

TSDuck



**an extensible toolkit for
MPEG/DVB transport streams**

TSDuck Version 3.9



Agenda

- TSDuck overview
- TS utilities
- Transport stream processor
- XML table compiler
- Extending TSDuck
- Using TSDuck as an MPEG/DVB library for C++



TSDuck overview

- Process ISO/IEC 13818-1 transport streams
- Set of low-level utilities
 - extensible through plugins
- « Batch & Bash » oriented
 - command-line only, no fancy GUI
 - one utility or plugin = one elementary function
 - can be combined in any order
- Written in C++
 - reusable and extensible code
- Available on Linux, Windows and macOS



TSDuck sample usages (1/2)

- TS acquisition (satellite, terrestrial, IP, etc.)
- TS analysis
- Transmodulation
- Analysis, edition, injection of PSI / SI
 - using and editing PSI/SI in XML format
- TS packets carousel generation
 - packetization of SSU, etc.
- MPE injection and extraction (Multi-Protocol Encapsulation)



TSDuck sample usages (2/2)

- Test bed for CAS or STB
 - injection of test cases
 - DVB Scrambling and DVB SimulCrypt support
- Extraction of specific streams
 - T2-MI (DVB-T2 Modulator Interface)
 - PLP's (Physical Layer Pipe)
 - Teletext subtitles
 - SCTE 35 splicing
- Any combination of the above and more...

TSDuck availability



- Web site
<https://tsduck.github.io/>
- Open-source code
<https://github.com/tsduck/tsduck>
- BSD license
 - liberal, no GPL-like contamination
- Installation
 - pre-built binary installers for Windows, Fedora, Ubuntu, Raspbian
 - using Homebrew on macOS

TSDuck documentation



- User's Guide

- <https://github.com/tsduck/tsduck/raw/master/doc/tsduck.pdf>

- utilities reference
 - tsp plugins reference
 - sample usages

- Programmer's Reference

- <https://tsduck.github.io/doxy/html/>

- generated by Doxygen from source code
 - C++ common code reference
 - writing tsp plugins guidelines



TS Utilities

the command line utilities summary



TS utilities : data & devices

- Transport stream file
 - raw binary file, sequence of 188-byte TS packets
 - use ***tsresync*** to convert 204-byte packets or corrupted files
 - by default, use standard input & output
 - can use pipes from / to any DVB source
- PSI / SI file
 - raw binary file, sequence of sections
- Specialized hardware
 - DVB-S, DVB-T, DVB-C tuners (cheap CE devices)
 - Dektec modulators and ASI input / output (PCI, USB)
 - smartcards
 - on Linux and Windows but not macOS



TS utilities summary (1/4)

- Transport stream processor
 - ***tsp*** : processing framework using plugins
- TS analysis
 - ***tsanalyze*** : synthetic report
 - TS structure, services, PID's
 - can also produce a « normalized » output for automatic processing
 - ***tspsi*** : detailed analysis of main PSI / SI tables in TS
 - PAT, CAT, PMT, SDT, NIT, BAT
 - ***tsbitrate*** : evaluate original bitrate from PCR's
 - ***tsdate*** : extract date & time information



TS utilities summary (2/4)

- Transport packet analysis
 - ***tsdump*** : dump and analyze transport packets
- TS files recovery
 - ***tsresync*** : fix corrupted capture files
 - ***tsftrunc*** : truncate TS files
 - ***tsfixcc*** : fix continuity counters



TS utilities summary (3/4)

- PSI / SI tables
 - **tstables** : extract sections & tables from TS
 - either binary or textual analysis
 - **tstabdump** : textual analysis of binary table files
 - **tspacketize** : generate TS packets from tables
 - sample usage : delivery of packet carousel for tables
 - **tstabcomp** : table compiler from XML source files
 - also a decompiler which generates XML from captured binary tables

TS utilities summary (4/4)



- Various DVB hardware support
 - ***tsdektec*** : control Dektec devices
 - ***tslsdvb*** : list DVB receiver devices
 - ***tsscan*** : scan frequencies in a DVB network
 - ***tsterinfo*** : compute various DVB-T information
 - ***tssmartcard*** : list or reset smartcard reader devices



