Building TSDuck

Thierry Lelégard

Version 3.46-4810, September 2026

Contents

Preface

TSDuck is a free and open-source toolkit to manipulate MPEG Transport Streams.

This guide describes how to build and install TSDuck on various platforms. It is particularly useful if TSDuck has no pre-built binary package for your platform.

In the open source world, there are many different ways of building an application. Sometimes, building an open source application is straightforward. However, in other cases, it can be challenging, especially on non-Linux platforms.

TSDuck deliberately sides with the user, using simple and straightforward ways of building and installing.

On each platform, TSDuck always uses "native" mechanisms, tools, and procedures. The default compiler is always the "usual" one on the platform: GCC on Linux, Clang on macOS, Visual Studio on Windows. The packaging of installers, when available, is also "native": `.rpm` packages on Fedora and Red Hat clones, `.deb` packages on Ubuntu, Debian and the like, Homebrew distribution on macOS, NSIS-based executable installers on Windows. TSDuck also provides native support for Nix on Linux and macOS.

Structure of this guide:

- The chapter 1 describes how to build TSDuck.
- The chapter 2 describes how to build the documentation.
- The chapter 3 describes how to install TSDuck.

License

TSDuck is released under the terms of the license which is commonly referred to as "BSD 2-Clause License" or "Simplified BSD License" or "FreeBSD License". This is a liberal license which allows TSDuck to be used in a large number of environments. See the appendix A for more details.

Documentation format

The TSDuck guides are built using asciidoctor, from a set of text files which are maintained alongside the source code, in the same `git` repository.

This guide is formatted for HTML. The file tsduck-build.html is monolithic and self-sufficient, without reference to external images. Therefore, this HTML file can be downloaded, saved, and copied, as long as the license and content are not modified.

A PDF version tsduck-build.pdf is also available. However, due to limitations in the PDF generator of asciidoctor, the rendering is sometimes not as good as the HTML document.

Documentation set

The TSDuck documentation set is made of:

1. TSDuck User Guide (also from tsduck.io and in PDF format)
2. TSDuck Builder Guide, building and installing TSDuck (also from tsduck.io and in PDF format)
3. TSDuck Developer Guide (also from tsduck.io and in PDF format)
4. TSDuck Programming Reference

Chapter 1. Building TSDuck

TSDuck can be built on Windows, Linux, macOS and BSD systems.

Support for Dektec devices, DVB tuners and HiDes modulators is implemented only on Windows and Linux. MacOS and BSD systems can only support files and networking for TS input and output. AstroMeta-based modulators, however, are supported on all platforms.

Some protocols such as SRT and RIST require external libraries which may not be available on all platforms or all versions of a specific distro.

1.1. Tested systems

TSDuck has been tested on the following operating systems on at least one CPU architecture.

Table 1. Tested operating systems

OS	Variants
macOS	Intel and "Apple Silicon" (Arm64)
Windows	Intel (32 and 64 bits) and Arm64
Linux	Ubuntu, Debian, Raspbian, Mint, Fedora, Red Hat, CentOS, Rocky, Alma, openSUSE (Leap and Tumbleweed), Arch, Alpine, Gentoo, Slackware
BSD	FreeBSD, OpenBSD, NetBSD, DragonFlyBSD

On OpenBSD and DragonFlyBSD, there is a mixture of LibreSSL as core cryptographic library and several simultaneous versions of OpenSSL in the package manager. As a consequence, building or running TSDuck may fail in some configurations. This inconvenience is expected to disappear when LibreSSL in the core OS is replaced with a recent, stable and complete version of OpenSSL.

TSDuck has been tested on the following CPU architectures on at least one operating system.

Table 2. Tested CPU architectures

Architecture	Bits	Endianness
Intel x86	32 bits	Little endian
Intel x86-64	64 bits	Little endian
Armv7	32 bits	Little endian
Armv8	64 bits	Little endian
MIPS	32 and 64 bits	Little endian
RISC-V	64 bits	Little endian
PowerPC	64 bits	Big endian
IBM s390x	64 bits	Big endian

Some tests were done by contributors and were not verified (MIPS for instance). Some tests were performed using `qemu`, on an emulated platform, not a physical CPU. This is the case for RISC-V, PowerPC, IBM s390x.

TSDuck has been tested with the following compilers on at least one operating system.

Table 3. Tested compilers

Compiler	OS	Minimum version
GCC	Linux, BSD	13.0
Clang	Linux, macOS	17.0
MSVC	Windows	Visual Studio 2022

The required minimum version of each compiler depends on a correct implementation of C++20.

1.2. Building on Windows systems

On Windows systems, building a TSDuck installer simply means executing the PowerShell script `pkg\nsis\build-installer.ps1`. More details and options are provided in the next sections.

1.2.1. Pre-requisites

Operations in this section must be run once, before building TSDuck for the first time one a given Windows system. It should also be run to get up-to-date versions of the build tools and libraries which are used by TSDuck.

First, install Visual Studio Community Edition. This is the free version of Visual Studio. It can be downloaded here. If you already have Visual Studio Enterprise Edition (the commercial version), it is fine, no need to install the Community Edition.

Then, execute the PowerShell script `scripts\install-prerequisites.ps1`. It downloads and installs the requested packages which are necessary to build TSDuck on Windows.

If you prefer to collect the various installers yourself, follow the links to NSIS downloads, Git downloads, SRT downloads, RIST downloads, Dektec downloads, AstroMeta downloads, Java downloads, Python downloads, Doxygen downloads, Graphviz downloads.

TSDuck now requires a C++20 compliant compiler. C++20 support started with Visual Studio 2022, maybe 2019. We recommend to use Visual Studio 2022.

1.2.2. Building the binaries without installer

Execute the PowerShell script `scripts\build.ps1`. The TSDuck binaries, executables and DLL's, are built in directories named `bin\<target>-<platform>`, for instance `bin\Release-x64`, `bin\Debug-Win32`, or `bin\Release-ARM64`.

By default, the binaries are produced for the same architecture as the build system (typically Win64). To build for other targets, run the build script from a PowerShell window and add one or more options from `-Win32`, `-Win64`, `-Arm64`.

If you want to build the Arm64 version from an Intel Windows system, do not forget to install the latest version of the ARM64 build tools and libraries from the "individual components" list in the Visual Studio installer program. Be careful to select "ARM64" or "ARM64/ARM64EC" tools. Do not select "ARM" build tools, they target Arm32 architectures.

