

BAWT - Build Automation With Tcl

1	INTRODUCTION	3
2	INSTALLATION AND USAGE EXAMPLES	4
2.1	Installation on Windows	4
2.2	Installation on Linux	5
2.3	Installation on Darwin	8
2.4	Use of Batch Scripts	8
3	DIRECTORY AND FILE STRUCTURE	11
3.1	Directory Structure	11
3.1.1	<i>Structure of the input directories</i>	11
3.1.2	<i>Structure of the output directories</i>	12
3.1.3	<i>Directory access</i>	12
3.2	Setup Files	13
3.3	Build Files	22
3.3.1	<i>User supplied build files</i>	25
3.3.2	<i>User configurable build files</i>	26
4	BUILD STAGES	29
4.1	Stage Bootstrap	29
4.2	Stage Setup	30
4.3	Stage Clean	31
4.4	Stage Extract	32
4.5	Stage Configure	32
4.6	Stage Compile	34
4.7	Stage Distribute	34
4.8	Stage Finalize	35
4.9	Stage Test	36
5	BUILD PROCESS	38
5.1	User Perspective	38
5.1.1	<i>Use Case: Cosmetic change of Build file Tcl.bawt</i>	38
5.1.2	<i>Compiler selection on Windows</i>	41
5.1.3	<i>Online updates of libraries</i>	43
5.1.4	<i>Use the generated libraries</i>	43
5.1.5	<i>Change icons of executables</i>	46
5.1.6	<i>Sign executables</i>	47
5.1.7	<i>Parallel builds</i>	47
5.2	Developer Perspective	48
5.2.1	<i>Upgrade a library</i>	48
5.2.2	<i>Add a library</i>	48
5.2.3	<i>Add a Tcl program</i>	50
5.2.4	<i>Manually compile a library</i>	50
5.3	Known issues	52
5.3.1	<i>Build deadlock</i>	52
5.3.2	<i>BawtLogViewer shows incorrect build time</i>	52
5.3.3	<i>Package SWIG</i>	52
5.3.4	<i>Package Trf</i>	52
5.3.5	<i>Package tcllib/crc32</i>	52
5.4	Tips and Tricks	53

5.4.1	<i>Tips for Windows</i>	53
5.4.2	<i>Tips for Linux</i>	53
5.5	Advanced Batch Scripts.....	54
5.5.1	<i>Build Tcl-Pure distributions</i>	54
5.5.2	<i>Build Tcl-BI distributions</i>	56
6	LOGGING	58
6.1	Graphical Log Viewer	58
7	COMMAND LINE OPTIONS	65
7.1	General Options	65
7.2	List Action Options.....	65
7.3	Build Action Options	65
7.4	Build Configuration Options	66
8	SUPPORTED LIBRARIES	69
9	MSYS / MINGW INFORMATION	79
9.1	Introduction.....	79
9.2	Installation of MSYS	80
9.2.1	<i>Download MSYS</i>	80
9.2.2	<i>Download MinGW</i>	80
9.2.3	<i>Configuration</i>	81
9.3	Installation of MSYS2	82
9.3.1	<i>MSYS2/MinGW 64-bit</i>	82
9.3.2	<i>MSYS2/MinGW 32-bit</i>	83
9.4	Further Informations	83
9.4.1	<i>What is MSYS</i>	84
9.4.2	<i>Where to get MSYS</i>	84
9.4.3	<i>How to use MSYS</i>	84
10	RELEASE HISTORY	85

1 Introduction

BAWT is a configurable framework written in **Tcl** for building **C/C++** based software libraries from source code without user interaction. Its main usage is for the **Windows** operating system, where heterogeneous build environments and compilers are needed (or wanted) to build these libraries:

- `configure/make` (via MSYS / MinGW)
- `nmake`
- `CMake`
- Visual Studio Solutions