TSP

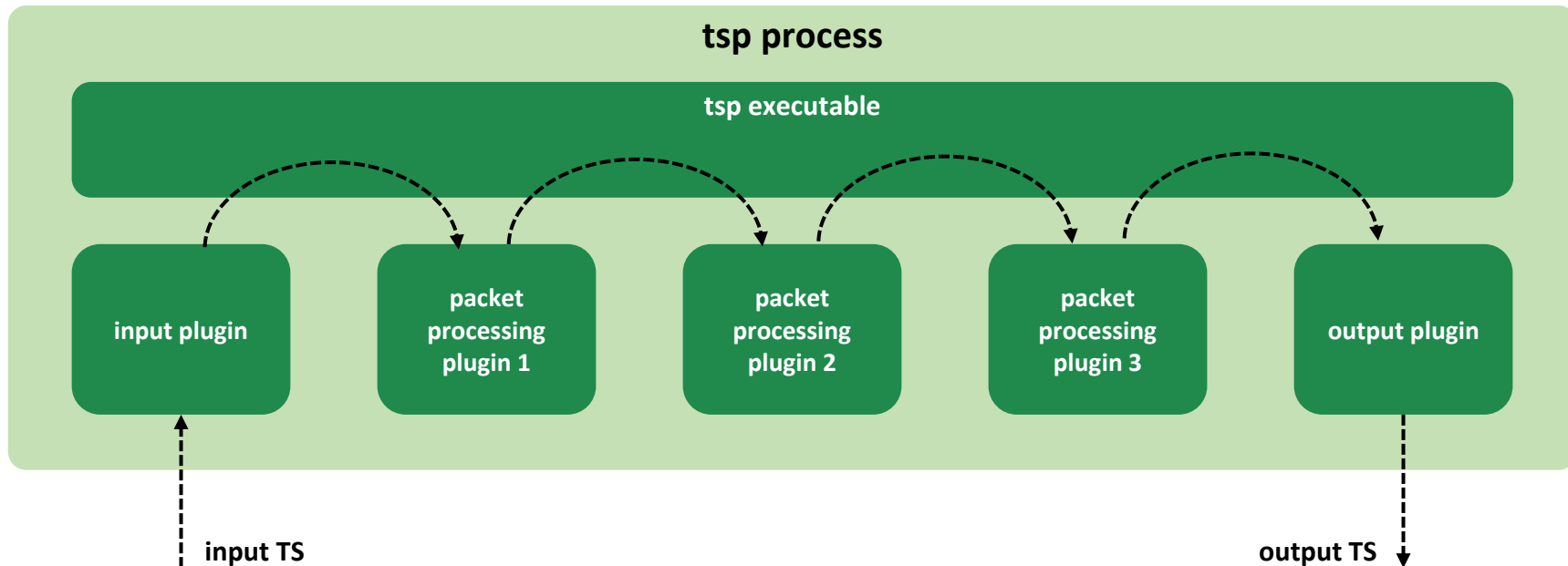
the transport stream processor



TSP overview

- Transport stream processing framework
 - Combination of elementary processing using plugins
 - One input plugin
 - receive a TS from various sources
 - Any number of packet processing plugins
 - perform transformations on TS packets
 - may remove packets
 - may NOT add packets
 - One output plugin
 - send the resulting TS to various destinations

TSP processing overview





TSP plugins

- Each tsp plugin is a shareable library
 - .so file on Linux and macOS
 - .dll file on Windows
- File naming
 - plugin named *foo* in file `tsplugin_foo.so` (or .dll)
 - same directory as tsp executable

- General command line syntax

```
tsp [tsp-options]  
    [-I input-name [input-options]]  
    [-P processor-name [processor-options]] ...  
    [-O output-name [output-options]]
```



TSP examples (1/5)

- TS acquisition

```
tsp -I dvb --uhf 21  
-P until --seconds 20  
-O file capture.ts
```

capture DVB-T stream from UHF channel 21

pass packets during 20 seconds, then stop

save TS to file capture.ts

- Display the PMT of a selected service

```
tsp -I dvb --uhf 35  
-P zap france2  
-P sifilter --pmt  
-P tables --max 1  
-O drop
```

extract service « France 2 », rebuild SPTS

extract PID containing PMT

display one table, then stop

drop output packet (don't care)



TSP examples (2/5)