To cleanup the repository tree and return to a pristine source state, execute the PowerShell script `scripts\cleanup.ps1`.

1.2.3. Building the Windows installers

Execute the PowerShell script `pkg\nsis\build-installer.ps1`.

As with `scripts\build.ps1`, only one installer is built by default, for the same architecture as the build system. To build other installers, run the build script from a PowerShell window and add one or more options from `-Win32`, `-Win64`, `-Arm64`.

There is no need to build the TSDuck binaries before building the installers. Building the binaries, is part of the installer build.

All installation packages are dropped into the subdirectory `pkg/installers`. The packages are not deleted by the cleanup procedures. They are not pushed into the git repository either.

1.2.4. Installing in non-standard locations

On systems where you have no administration privilege and consequently no right to use the standard installers, you may want to manually install TSDuck is some arbitrary directory.

On Windows systems, a so-called *portable* package is built with the installers. This is a zip archive file which can be expanded anywhere. It is automatically built by `pkg\nsis\build-installer.ps1`, in addition to the executable installer.

1.2.5. Running from the build location

It is sometimes useful to run a TSDuck binary, `tsp` or any other, directly from the build directory, right after compilation. This can be required for testing or debugging.

The commands can be run using their complete path without additional setup. For instance, to run the released 64-bit version of `tsp`, use:

```
PS C:\tsduck> bin\Release-x64\tsp.exe --version
tsp: TSDuck - The MPEG Transport Stream Toolkit - version 3.12-730
```

For other combinations (release vs. debug and 32 vs. 64 bits), the paths from the repository root are:

```
bin\Release-x64\tsp.exe
bin\Release-Win32\tsp.exe
bin\Debug-x64\tsp.exe
bin\Debug-Win32\tsp.exe
```

1.3. Building on UNIX systems (Linux, macOS, BSD)

On UNIX systems (Linux, macOS, BSD), building TSDuck simply means typing `make` in the top level directory of the project. More details and options are provided in the next sections.

TL;DR

If you don’t like the details, just run this:

```
$ git clone https://github.com/tsduck/tsduck.git
$ cd tsduck
$ scripts/install-prerequisites.sh
$ make -j10 default docs-html
$ sudo make install
```

If you like thinking before doing, we recommend to read the following sections.

1.3.1. Pre-requisites

Operations in this section must be run once, before building TSDuck for the first time one a given system.

Execute the shell-script `scripts/install-prerequisites.sh`. It downloads and installs the requested packages which are necessary to build TSDuck. The list of packages and how to install them depend on the operating system distribution and version.

If you intend to use exclusion options in the `make` command line (for instance `NOSRT=1 NORIST=1`), specify them to `scripts/install-prerequisites.sh` too. This will prevent the installation of unused libraries.

In addition to the `make` exclusion options, `install-prerequisites.sh` supports `NODOXYGEN=1`.

Currently, this script supports all UNIX operating systems which were listed in Table 1.

Since all packages are pulled from the standard repositories of each distro, there is generally no need to re-run this script later. The packages will be updated as part of the system system updates. Note, however, that a new version of TSDuck may require additional dependencies. In case of build error, it can be wise to run `scripts/install-prerequisites.sh` again and retry.

Although TSDuck has been built and tested on Slackware, the script `install-prerequisites.sh` does not support this distro yet. Slackware is not very friendly for automation. Some package shall be individually searched for a specific version and installed by hand. It has not been possible to find an automated way to setup the required environment to build TSDuck. Should this be possible, contributions from Slackware experts are welcome.

1.3.2. C++ compiler requirements

C++ language version

TSDuck now requires a C++20 compliant compiler. C++20 support in GCC offically starts with version 11. However, because of several bugs in the compiler, TSDuck can only be compiled with GCC 13 or higher.

Most recent Linux distros use GCC 13. Some older distros which come with older GCC versions may propose alternative GCC packages with more recent versions.

If your distro is too old and doesn't provide any GCC 11 package, then you cannot build TSDuck version 3.36 and higher. On such systems, the highest TSDuck version which can be built is 3.35. Similarly, if your distro doesn't provide any GCC 13 package, then you cannot build TSDuck version 3.40 and higher. On such systems, the highest TSDuck version which can be built is 3.39. This is the cost of obsolescence...

Using Clang as an alternative to GCC

If your distro is too old and doesn't provide any GCC 13 package, another alternative is to use LLVM/Clang. Most distros with old versions of GCC provide decently recent versions of Clang. To force a build with LLVM/Clang instead of GCC, defined the `make` variable `LLVM`:

The minimum required version for Clang is 17.0. C++20 support in Clang was gradually included from versions 8 to 16. However, just like GCC 11, the initial versions are buggy and TSDuck can be built only with Clang 17 onwards.

```
$ make LLVM=1 ....
```

However, when the installed GCC is really old (typically before GCC 8), using Clang may not work either because Clang uses the GCC C/C++ standard libraries and their header files. If the GCC issue is a compilation issue on GCC 8 to 10, using Clang may work. With older versions of GCC, using Clang probably does not work because the corresponding standard library does not contain the C++20 features.

Red Hat example

As of Red Hat Entreprise Linux 9.5, the default GCC version is 11, which does not correctly support C++20.

However, you can install and use GCC 13 using the following commands:

```
$ sudo dnf install gcc-toolset-13 gcc-toolset-13-gcc-c++ gcc-toolset-13-runtime \
                   gcc-toolset-13-binutils gcc-toolset-13-libatomic-devel
$ source /opt/rh/gcc-toolset-13/enable
$ make ...
```

The first command installs the GCC 13 packages. The second command defines the required environment variables in the current process. The last one builds TSDuck.

On RHEL, the GCC 13 packages are available in the `appstream` repository. Make sure to have activated it first.

Other Linux distros

Older versions of other distros such as Ubuntu, Debian and others have equivalent alternative packages for GCC 13, with different names, when they come with an older version of GCC.

If there is no `enable` script (as in the example above) to setup the environment, you need to define the following variables, either as environment variables or on the make command line. The provided values are examples only and may be different in specific environments.

```
$ make CXX=g++-13 CC=gcc-13 GCC=gcc-13 CPP="gcc-13 -E" AR=gcc-ar-13 ...
```

Since `make` uses the environment for the initial values of its variables, it is also possible to define them as environment variables in some initialization script instead of using such a complex `make` command..

NetBSD example

As of this writing, the most recent version of NetBSD is 9.3, which comes with GCC 7.5.

