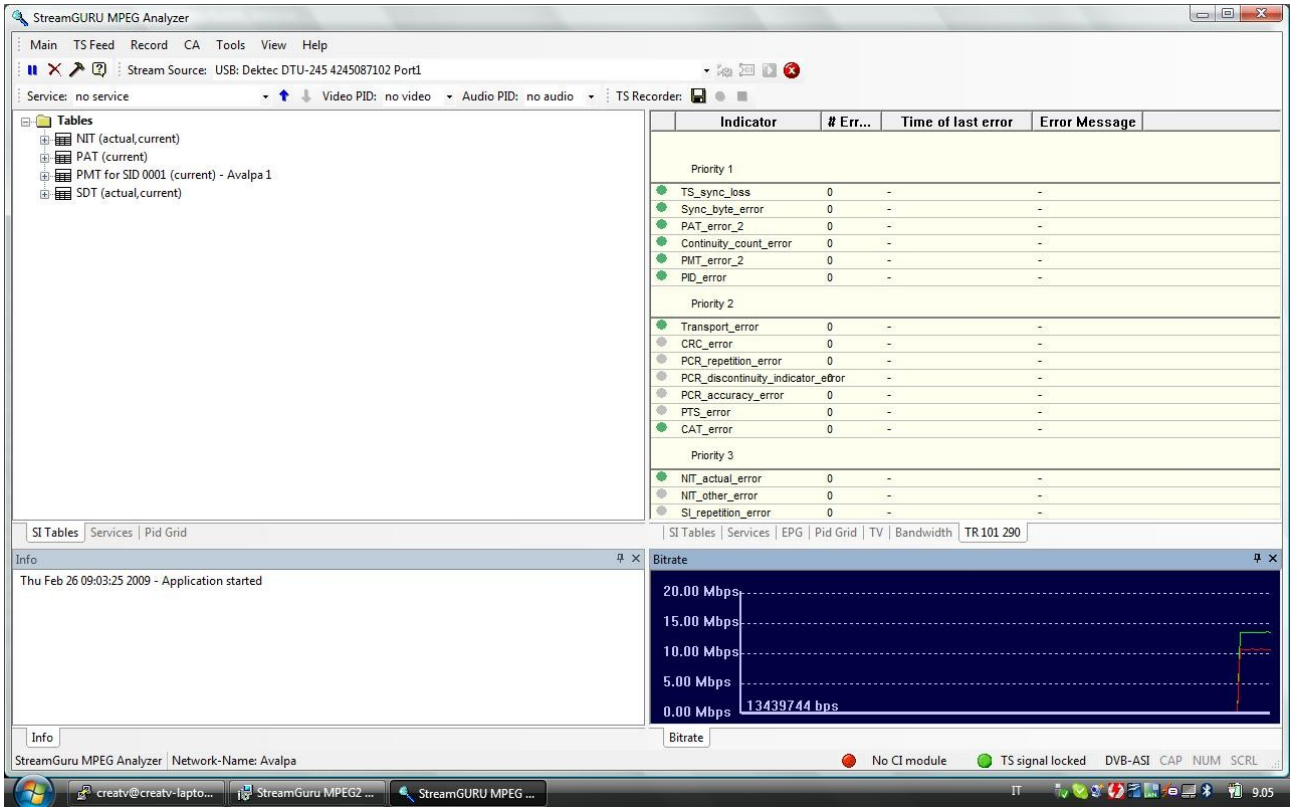
In the next pages a gallery of some screen shots from different analysers is shown:



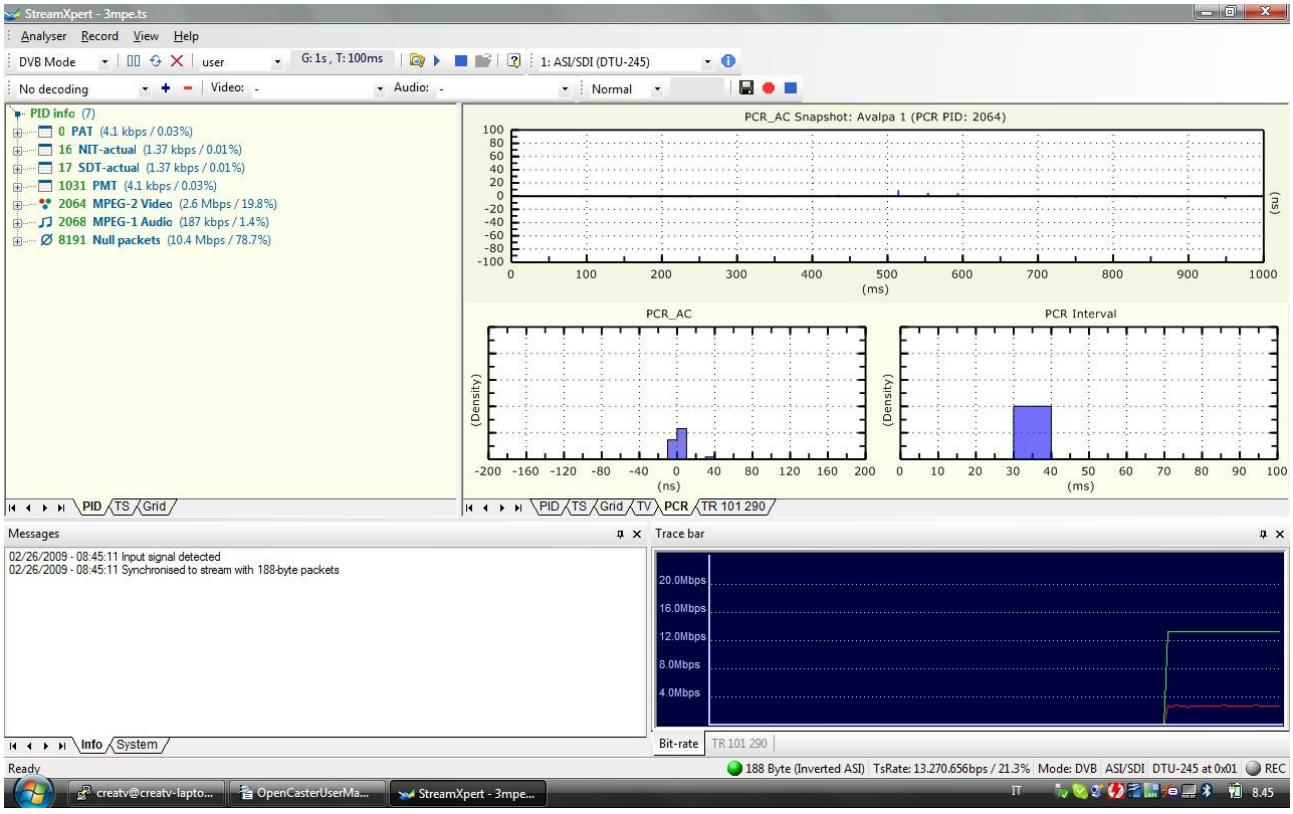
VK Tafe PCR Analysis



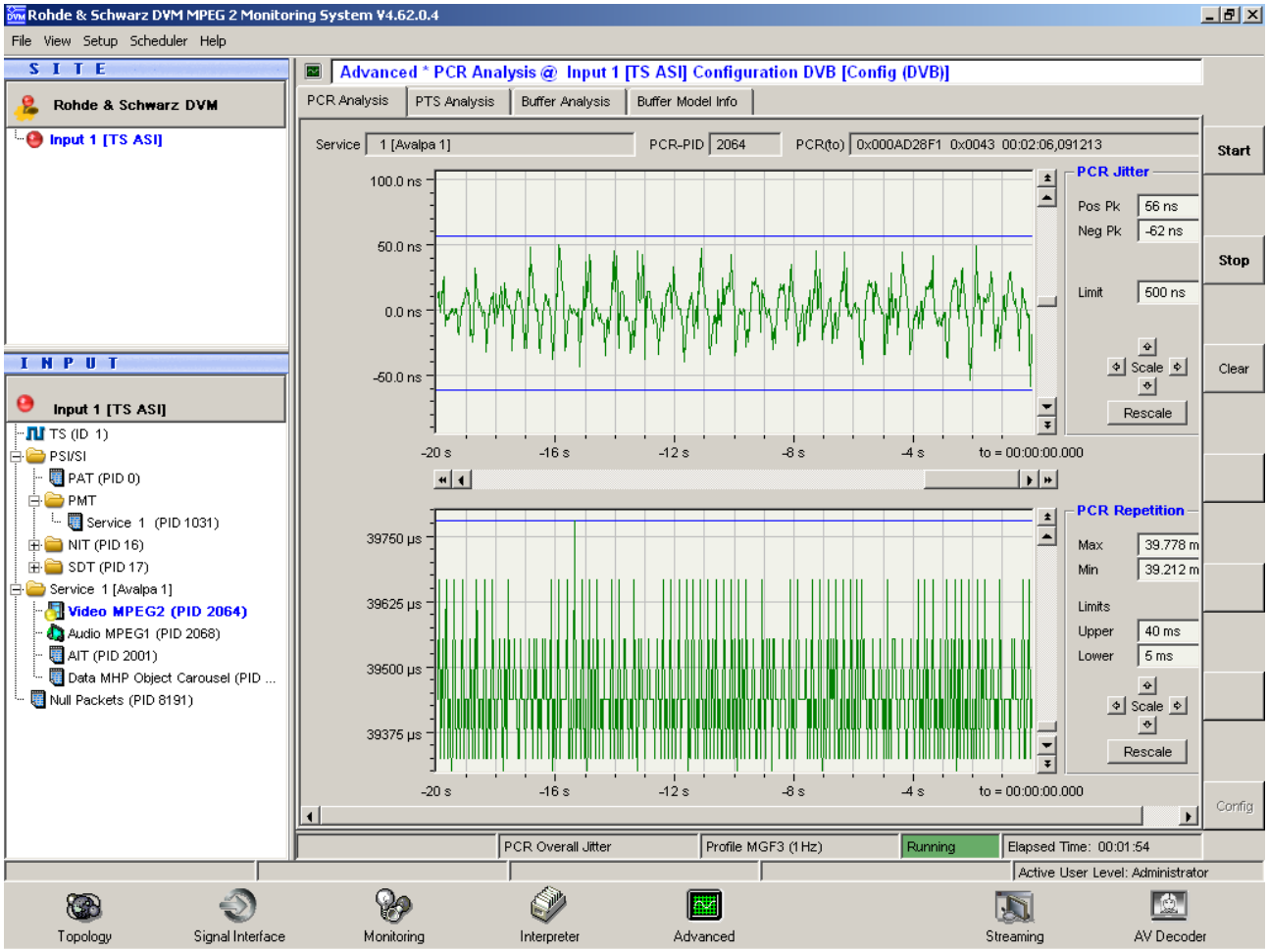
VK Tafe T-STD Analysis



StreamGuru Tr101 290



StreamXpert PCR analysis



DVM100 Rohde & Schwarz PCR analysis

Now another step towards a more flexible play out system: what if we want to change our schedule after it started? We can do that easily with some **symbolic linking**, suppose that instead to use file names we use symbolic links:

```
ln -s ed.audio.pes audio1.pes
ln -s ed.video.pes video1.pes
ln -s ed.audio.pes audio2.pes
ln -s ed.video.pes video2.pes
ln -s ed.audio.pes audio3.pes
ln -s ed.video.pes video3.pes
pesvideo2ts 2064 25 112 2900000 1 video1.pes video2.pes video3.pes > video.ts &
pesaudio2ts 2068 1152 48000 384 1 audio1.pes audio2.pes audio3.pes > audio.ts &
```

To change a scheduled file we just need to change the symbolic link before it starts to be play:

```
rm audio2.pes
rm video2.pes
ln -s bb.audio.pes audio2.pes
ln -s bb.video.pes video2.pes
```

Re-multiplexing input transport streams

Re-multiplexing is a very common scenario in many DVB networks and can be achived for ethernet input/output also without Avalpa Broadcast Server dedicated DVB hardware and software if OpenCaster is correctly installed, configured and managed.