- Transmodulation of a service over IP multicast

```
tsp -I dvb --uhf 35  
-P zap france2 --audio fra  
-O ip 224.10.11.12:1000
```

extract service « France 2 »,
keeping only one audio track

broadcast resulting SPTS to
multicast IP address:port

- On-the-fly replacement of a PSI / SI table

```
tsp -I dvb --uhf 24  
-P inject nit.bin --pid 16 --replace --stuffing  
-O dektec --uhf 24 --convolution 2/3 --guard 1/32
```

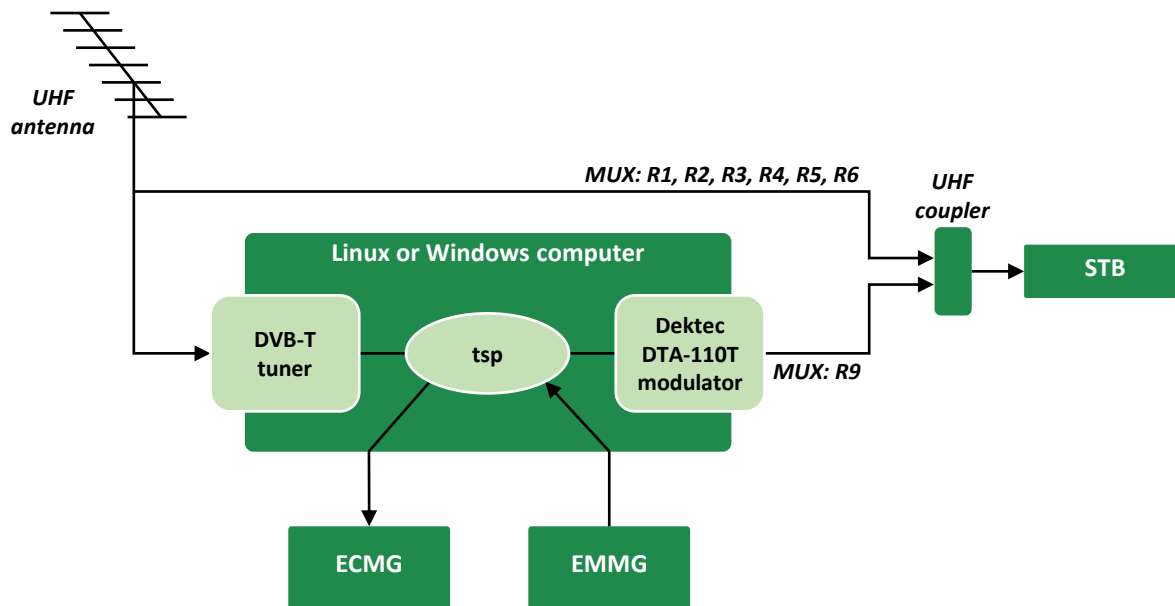
replace content of PID 16 with
table from binary file

send modified TS to a Dektec DVB-T
modulator on same frequency

TSP examples (3/5)



- Conditional Access System test bed
 - example using French DVB-T network



TSP examples (4/5)