More recent GCC packages are available. To install GCC 14:

```
$ sudo pkgin install gcc14 gcc14-libs
```

The compilation environment is installed in `/usr/pkg/gcc14`. Using GCC 14 is enabled by adding `/usr/pkg/gcc14/bin` at the beginning of the `PATH`:

```
$ export PATH="/usr/pkg/gcc14/bin:$PATH"
```

However, linker options shall be passed to `gmake`:

```
$ gmake LDFLAGS_EXTRA="-L/usr/pkg/gcc14/lib -Wl,-rpath=/usr/pkg/gcc14/lib" ...
```

Note the command `gmake`, the GNU Make command. See section 1.3.3 for more details.

Since `gmake` uses the environment for the initial values of its variables, it is also possible to define `LDFLAGS_EXTRA` as environment variables in some initialization script.

DragonFlyBSD example

As of this writing, the most recent version of DragonFlyBSD is 6.4.0, which comes with GCC 8.3. Even though

DragonFlyBSD is supposed to be based on FreeBSD, its GCC version is way behind FreeBSD version 14.0 which comes with GCC 12.2.

More recent GCC packages are available for DragonFlyBSD. To install GCC 14:

```
$ sudo pkg install gcc14
```

However, because all *BSD systems are carefully incompatible between each other, using the alternative compiler is very different from NetBSD.

Building TSDuck:

```
$ gmake CXX=g++14 CC=gcc14 GCC=gcc14 CPP="gcc14 -E" AR=gcc-ar14 LDFLAGS_EXTRA="-Wl,-
rpath=/usr/local/lib/gcc14" ...
```

Again, since `gmake` uses the environment for the initial values of its variables, it is also possible to define them as environment variables in some initialization script instead of using such a complex `gmake` command.

1.3.3. GNU Make requirements

The makefiles in the TSDuck project use a GNU Make syntax. TSDuck requires GNU Make version 4 or higher. The makefiles are not compatible with the non-GNU versions of the `make` command or GNU Make version 3 or lower.

All Linux distros which are less than ten years old have a compatible GNU Make.

GNU Make on macOS

On macOS, GNU Make is the default `make` command and is installed in `/usr/bin`. However, because the GNU Make developers switched their license from GLPv2 to GPLv3, recent versions of GNU Make can no longer be distributed with macOS. Therefore, the preinstalled GNU Make on macOS is version 3.81, which is incompatible with some TSDuck makefiles.

Installing the latest version of GNU Make on macOS is straightforward using HomeBrew. The script `install-prerequisites.sh` installs it, as part of all prerequisites. However, to avoid interfering with the preinstalled `/usr/bin/make`, the command is installed in `/opt/homebrew/bin` as `gmake`.

For convenience, when GNU commands which are installed by HomeBrew interfere with standard system commands, HomeBrew provides a `libexec/gnubin` alternative, a directory where the command is available under its native name, here `make`.

Therefore, there are two solutions to use the latest GNU Make on macOS:

1. Use command `gmake` instead of `make` all the time.
2. Add `/opt/homebrew/opt/make/libexec/gnubin` in the `PATH` (example below).

We recommend the second option and add the following line in your `.bashrc` file:

```
export PATH="$(brew --prefix)/opt/make/libexec/gnubin:$PATH"
```

The decision to switch from GLPv2 to GPLv3 was a very counter-productive idea. It does not prevent using more recent versions of GNU Make on macOS, it just makes it more painful. And being a pain is counter-productive, to say the least (and remain polite).

GNU Make on BSD systems

On FreeBSD, OpenBSD, NetBSD, DragonFlyBSD, the standard BSD `make` command is the old `make` tool, before GNU,

which uses an old and restricted syntax. It is incompatible with GNU Make. As part of prerequisites for BSD systems, GNU Make is installed under the name `gmake`.

In all build commands in this document, when `make` is mentioned, use `gmake` on all BSD systems.

1.3.4. Hardware device libraries

Dektec DTAPI: The command `make` at the top level will automatically download the LinuxSDK from the Dektec site. There is no manual setup for DTAPI on Linux. Note that the Dektec DTAPI is available only for Linux distros on Intel CPU's with the GNU libc. Non-Intel systems (for instance Arm-based devices such as Raspberry Pi) cannot use Dektec devices. Similarly, Intel-based distros using a non-standard libc (for instance Alpine Linux which uses musl libc) cannot use Dektec devices either.

AstroMeta API: On Linux, the command `make` at the top level will automatically download the Linux version of the AstroMeta API from the GitHub. There is currectly no Linux package for the AstroMeta API in the standard distros. On Windows and macOS, binary packages are available and are installed by the `install-prerequisites` scripts. Using AstroMeta devices on BSD systems is currently not supported but should work if necessary (accessing AstroMeta devices is performed through `libusb` and not a specific kernel driver).

1.3.5. Building the TSDuck binaries alone

Execute the command `make` at top level.

The TSDuck binaries, executables and shared objects (`.so` or `.dylib`), are built in directory `bin/release-<arch>-<hostname>` by default. Consequently, the same work area can be simultaneously used by several systems. Each system builds in its own area. You can also override the build directory using `make BINDIR=⋯`.

Note that TSDuck contains thousands of source files and building it can take time. However, since most machines have multiple CPU's, all makefiles are designed for parallel builds. On a quad-core machine with hyperthreading (8 logical cores), for instance, the command `make -j10` is recommended (10 parallel compilations), reducing the total build time to a few minutes.

As an example, on an Intel system from 2020, building TSDuck without parallelism takes several hours. On the same system, using `-j10`, it takes 20 minutes. On a recent iMac M3, using `-j10`, the build time is 2 minutes.

To cleanup the repository tree and return to a pristine source state, execute `make clean` at the top level.

1.3.6. Building without specialized dependencies

In specific configurations, you may want to disable some external libraries such as `libcurl` or `pcsc-lite`. Of course, the corresponding features in TSDuck will be disabled but the impact is limited. For instance, disabling `libcurl` will disable the input plugins `http` and `hls`.

The following `make` variables can be defined:

`NOTEST`	Do not build unitary tests.
`NODEKTEC`	No Dektec device support, remove dependency to `DTAPI`.
`NOHIDES`	No HiDes device support.
`NOVATEK`	No AstroMeta device support (modulators based on AstroMeta chips, formerly VATek), remove dependency to `libvatek`.
`NOOPENSSL`	No cryptographic support, remove dependency to `openssl`.
`NOZLIB`	Don't use zlib, use embedded "Small Deflate" instead, remove dependency to `zlib`.
`NOSDEFL`	Don't build "Small Deflate", only in case of compilation issue.