Check the chapter “Ip network tools” before continue for a brief explanation of the design choices.

Tool tsorts

[mpts]

Tsorts tool will listen to input network and grab a packet from it if it's ready or replace it with a null packet if there was a transmission problem and the packet couldn't be delivered, here is an example setup:

```
tsudpreceive 224.0.1.2 1234 > input.ts &
torts input.ts null.ts > tsored.ts &
tspcrrestamp tsored.ts 24130000 > stamped.ts &
tsudpsend stamped.ts 224.0.1.2.3 1234 24130000
```

Tsorts will also add null packets if the incoming stream is not loosing packets but its bit rate is lower than the specified.

Tspcrrestamp will take care to restamp the pcr so the jitter of the null pid insertion is reduced.

To achive multiplexing from more input it is necessary to increase tsudpreceive processes, let's check an example where two incoming streams are muxed and then sent to another multicast group in output to be received by an ip DVB-T modulator:

```
tsudpreceive 224.0.1.1 1234 > input1.ts &
tsudpreceive 224.0.1.2 1234 > input2.ts &
torts input1.ts null.ts > tsored1.ts &
torts input2.ts null.ts > tsored2.ts &
tscbrmuxer b:2000000 tsored1.ts b:3000000 tsored2.ts o:13271000 null.ts > muxed.ts &
tspcrrestamp muxed.ts 13271000 > stamped.ts &
tsudpsend stamped.ts 224.0.1.2.3 1234 13271000
```

In this example we assume the two streams are 2mbps for 224.0.1.1 1234 and 3mbps for 224.0.1.2 1234, tscbrmuxer will increase the bit rate to 1327100 matching a modulation scheme.

If the modulator is able to add null pid is a betterto reduce the bandwidth:

```
tsudpreceive 224.0.1.1 1234 > input1.ts &
tsudpreceive 224.0.1.2 1234 > input2.ts &
torts input1.ts null.ts > tsored1.ts &
torts input2.ts null.ts > tsored2.ts &
tscbrmuxer b:2000000 tsored1.ts b:3000000 tsored2.ts > muxed.ts &
tspcrrestamp muxed.ts 5000000 > stamped.ts &
tsudpsend stamped.ts 224.0.1.2.3 1234 5000000
```

The obvious problem in such approach is that the if the incoming transport stream have both the same PID like the PID for the PAT the output stream will be inconsistent, so it is better to replace tables, for example:

```
tsudpreceive 224.0.1.1 1234 > input1.ts &
tsudpreceive 224.0.1.2 1234 > input2.ts &
torts input1.ts null.ts > tsored1.ts &
torts input2.ts null.ts > tsored2.ts &
tscbrmuxer b:2000000 tsored1.ts b:3000000 tsored2.ts > muxed.ts &
tsmodder muxed.ts +0 pat.ts > modded.ts &
tspcrrestamp modded.ts 5000000 > stamped.ts &
tsudpsend stamped.ts 224.0.1.2.3 1234 5000000
```

Real life usage require far more complex set-ups but are basically using the same tools as shown up to now, you will probably need to filter some packets with tsmask or change pid number with tsmodder and so on, it all depends from the incoming streams.

If you can configure incoming streams to be only content pids, like audio, video and teletext, the final setup won't differ from the multiplexing example of mpts tutorial where audio and video ts files are replaced by pipes.