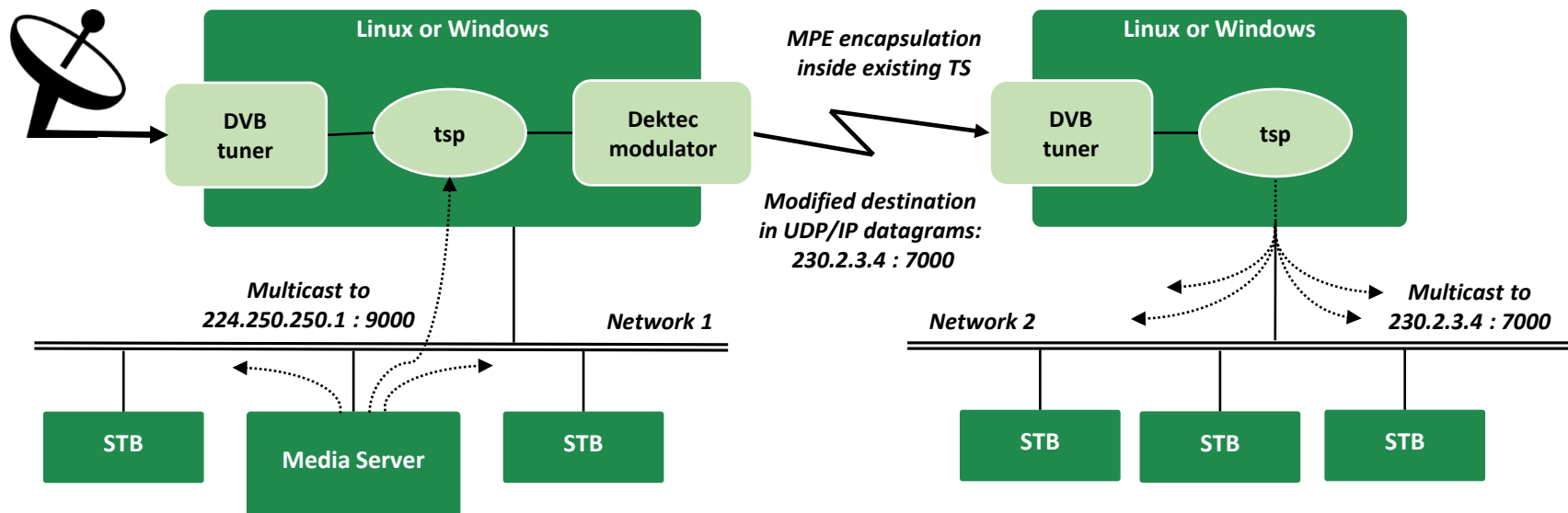
- Conditional Access System test bed (continued)

```
tsp -I dvb -u $UHF_INPUT
-P tsrename -t 9 -a
-P svrename direct8 -i 0x0901 -l 41 -n "Direct 8 Test"
-P svrename bfmtv -i 0x0903 -l 42 -n "BFM TV Test"
-P svrename 'i>tele' -i 0x0904 -l 43 -n "i>TELE Test"
-P svrename virgin17 -i 0x0905 -l 44 -n "Virgin 17 Test"
-P svrename gulli -i 0x0906 -l 45 -n "Gulli Test"
-P svrename france4 -i 0x0907 -l 46 -n "France 4 Test"
-P svrename 0x02FF -i 0x09FF
-P scrambler GulliTest -e $ECMG -s $SUPER_CAS_ID
    -p $PMT_CADESC_PRIVATE -a $AC
    -b $ECM_BITRATE --pid $ECM_PID
-P cat -c -a $CAS_ID/$EMM_PID/$CAT_CADESC_PRIVATE
-P datainject -r -s $MUX_SERVER_PORT
    -b $EMM_MAX_BITRATE -p $EMM_PID
-O dektec -u $UHF_OUTPUT --convolution 2/3 --guard 1/32
```

TSP examples (5/5)



- MPE injection and extraction





TSP input & output plugins

- Input plugins
 - ***null*** : null packet generator
 - ***file*** : binary TS file
 - ***dektec*** : Dektec ASI device
 - ***dvb*** : DVB-S, DVB-T, DVB-C receiver devices
 - ***ip*** : UDP/IP (unicast or multicast)
- Output plugins
 - ***drop*** : drop packets
 - ***file*** : binary TS file
 - ***dektec*** : Dektec ASI or modulator device
 - ***ip*** : UDP/IP (unicast or multicast)
 - ***play*** : render output using VLC, mplayer, xine, whichever is available



TSP processing plugins

- TS transformations
 - PID or packet filtering, PSI/SI transformation or injection, service extraction, time regulation, etc.
- TS analysis and monitoring
 - TS analysis, PSI/SI extraction, PID, bitrate monitoring, ECM or EMM monitoring, etc.
- TS scrambling & descrambling
 - DVB SimulCrypt support for ECM / EMM injection
- Any other processing you wish to develop...
 - 50 packet processing plugins available today



tstabcomp

the PSI / SI table compiler



Compiling PSI/SI tables

- Input source files
 - describe PSI/SI tables in text files
 - XML format
- Output binary files
 - concatenated list of sections
 - same format as used by other tools and plugins
- Reverse operation (decompilation) also available
 - input: binary sections file
 - output: XML file