NOCURL	No HTTP support, remove dependency to libcurl.
NOPCSC	No smartcard support, remove dependency to pcsc-lite.
NOEDITLINE	No interactive line editing, remove dependency to libedit.
NOSRT	No SRT support (Secure Reliable Transport), remove dependency to libsrt.
NORIST	No RIST support (Reliable Internet Stream Transport), remove dependency to librist.
NOJAVA	No Java bindings.
NOPYTHON	No Python bindings.
NOHWACCEL	Disable hardware acceleration such as crypto instructions.
ASSERTIONS	Keep assertions in production mode (slower code).

The following command, for instance, builds TSDuck without dependency to pcsc-lite, libcurl and Dektec DTAPI:

```
$ make NOPCSC=1 NOCURL=1 NODEKTEC=1
```

The variables NOJAVA and NOPYTHON remove the bindings for the Java and Python languages, respectively. However, they do not remove any external dependency because these bindings do not need any. Therefore, removing them does not bring any benefit in terms of dependencies on the target system.

They do not bring any benefit in terms of build system either. Building the Python bindings does not require any specific environement. And if the Java Development Kit (JDK) is not installed on the build system, the Java bindings are not built anyway, even without explicit NOJAVA.

For a complete list of the variables which are used by make, see the file CONFIG.txt at the root of the TSDuck source tree.

When a set of make variables are used to build TSDuck, the exact same set of variables shall be used in all make commands on this build. For instance, if TSDuck is built using make NOSRT=1 NORIST=1, the installation package shall be built using make installer NOSRT=1 NORIST=1. Failing to provide a consistent set of variables across successible make commands on the same build may produce unexpected results.

You may create independent builds with distinct options. You simply need to redirect the produced binaries in distinct subdirectories. For instance, the following commands build two separate versions of TSDuck in two subdirectories, one without SRT support, the other without RIST support.

```
$ make -j10 NOSRT=1 BINDIR_SUFFIX=-nosrt
$ make -j10 NORIST=1 BINDIR_SUFFIX=-norist
$ make installer NORIST=1 BINDIR_SUFFIX=-norist
```

The last command builds a binary package for the version without RIST support. Note that the same set of variables shall be used to locate the right binaries and options.

1.3.7. Building with specific debug capabilities

The following additional make variables can be defined to enable specific debug capabilities:

DEBUG	Compile with debug information and no optimization.
GPROF	Compile with code profiling using gprof.
GCOV	Compile with code coverage using gcov.

`ASAN` Compile with code sanitizing using AddressSanitizer with default optimization.

`UBSAN` Compile with code sanitizing using UndefinedBehaviorSanitizer with default optimization.

All these options produce binaries in distinct subdirectories. Therefore, they can be considered as independent builds which do not interfere with each other.

Release builds are created in directories starting with `bin/release-⋯`. Debug builds (`DEBUG=1`) are created in directories starting with `bin/debug-⋯`. The other above-listed options redefine `BINDIR_SUFFIX` with a meaningful suffix.

1.3.8. Displaying full build commands

Because of the number of include directories and warning options, the compilation commands are very long, typically more than 4000 characters, 30 to 50 lines on a terminal window. If the `make` commands displays all commands, the output is messy. It is difficult to identify the progression of the build. Error messages are not clearly identified.

Therefore, the `make` command only displays a synthetic line for each command such as:

```
[CXX] dtv/tables/dvb/tsAIT.cpp
[CXX] dtv/tables/atsc/tsATSCEIT.cpp
[CXX] dtv/tables/tsAbstractDescriptorsTable.cpp
```

In some cases, if can be useful to display the full compilation commands. To do this, define the variable `VERBOSE` as follow:

```
$ make VERBOSE=1
```

For convenience and compatibility with some tradition, `V` can be used instead of `VERBOSE`.

1.3.9. Building the TSDuck installation packages

Execute the command `make installer` at top level to build all packages.

```
$ make installer
```

Depending on the platform, the packages can be `.deb` or `.rpm` files. There is currently no support to build an installation package on other Linux distros and BSD systems.

There is no need to build the TSDuck binaries before building the installers. Building the binaries, when necessary, is part of the installer build.

All installation packages are dropped into the subdirectory `pkg/installers`. The packages are not deleted by the cleanup procedures. They are not pushed into the git repository either.

After building the installation packages, it is possible to collect them into one single "tarball" archive using the command `make installer-tarball`. The resulting archive file is also dropped into `pkg/installers`.

The name of the installation packages and the tarball archive depend on the package manager, the processor architecture, and the version of TSDuck. The name of the packages cannot be changed because they need to follow the rules of the corresponding package manager. However, the name of the tarball can be changed as follow:

```
$ make installer-tarball INSTALLER_TARBALL=/tmp/tsduck.tgz
```

On macOS, there is no binary package for TSDuck on macOS. On this platform, TSDuck is installed using Homebrew, a package manager for open-source projects on macOS. See section 3.2 for more details.

1.3.10. For packagers of Linux distros

Packagers of Linux distros may want to create TSDuck packages. The build methods are not different. This section contains a few hints to help the packaging.

By default, TSDuck is built with capabilities to check the availability of new versions on GitHub. The `tsversion` command can also download and upgrade TSDuck from the binaries on GitHub. Packagers of Linux distros may want to disable this since they may prefer to avoid mixing their TSDuck packages with the generic TSDuck packages on GitHub. To disable this feature, build TSDuck with `make NOGITHUB=1`.

The way to build a package depends on the package management system. Usually, the build procedure includes an installation on a temporary fake system root. To build TSDuck and install it on `/temporary/fake/root`, use the following command:

```
$ make NOGITHUB=1 install SYSROOT=/temporary/fake/root
```

It is recommended to create two distinct packages: one for the TSDuck tools and plugins and one for the development environment. The development package shall require the pre-installation of the tools package.

If you need to separately build TSDuck for each package, use `make` targets `install-tools` and `install-devel` instead of `install` which installs everything.

```
$ make NOGITHUB=1 install-tools SYSROOT=/temporary/fake/root
$ make NOGITHUB=1 install-devel SYSROOT=/temporary/fake/root
```

1.3.11. Installing in non-standard locations

On systems where you have no administration privilege and consequently no right to use the standard installers, you may want to manually install TSDuck is some arbitrary directory.