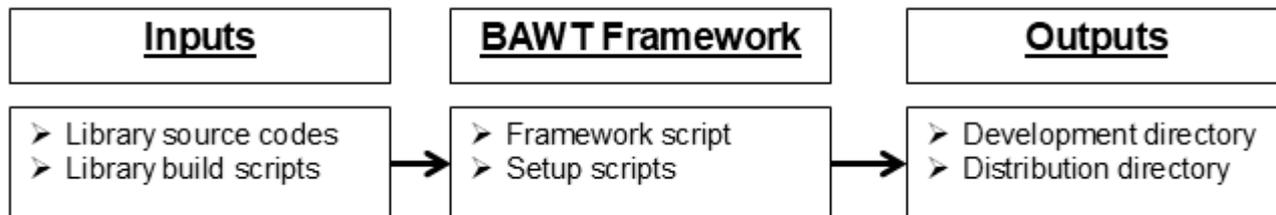
- `gcc` (via MSYS / MinGW)
- Visual Studio

Due to the portable nature of **Tcl** the framework can be used on **Linux** and **Darwin** as well using the `configure/make/gcc` build chain.

The libraries currently supported by **BAWT** are mainly from the **Tcl** and **OpenSceneGraph** domain. For these two domains the framework supports creating installation executables on Windows based on **InnoSetup** and simple shell-based installation programs for Linux and Darwin.

See chapter [8 Supported Libraries](#) for a list of currently supported libraries.

The framework itself is just one plain Tcl file *Bawt.tcl*, which reads a *Setup* file containing all the libraries to be built. Each library must have an accompanying *Build* file, which contains the details on how to extract, configure, compile and distribute the library. The library itself is stored as one or more zipped source code files, **which** may contain different versions of the library. The generated shared or static libraries, programs and header files are finally copied into ready-to-use directory structures for use by developers or for software distribution.



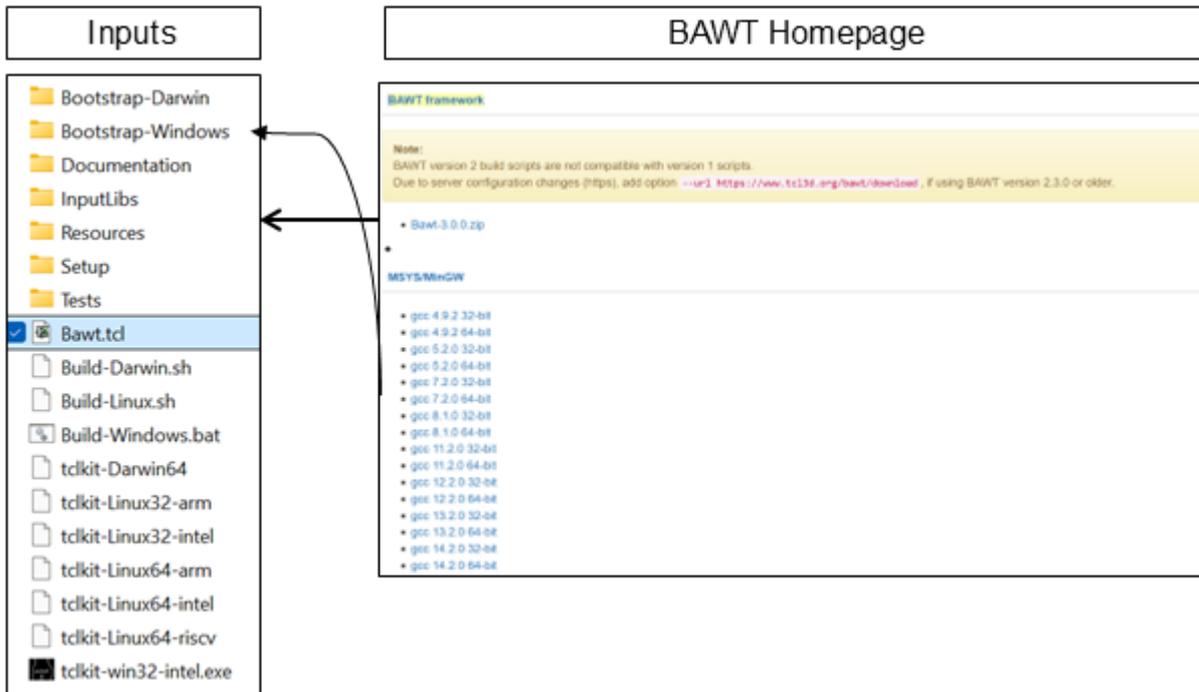
The **BAWT** framework (including *Bootstrap* and *Setup* files) as well as the needed MSYS/MinGW files (if running on Windows) must be downloaded manually. You do not need to have Tcl installed to execute the framework. **BAWT** comes with Tclkits (single-file Tcl interpreter) for Windows, Linux and Darwin. The library *Build* and source files can be downloaded automatically on demand.