Sample XML source file

```
<?xml version="1.0" encoding="UTF-8"?>
<tsduck>
  <PAT version="8" transport_stream_id="0x0012" network_PID="0x0010">
    <service service_id="0x0001" program_map_PID="0x1234"/>
    <service service_id="0x0002" program_map_PID="0x0678"/>
  </PAT>
  <PMT version="4" service_id="0x0456" PCR_PID="0x1234">
    <CA_descriptor CA_system_id="0x0777" CA_PID="0x0251"/>
    <component elementary_PID="0x0567" stream_type="0x12">
      <ISO_639_language_descriptor>
        <language code="fre" audio_type="0x45"/>
        <language code="deu" audio_type="0x78"/>
      </ISO_639_language_descriptor>
    </component>
  </PMT>
</tsduck>
```

- Reference format in user's guide



Typical application: manual table modification

- Tables can be used in XML or binary format anywhere
- Capture a table from a stream directly in XML format

```
tsp -I dvb ... \  
    -P tables --pid 16 --tid 0x40 --max 1 --xml nit.xml \  
    -O drop
```

- Manually edit the XML file with a text editor
- Inject the updated XML table in the stream

```
tsp -I dvb ... -P inject nit.xml --pid 16 ... -O dektec ...
```



Extending TSDuck

C++ transport stream programming



Extending TSDuck

- TSDuck is extensible
 - Source code provided

```
git clone https://github.com/tsduck/tsduck.git
```
 - Common API for Linux, Windows and macOS
 - DVB tuners and Dektec cards are not supported on macOS
 - Programmer's guide
 - Doxygen-generated, see <https://tsduck.github.io/>
- You can modify it yourself !



Why extending TSDuck ?

- Identify your needs
- Try to find a solution using existing TSDuck
 - review utilities and plugins
- Try to extend an existing utility or plugin
 - add new options
 - add features, don't modify existing behavior
 - remain upward compatible
- Develop your own plugin
 - it is quite simple, really
- Send your code back to TSDuck maintainer
 - so that everyone can benefit from it



Coding hints

- Don't write a plugin from scratch
 - use an existing one as code base
 - choose one which is technically similar
 - input? output? PSI/SI transformation? packet filtering?
- Implement simple & elementary features
 - preserve TSDuck philosophy
 - develop several elementary plugins if necessary
 - not a single big plugin implementing several features
- RTFM as usual !



Supported environments

- Linux
 - tested on Intel 32 & 64 bits (Fedora, Ubuntu), ARM 32 bits (Raspberry Pi)
- macOS
 - tested on macOS High Sierra 10.13
- Windows
 - tested on Intel 32 & 64 bits, Windows 7 & 10
 - Microsoft Visual Studio 2017 Community Edition
 - free download from [microsoft.com](https://www.microsoft.com), no license fee
 - NSIS (Nullsoft Scriptable Install System)
 - free software,
 - used to create TSDuck installer with precompiled binaries



Using TSDuck Library

to develop third-party applications



The TSDuck library

- All TSDuck common code is in one large library
 - `tsduck.so` / `tsduck.dll`
- Contains generic and reusable C++ code
 - basic operating system independent features
 - system, multi-treading, synchronization, networking, cryptography, etc.
 - MPEG / DVB features
 - TS packets, PSI/SI tables, sections and descriptors, demultiplexing, packetization, DVB tuners, etc.
- Can be used in your application
 - even if not part of TSDuck



Using TSDuck as a library

- Used in an application outside TSDuck
- Install the TSDuck development environment
 - Windows: “Development” option in installer
 - Ubuntu, Debian, Raspbian: package `tsduck-dev`
 - Fedora, Red Hat, CentOS: package `tsduck-devel`
- Typical application source file

```
#include "tsduck.h"  
... application code ...
```



Building with TSDuck library on UNIX

- Typical Linux Makefile

```
include /usr/include/tsduck/tsduck.mk  
... application-specific rules ...
```

- Typical macOS Makefile

```
include /usr/local/include/tsduck/tsduck.mk  
... application-specific rules ...
```



Building with TSDuck library on Windows

- Use Microsoft Visual Studio 2017
 - Community Edition is free
- Modify the application's project file (*app.vcxproj*)
 - Add one reference to the TSDuck property file

```
<Import Project="$(TSDUCK)\tsduck.props" />
```
 - Just before the final `</Project>` closing tag
 - And build the application as usual

Thank you