You have to rebuild TSDuck from the source repository and install it using a command like this one:

```
$ make install SYSPREFIX=$HOME/usr/local
```

Unlike many open source applications on Linux, the TSDuck binaries are independent from the installation locations. There is no equivalent to `./configure --prefix ⋯`. The same binaries can be installed in different locations, provided that the installation is consistent (typically using `make install ⋯`).

The TSDuck commands are located in the `bin` subdirectory and can be executed from here without any additional setup. It is probably a good idea to add this `bin` directory in your `PATH` environment variable.

1.3.12. Using pkgconfig after installation

Applications may use the `pkgconfig` utility to reference the TSDuck library. A file named `tsduck.pc` is installed in the appropriate directory.

However, `pkgconfig` has its own limitations, specifically regarding the configured compilation options.

TSDuck is a C++ library which requires a minimum revision of the language. Currently, the minimum revision is C++20. All more recent revisions are supported. By default, most C++ compilers are based on older revisions. Therefore, compiling an application using TSDuck with the default options fails. At least, `-std=c++20` is required. To avoid compilation problems with most applications, `-std=c++20` is enforced in `tsduck.pc`.

However, some applications may need to explicitly specify an even more recent revision, such as `-std=c++20`, which conflicts with `-std=c++20` in `tsduck.pc`.

For that use case, you may install TSDuck without reference to the C++ revision using the following command:

```
$ make install NOPCSTD=1
```

The counterpart is that the applications *must* specify a `-std` option and the revision must be C++20 or more recent.

A generic solution would be that each library and the application all provide a *minimum* revision of the C++ language and pkgconfig would provide a synthetic `-std` option which fulfills all requirements. However, this feature does not exist in pkgconfig, hence this trick.

1.3.13. Running from the build location

It is sometimes useful to run a TSDuck binary, `tsp` or any other, directly from the build directory, right after compilation, without going through `make install`. This can be required for testing or debugging.

Because the binary directory name contains the host name, it is possible to build TSDuck using the same shared source tree from various systems or virtual machines. All builds will coexist using distinct names under the `bin` subdirectory.

For `bash` users who wish to include the binary directory in the `PATH`, simply "source" the script `scripts/setenv.sh`.

Example:

```
$ . scripts/setenv.sh
$ which tsp
/Users/devel/tsduck/bin/release-x86_64-mymac/tsp
```

This script can also be used with option `--display` to display the actual path of the binary directory. The output can be used in other scripts (including from any other shell than `bash`).

Example:

```
$ scripts/setenv.sh --display
/Users/devel/tsduck/bin/release-x86_64-mymac
```

Use `scripts/setenv.sh --help` for other options.

1.4. Building with Nix

TSDuck provides native Nix flakes for Linux and macOS. Nix must be installed on your system with Flakes support enabled.

To build from the original TSDuck repository, use one of the following commands:

```
$ nix build github:tsduck/tsduck?dir=pkg/nix              # latest on master
$ nix build github:tsduck/tsduck/<tag|branch>?dir=pkg/nix # specific tag or branch
$ nix build github:tsduck/tsduck?dir=pkg/nix#tsduck-min   # minimal, no hardware
```

For local development, use the following sequence of commands:

```
$ git clone https://github.com/tsduck/tsduck.git
$ cd tsduck
$ nix build ./pkg/nix
$ nix develop ./pkg/nix
```

To build the default TSDuck package:

```
$ nix build ./pkg/nix
```

To build a specific variant, use the # fragment to select the package name:

```
$ nix build ./pkg/nix#tsduck-min           # no hardware support
$ nix build ./pkg/nix#python-tsduck       # Python bindings (against tsduck)
$ nix build ./pkg/nix#python-tsduck-min   # Python bindings (against tsduck-min)
$ nix build ./pkg/nix#java-tsduck         # Java bindings (against tsduck)
$ nix build ./pkg/nix#java-tsduck-min     # Java bindings (against tsduck-min)
```

The build result will be available in the `./result` symlink.

Nix provides development shells with all build dependencies pre-configured:

```
$ nix develop ./pkg/nix          # Main development shell with all tools
$ nix develop ./pkg/nix#python   # Python environment with TSDuck bindings
```

Within the development shell, you can use traditional build commands:

```
$ nix develop ./pkg/nix
[nix-shell]$ make
[nix-shell]$ make test
```

1.5. Build with alternative libraries

This section documents how to build TSDuck using some alternatives to the standard libraries which are used by default.

1.5.1. Build with Robotweax SRT library

By default, on most operating systems, the SRT library is the Haivision libsrt (see [Haivision-SRT]). This library is available in standard package managers for all operating systems. Haivision is the company which originally specified the SRT protocol and they published the first open source implementation of an SRT library.

Later, Robotweax, a company which was founded by former Haivision employees, published an alternative version of the SRT library (see [Robotweax-SRT]). This library was designed to be compatible with the original libsrt

and interoperable with it. It is expected to be lighter, with a more modern implementation, than the original libsrt.

By default, TSDuck is built with the SRT library which is installed on the build system. It is possible to explicitly build TSDuck with the Robotweax SRT library, even if the standard libsrt is installed on the build system.

Building with Robotweax SRT on UNIX systems

So far, Robotweax SRT is not available as a standard package in any Linux package manager, or Homebrew, or FreeBSD Ports. Its source code must be pulled from its GitHub repository and it must be locally rebuilt.

To build Robotweax SRT and install it on a specific directory, for instance `/opt/robotweax`, use the following commands:

```
$ git clone https://github.com/Robotweax/srt.git robotweax-srt
$ cd robotweax-srt
$ cmake -S . -B build -DCMAKE_BUILD_TYPE=Release -DCMAKE_POSITION_INDEPENDENT_CODE=ON
$ cmake --build build --parallel
$ cmake --install build --prefix /opt/robotweax
```

Then, build TSDuck with symbol or environment variable `ROBOTWEAX_SRT_DIR` pointing to the Robotweax SRT installation prefix:

```
$ make ROBOTWEAX_SRT_DIR=/opt/robotweax
```

You may of course install Robotweax SRT anywhere else and use other common `make` options, especially `-j` to parallelize the huge TSDuck build.

When building with Robotweax SRT on UNIX systems, the SRT library is statically linked inside the TSDuck library (`libtsduck.so` or `libtsduck.dylib`). The compiled TSDuck package can therefore be installed on any other system without deploying Robotweax SRT. This is different from the default Haivision libsrt where TSDuck is usually linked against the `libsrt.so` shared library and always uses the `libsrt.so` shared libray of the target systems.