Appendix A: Acronyms, glossary and references

The most frequent acronyms and something about it

AIT: Application Information Table (ETSI TS 102 812)

CBR: Constant Bit Rate

DVB: Digital Video Broadcasting (http://www.dvb.org/)

DVB-ASI: DVB-Asynchronous Serial Interface (EN 50083-9)

DSM-CC: Digital Storage Media Command and Control (ISO/IEC 13818-6, EN 301 192)

ETSI: European Telecommunications Standards Institute, the standardization body for many digital television standards (http://www.etsi.org)

GOP: Group Of Pictures, a sequence of frames on a video stream compressed together, usually 12 or 15

IP: Internet Protocol (rfc 791)

MHP: Multimedia Home Platform (ETSI TS 102 812)

MPE: Multi Protocol Encapsulation (ISO/IEC 13818-6, EN 301 192)

MHEG5:(ISO/IEC 13522-5)

MPEG: Motion Picture Expert Group (http://multimedia.telecomitalialab.com/)

MPTS: Multi Program Transport Stream, a MPEG2 TS carrying more then one service.

NIT: Network Information Tablet (ISO/IEC 13818-1)

NVOD: Near Video on Demand

PAT: Program Association Table (ISO/IEC 13818-1)

PID: Program IDentifier (ISO/IEC 13818-1)

PMT: Program Map Table (ISO/IEC 13818-1)

PTS: Presentation Time Stamp (ISO/IEC 13818-2)

PSI: Program Signalling Information (ISO/IEC 13818-1)

STB: Set Top Box, the decoder.

SPTS: Single Program Transport Stream (ISO/IEC 13818-1)

TS: Transport Stream (ISO/IEC 13818-1)

VBR: Variable Bit Rate

VOD: Video On Demand

PES: Program Elementary Stream (ISO/IEC 13818-1)

ES: Elementary Stream (ISO/IEC 13818-1)

Appendix B: DVB-T transmission parameters and net bitrates

Available bitrates (Mbit/s) for a DVB-T system in 8 MHz channels

(courtesy Wikipedia)

Modulation QPSK

Coding rate/ Guard interval 1/4 1/8 1/16 1/32

1/2 **4.976 5.529 5.855 6.032**

2/3 **6.635 7.373 7.806 8.043**

3/4 **7.465 8.294 8.782 9.048**

5/6 **8.294 9.216 9.758 10.053**

7/8 **8.709 9.676 10.246 10.556**

Modulation 16-QAM

Coding rate/ Guard interval 1/4 1/8 1/16 1/32

1/2 **9.953 11.059 11.709 12.064**

2/3 **13.271 14.745 15.612 16.086**

3/4 **14.929 16.588 17.564 18.096**

5/6 **16.588 18.431 19.516 20.107**

7/8 **17.418 19.353 20.491 21.112**

Modulation 64-QAM

Coding rate/ Guard interval 1/4 1/8 1/16 1/32

1/2 **14.929 16.588 17.564 18.096**

2/3 **19.906 22.118 23.419 24.128**

3/4 **22.394 24.882 26.346 27.144**

5/6 **24.882 27.647 29.273 30.160**

7/8 **26.126 29.029 30.737 31.668**

Appendix C: Related readings

Some interesting readings from Academy related to Avalpa Broadcast Server are:

Open Source End-2-End DVB-H Mobile TV services and network infrastructure — The DVB-H pilot in Denmark

about how OpenCaster has been used into dvb-h integration

UITBOUWEN VAN EEN TESTOPSTELLING VOOR TESTEN VAN MHP-FUNCTIES VOOR DIGITALE TELEVISIEONTVANGERS

about how OpenCaster has been used for MHP conformance test

An open source software framework for DVB-* transmission

Acm paper presented jointly with ftw. Telecommunications Research Center Vienna, Vienna, Austria at 16th ACM international conference on Multimedia Vancouver, British Columbia, Canada

Appendix D: Mpeg2 transport stream overview

Available from courtesy of Prof. Antonio Navarro from Aveiro University