Building with Robotweax SRT on Windows systems

The Robotweax team provides a binary SDK for their SRT library on Windows. It can be found as an asset of the corresponding release on GitHub (see [Robotweax-SRT]).

The SDK exists in two versions, one with an OpenSSL cryptographic backend, and one with a Microsoft BCrypt (CNG) backend. The BCrypt backend uses the standard native Microsoft cryptographic library, as installed on all Windows systems. The SDK with OpenSSL backend comes with all versions of the OpenSSL static libraries. These OpenSSL libraries are then linked with the final application (i.e. `tsduck.dll` in practice).

All versions of the SDK can be installed and coexist. They can also coexist with the Haivision libsrt package.

On Windows, the TSDuck build system automatically selects the preferred version of the SRT library, depending on which are installed on the system. The first available and installed one is used, in order of preference:

- Robotweax SRT with Microsoft BCrypt (CNG) backend.
- Robotweax SRT with OpenSSL backend.
- Haivision original libsrt.
- No SRT support.

Therefore, installing the Robotweax SRT SDK is sufficient to rebuild TSDuck with Robotweax SRT.

In case of doubt, the actual version can be displayed using options `--version=srt` or `--version=all`:

```
PS C:\tsduck> bin\Release-x64\tsversion.exe --version=srt
libsrt version 1.5.7 (Robotweax SRT version 0.2.4 with Microsoft BCrypt backend)
```

1.6. Installer files summary

The following list summarizes the packages which are built and dropped into the `pkg/installers` directory, through a few examples, assuming that the current version of TSDuck is 3.40-4134.

`tsduck_3.40-4134.ubuntu24_amd64.deb`	Binary package for Intel 64-bit Ubuntu 24.x
`tsduck_3.40-4134.ubuntu24_arm64.deb`	Binary package for Arm 64-bit Ubuntu 24.x
`tsduck_3.40-4134.ubuntu24_riscv64.deb`	Binary package for RISC-V 64-bit Ubuntu 24.x
`tsduck_3.40-4134.ubuntu24_s390x.deb`	Binary package for IBM s390x 64-bit Ubuntu 24.x
`tsduck_3.40-4134.debian12_amd64.deb`	Binary package for Intel 64-bit Debian 12
`tsduck_3.40-4134.debian12_arm64.deb`	Binary package for Arm 64-bit Debian 12
`tsduck_3.40-4134.raspbian12_armhf.deb`	Binary package for Arm 32-bit Raspbian 12 (Raspberry Pi)
`tsduck-3.40-4134.el9.x86_64.rpm`	Binary package for Intel 64-bit Red Hat 9.x and clones
`tsduck-3.40-4134.el9.aarch64.rpm`	Binary package for Arm 64-bit Red Hat 9.x and clones
`tsduck-3.40-4134.el9.src.rpm`	Source package for Red Hat and clones
`tsduck-3.40-4134.fc41.x86_64.rpm`	Binary package for Intel 64-bit Fedora 41
`tsduck-3.40-4134.fc41.aarch64.rpm`	Binary package for Arm 64-bit Fedora 41
`tsduck-3.40-4134.fc41.src.rpm`	Source package for Fedora
`tsduck-dev_3.40-4134.ubuntu24_amd64.deb`	Development package for Intel 64-bit Ubuntu 24.x
`tsduck-dev_3.40-4134.ubuntu24_arm64.deb`	Development package for Arm 64-bit Ubuntu 24.x
`tsduck-dev_3.40-4134.ubuntu24_riscv64.deb`	Development package for RISC-V 64-bit Ubuntu 24.x
`tsduck-dev_3.40-4134.ubuntu24_s390x.deb`	Development package for IBM s390x 64-bit Ubuntu 24.x
`tsduck-dev_3.40-4134.debian12_amd64.deb`	Development package for Intel 64-bit Debian 12
`tsduck-dev_3.40-4134.debian12_arm64.deb`	Development package for Arm 64-bit Debian 12
`tsduck-dev_3.40-4134.raspbian12_armhf.deb`	Development package for Intel 32-bit Raspbian (Raspberry Pi)
`tsduck-devel-3.40-4134.el9.x86_64.rpm`	Development package for Intel 64-bit Red Hat 9.x and clones
`tsduck-devel-3.40-4134.el9.aarch64.rpm`	Development package for Arm 64-bit Red Hat 9.x and clones
`tsduck-devel-3.40-4134.fc41.x86_64.rpm`	Development package for Intel 64-bit Fedora 41
`tsduck-devel-3.40-4134.fc41.aarch64.rpm`	Development package for Arm 64-bit Fedora 41
`TSDuck-Win32-3.40-4134.exe`	Binary installer for Intel 32-bit Windows
`TSDuck-Win64-3.40-4134.exe`	Binary installer for Intel 64-bit Windows
`TSDuck-Arm64-3.40-4134.exe`	Binary installer for Arm 64-bit Windows
`TSDuck-Win32-3.40-4134-Portable.zip`	Portable package for Intel 32-bit Windows
`TSDuck-Win64-3.40-4134-Portable.zip`	Portable package for Intel 64-bit Windows
`TSDuck-Arm64-3.40-4134-Portable.zip`	Portable package for Arm 64-bit Windows

On Linux systems, there are two different packages. The package `tsduck` contains the tools and plugins. This is the only required package if you just need to use TSDuck. The package named `tsduck-devel` (Red Hat family) or `tsduck-dev` (Debian family) contains the development environment. It is useful only to build third-party applications which use the TSDuck library.

On Windows systems, there is only one binary installer which contains the tools, plugins, documentation and development environment. The user can select which components shall be installed. The development environment is unselected by default.

On macOS systems, the Homebrew package `tsduck` installs all components.

Chapter 2. Building the documentation

There are several TSDuck documents:

1. TSDuck User Guide (HTML and PDF)
2. TSDuck Builder Guide (HTML and PDF)
3. TSDuck Developer Guide (HTML and PDF)
4. TSDuck Programming Reference (HTML only)

All documents,except the last one, are written in Asciidoc format. Their HTML and PDF versions are built using Asciidoctor. The two HTML files are large standalone files, without reference to any other local file; they can be safely copied without breaking the navigation.

These guides are installed with TSDuck on UNIX systems (Linux, macOS, BSD) and Windows (HTML version only).

The TSDuck Programming Reference contains the documentation of all public classes which can be used by applications in C++, Java, or Python. This reference is built using Doxygen.

Asciidoctor and Doxygen are automatically installed by the scripts `install-prerequisites.sh` on UNIX systems (Linux, macOS, BSD) and `install-prerequisites.ps1` on Windows.

On large libraries, Doxygen is extremely verbose. The TSDuck Programming Reference is made of a large number of HTML files, more than 14,000 files and directories. It also takes some time to generate. Therefore, the Programming Reference is neither built by default nor installed with the rest of TSDuck. Every night, a fresh copy is generated and published online at https://tsduck.io/doxy.

2.1. Building on Windows

The user guide and the developer guide are built using the PowerShell script `doc\build-doc.ps1`. The HTML and PDF files are built in subdirectory `bin\doc`. By default, they are automatically opened using the default HTML and PDF viewers of the system.

Because the two guides are installed with the rest of TSDuck, this script is automatically executed as part of the script `pkg\nsis\build-installer.ps1`.

The programming reference is built using the PowerShell script `doc\doxy\build-doxygen.ps1`. The set of files is built in subdirectory `bin\doxy\html`. By default, the start page is automatically opened using the default HTML viewer of the system.

When used in an automation system, the two scripts `doc\build-doc.ps1` and `pkg\nsis\build-installer.ps1` can be called with options `-NoOpen -NoPause` to skip the opening of documents using the default viewers and exit without waiting for a user to close the command window.

2.2. Building on UNIX systems (Linux, macOS, BSD)

The user guide and the developer guide are built using the target `docs`. The HTML and PDF files are built in subdirectory `bin/doc`.

```
$ make docs
```

Because the two guides are installed with the rest of TSDuck, they are automatically rebuilt as part of `make install`.

The following targets are also available to build a subset of the guides:

`userguide-html`	Build the user guide HTML version
`userguide-pdf`	Build the user guide PDF version
`userguide`	Build the user guide HTML and PDF versions
`open-userguide-html`	Build the user guide HTML version and opens it with the default HTML viewer
`open-userguide-pdf`	Build the user guide PDF version and opens it with the default PDF viewer
`open-userguide`	Build the user guide HTML and PDF versions and opens them with their default viewers
`buildguide-html`	Build the builder guide HTML version
`buildguide-pdf`	Build the builder guide PDF version
`buildguide`	Build the builder guide HTML and PDF versions
`open-buildguide-html`	Build the builder guide HTML version and opens it with the default HTML viewer
`open-buildguide-pdf`	Build the builder guide PDF version and opens it with the default PDF viewer
`open-buildguide`	Build the builder guide HTML and PDF versions and opens them with their default viewers
`devguide-html`	Build the developer guide HTML version
`devguide-pdf`	Build the developer guide PDF version
`devguide`	Build the developer guide HTML and PDF versions
`open-devguide-html`	Build the developer guide HTML version and opens it with the default HTML viewer
`open-devguide-pdf`	Build the developer guide PDF version and opens it with the default PDF viewer
`open-devguide`	Build the developer guide HTML and PDF versions and opens them with their default viewers
`docs`	Build all guides in HTML and PDF formats
`docs-html`	Build all guides in HTML format
`docs-pdf`	Build all guides in PDF format

The programming reference is built using the target `doxygen`.

```
$ make doxygen
```

The set of files is built in subdirectory `bin/doxy/html`.

Chapter 3. Installing TSDuck

TSDuck can be installed on Windows, Linux, macOS and BSD systems.

3.1. Installing on Windows

On Windows systems, TSDuck can be installed using a binary installer (traditional method) or using the `winget` package manager (modern method).

3.1.1. Using winget

TSDuck is installable on Windows systems using the winget package manager.

`winget` is now the preferred package manager for open source and third-party products on Windows systems. It is documented and supported by Microsoft. It should be pre-installed on all recent Windows 10 and Windows 11 systems.

The TSDuck installation command is simply:

```
PS C:\> winget install tsduck
```

3.1.2. Download an installer

Executable binary installers for the latest TSDuck version are available for 64-bit Windows on Intel systems.

All tools, plugins and development environments are in the same installer. Running the installer provides several options:

- Tools & Plugins
- Documentation
- Python Bindings (optional)
- Java Bindings (optional)
- C++ Development (optional)

Older versions of TSDuck remain available on GitHub.

Nightly builds and pre-releases can be found on the TSDuck Web site.

To automate the installation, the executable binary installer can be run from the command line or a script.

- The option `/S` means "silent". No window is displayed, no user interaction is possible.
- The option `/all=true` means install all options. By default, only the tools, plugins and documentation are installed. In case of upgrade over an existing installation, the default is to upgrade the same options as in the previous installation.

3.2. Installing on macOS

TSDuck is installable on macOS systems using Homebrew, the package manager for open-source projects on macOS.

If you have never used Homebrew on your system, you can install it using the following command (which can also be found on the Homebrew home page):

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Once Homebrew is set up, you can install TSDuck using:

```
$ brew install tsduck
```

All tools, plugins and development environments are installed.

After installation, to upgrade to latest version:

```
$ brew update
$ brew upgrade tsduck
```

When Homebrew upgrades packages, the old versions are not removed. The new versions are just added. After a while, megabytes of outdated packages accumulate on disk. To remove outdated packages:

```
$ brew cleanup
```

To uninstall TSDuck:

```
$ brew uninstall tsduck
```

If you would like to install the latest test version from the source repository (HEAD version) use the following command. Be aware that it takes time since TSDuck is locally recompiled.

```
$ brew install --HEAD tsduck
```

3.3. Installing on Linux

Pre-build packages for the latest TSDuck version are available for the following configurations:

- Fedora (64-bit Intel)
- Ubuntu (64-bit Intel and Arm)
- RedHat, CentOS, Alma Linux, Rocky Linux (64-bit Intel)
- Debian (64-bit Intel)
- Raspbian (32-Bit Arm, Raspberry Pi)

The type of package, `.rpm` or `.deb`, depends on the configuration. The pre-built packages are provided for the latest version of each distro only.

For each distro, two packages exist: the `tsduck` package installs the TSDuck commands, plugins, Java and Python bindings, the `tsduck-devel` or `tsduck-dev` package installs the development environment for C++ programmers.

Older versions of TSDuck remain available on GitHub. Nightly builds and pre-releases for Ubuntu can be found on the TSDuck Web site.

To use older versions of the above distros, rebuilding the packages is easy:

```
$ make installer
```

To install TSDuck on other types of Linux systems for which no package is available:

```
$ make -j10 default docs-html
$ sudo make install
```

More details on how to build TSDuck are available in chapter 1.

If you exclude some dependencies, the exact same set of `make` options must be used during build and installation. For instance, to remove the dependencies on SRT and RIST, use:

```
$ make -j10 default docs-html NOSRT=1 NORIST=1
$ sudo make install NOSRT=1 NORIST=1
```

When building an installer package (`.rpm`, `.deb`) or when directly installing TSDuck, the HTML version of the user guide and developer guide are included.

If the documentation was not built yet, the command `make install` rebuilds the HTML files first. To avoid running Asciidoc under the superuser account, the examples above use `make default docs-html` first. The target `default` builds the binaries and the target `docs-html` builds the HTML documentation. Then, `sudo make install` only copies files into the filesystem, nothing else.

If you prefer not to install Asciidoctor (which pulls the Ruby environment as a dependency), you can install or build an installer package without documentation using the `make` variable `NODOC`.

```
$ make installer NODOC=1
```

```
$ make -j10 NODOC=1
$ sudo make install NODOC=1
```

3.4. Installing on FreeBSD systems

Starting with version 3.43, TSDuck is part of FreeBSD Ports. It can be installed using the `pkg` command:

```
$ sudo pkg install tsduck
```

TSDuck is supposed to be included the April 2026 "Quarterly" branch of FreeBSD Ports. Before that date, use the "Latest" branch to get the `tsduck` package.

3.5. Installing on other BSD systems

There is currently no installer for OpenBSD, NetBSD, DragonFlyBSD. You need to build and install as follow:

```
$ gmake -j10 default docs-html
$ sudo gmake install
```

Note that GNU Make (`gmake`) shall be used instead of the standard BSD `make`.

3.6. Installing with Nix

TSDuck can be installed using the Nix package manager on Linux, macOS, and BSD systems. Nix provides reproducible, declarative package management with automatic dependency resolution.

3.6.1. Installing TSDuck

Install TSDuck into your user profile:

```
$ nix profile add github:tsduck/tsduck?dir=pkg/nix
```

Depending on your Nix configuration, you may have to add the option `--no-write-lock-file`.

```
$ nix profile add --no-write-lock-file github:tsduck/tsduck?dir=pkg/nix
```

3.6.2. Run TSDuck without Install

Or run TSDuck commands without installation:

```
$ nix run github:tsduck/tsduck?dir=pkg/nix
$ nix run github:tsduck/tsduck?dir=pkg/nix#tsp -- --help
```

Launch a Python REPL with TSDuck bindings:

```
$ nix run ./pkg/nix#python
>>> import tsduck
>>> print(tsduck.version())
```

3.6.3. Installing a specific version

A git ref (tag or branch) can be placed between the repository name and the `?dir=pkg/nix` parameter to install an exact version:

```
$ nix profile add github:tsduck/tsduck/v3.40?dir=pkg/nix            # a tagged release
$ nix profile add github:tsduck/tsduck/develop?dir=pkg/nix          # a branch
$ nix profile add github:tsduck/tsduck/v3.40?dir=pkg/nix#tsduck-min # tag + variant
```

When no ref is given, Nix defaults to the repository's default branch (typically `master`).

3.6.4. Installing specific variants

Install the minimal build (no hardware device support):

```
$ nix profile add github:tsduck/tsduck?dir=pkg/nix#tsduck-min
```

Install Python bindings. Each variant has a matching Python package so that the baked `libtsduck` path is consistent:

```
$ nix profile add github:tsduck/tsduck?dir=pkg/nix#python-tsduck       # against tsduck
```

```
$ nix profile add github:tsduck/tsduck?dir=pkg/nix#python-tsduck-min   # against tsduck-min
```

Install Java bindings. As with Python, each tsduck variant has a matching Java package:

```
$ nix profile add github:tsduck/tsduck?dir=pkg/nix#java-tsduck       # against tsduck
$ nix profile add github:tsduck/tsduck?dir=pkg/nix#java-tsduck-min   # against tsduck-min
```

Appendix A: Licenses

A.1. TSDuck license

TSDuck is released under the terms of the license which is commonly referred to as "BSD 2-Clause License" or "Simplified BSD License" or "FreeBSD License". See http://opensource.org/licenses/BSD-2-Clause.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

A.2. Third-party libraries

TSDuck includes a few third-party libraries, either in source form, binary form or both. For more details about the licenses of these third-party libraries, see the file named `OTHERS.txt` in the TSDuck source code repository.

DTAPI: On Linux and Windows, the TSDuck binary distributions contain the DTAPI library in static form. This library is part of the Dektec SDK. This software is available in binary format only and is distributed under the BSD 2-Clause License. *"Copyright (c) 2017 by Dektec Digital Video B.V."*

LIBSRT: On Windows, the TSDuck binary distribution contains the SRT library in static form. There are two versions of the SRT library, one from Haivision, one from Robotweax. TSDuck can be built with any of them.

Haivision LIBSRT: The Haivision SRT library is open-source and distributed under the Mozilla Public License v2.0. *"Copyright (c) 2018 Haivision Systems Inc."*

Robotweax LIBSRT: The Robotweax SRT library is open-source and distributed under the MIT License. *"Copyright (c) 2026 Robotweax GmbH and contributors"*

LIBRIST: On Windows, the TSDuck binary distribution contains the RIST library in static form. This is an open-source library which is distributed under the BSD 2-Clause License. *"Copyright © 2019-2020 SipRadius LLC. All right reserved."*

LibVatek: On Windows and Linux, the TSDuck binary distribution contains the LibVatek library in static form. This is an open-source library which is distributed under the BSD 2-Clause License. *"Copyright © 2022, Richie Chang."*

Small Deflate (sdefl): On Windows, the TSDuck binary distribution contains the Small Deflate library in static form. This is an open-source library which is distributed under two licenses, MIT License and Public Domain License. *"Copyright (c) 2020-2023 Micha Mettke."*

Appendix B: References

Bibliography

- [BSD-2C] BSD 2-Clause License, http://opensource.org/licenses/BSD-2-Clause
- [Haivision-SRT] Haivision original SRT library, https://github.com/Haivision/srt/
- [RIST] RIST library, https://code.videolan.org/rist/librist/
- [Robotweax-SRT] Robotweax alternative SRT library, https://github.com/Robotweax/srt/
- [TSDuck] TSDuck Web site, https://tsduck.io/
- [TSDuck-Issues] TSDuck issues tracker and discussion forum, https://github.com/tsduck/tsduck/issues
- [TSDuck-Source] TSDuck source code repository, https://github.com/tsduck/tsduck/