The **BAWT** homepage is at <https://www.tcl3d.org/bawt>.

BAWT is copyrighted by Paul Obermeier and distributed under the [3-clause BSD license](#).

2 Installation and Usage Examples

This chapter explains the installation of the **BAWT** framework and gives first simple use cases. **BAWT** related downloads are available at <https://www.tcl3d.org/bawt/download.html>.



2.1 Installation on Windows

Prerequisites:

- None for building libraries supporting MSYS / MinGW.
- Otherwise, Visual Studio (Express, Community or Professional).
 - Visual Studio Versions 2013, 2015, 2017, 2019 and 2022 are currently supported.
 - If Visual Studio is not installed in the standard location, you have to use procedure *SetVcvarsProg* with the absolute path to batch script *vcvarsall.bat*.

Downloads:

- **BAWT** framework *Bawt-3.2.0.zip*
- MSYS / MinGW distribution file(s), ex. *gcc7.2.0_x86_64-w64-mingw32.7z*

Installation:

- Extract **BAWT** framework *Bawt-3.2.0.zip* in a directory of choice, ex. *C:\Bawt*
- Copy MSYS / MinGW distribution file(s) into *C:\Bawt\Bawt-3.2.0\Bootstrap-Windows*
- Open command shell window and go into directory *C:\Bawt\Bawt-3.2.0*

Usage examples:

- Create basic Tcl packages for a 32-bit Intel machine (using only MSYS / MinGW):


```
> Build-Windows.bat intel 32 gcc Setup\Tcl_Basic.bawt update
```
- Create basic Tcl packages for a 64-bit Intel machine (using only MSYS / MinGW):


```
> Build-Windows.bat intel 64 gcc Setup\Tcl_Basic.bawt update
```

- Create extended Tcl packages including InnoSetup installation executable for a 64-bit Intel machine (using Visual Studio 2019 to build Tcl packages supporting Visual Studio like *Mpexpr* and *tkdnd*):
> Build-Windows.bat intel 64 gcc+vs2019 Setup\Tcl_Distribution.bawt update
- Run the test suite of Tcl and Tk:
> Build-Windows.bat intel 64 gcc Setup\Tcl_MinimalDist.bawt test Tcl Tk

2.2 Installation on Linux

Prerequisites:

- Required: C/C++ development package, curl, p7zip
- Optional: Dependent on the libraries. See below for distribution specific examples.

Downloads:

- **BAWT** framework *Bawt-3.2.0.zip*

Installation:

- Extract **BAWT** framework *Bawt-3.2.0.zip* in a directory of choice, ex. *~/Bawt*
- Open shell (Terminal window), go into created directory *~/Bawt Bawt-3.2.0* and execute:
> chmod u+x Build*.sh
> chmod u+x tclkit*

Usage examples:

- Create basic Tcl packages for a 64-bit Intel machine:
> ./Build-Linux.sh intel 64 Setup/Tcl_Basic.bawt update
- Create extended Tcl packages including simple shell-based installation script for a 64-bit ARM machine:
> ./Build-Linux.sh arm 64 Setup/Tcl_Distribution.bawt update
- Create minimal Tcl packages including simple shell based installation script for a 64-bit Risc-V machine:
> ./Build-Linux.sh riscv 64 Setup/Tcl_Basic.bawt update
- Run the test suite of Tcl and Tk:
> ./Build-Linux.sh arm 64 gcc Setup/Tcl_MinimalDist.bawt test Tcl Tk

If compiling Tcl3DFull on slow machines like Raspberry or Risc-V, you should add the following BAWT options:

```
--osgversion 3.4.1 --libjobs tcl3dFull 1 --copt tcl3dFull OptOsg=OFF
```

Distribution specific prerequisites:

See chapter [3.2 Setup Files](#) for a list of available Setup files and the dependencies between Setup files. If you want to build ex. *Tcl_Extended.bawt*, you must not only install the prerequisites of this Setup file, but also the prerequisites of the dependent Setup file *Tcl_Basic.bawt*.

[Debian 12.0 Bookworm \(gcc 12.2.0\)](#)

- Install default Debian 12.0 desktop distribution (ex. *debian-12.0.0-amd64-DVD-1.iso*)
- Use ex. `Synaptic` to install further packages:

Setup file	Debian package	Needed by library
<i>All</i>	build-essential	All C/C++ based libraries.
	curl	BAWT framework.
	p7zip	
	zip	
<i>Tcl_Basic.bawt</i>	libx11-dev	Tk
	libcups2-dev	Tk (printing)
	libcairo2-dev	tkpath
	libglx-dev	Canvas3d
	libglul-mesa-dev	
	libasound2-dev	Snack
<i>Tcl_Extended.bawt</i>	libxrandr-dev	tcl3dBasic
	libpython3.11-dev	tclpy
	python3-numpy	
	libxcursor-dev	tkdnd
	libudev-dev	tcluvc
	libjpeg*-turbo-dev	tcluvc libgd
<i>Tcl3D.bawt</i>	libxi-dev	glfw
	libxinerama-dev	
<i>OSG_Extended.bawt</i>	freeglut3-dev	Cal3D

[*Raspberry Pi OS \(gcc 10.2.1\)*](#)

- Install default Raspberry Pi OS using the Raspberry Pi Imager.
- Packages `build-essential`, `curl` and `p7zip` are part of the OS installation.
- As Raspberry Pi OS is based on Debian, install additional packages as for Debian.

[*Ubuntu 23.04 \(gcc 12.2.0\)*](#)

- Install default Ubuntu 23.04 desktop distribution (ex. *ubuntu-23.04-desktop-amd64.iso*)
- Use ex. `Synaptic` to install further packages:

Setup file	Ubuntu package	Needed by library
<i>All</i>	build-essential	All C/C++ based libraries.
	curl	BAWT framework.
	p7zip	
	zip	
<i>Tcl_Basic.bawt</i>	libx11-dev	Tk
	libcups2-dev	Tk (printing)
	libcairo2-dev	tkpath
	libglx-dev	Canvas3d
	libglul-mesa-dev	
	libasound2-dev	Snack
<i>Tcl_Extended.bawt</i>	libxrandr-dev	tcl3dBasic
	libpython3.11-dev	tclpy
	python3-numpy	
	libxcursor-dev	tkdnd
	libudev-dev	tcluvc

	libjpeg*-turbo-dev	tcluvc libgd
<i>Tcl3D.bawt</i>	libxi-dev libxinerama-dev	glfw
<i>OSG_Extended.bawt</i>	freeglut3-dev	Cal3D

SUSE 15.5 (gcc 7.5.0)

- Install default SUSE 15.5 desktop distribution (ex. *openSUSE-Leap-15.5-DVD-x86_64-Build491.1-Media.iso*)
- Use `Yast` to install further packages:

Setup file	SUSE schema	Needed by library
<i>All</i>	General development C++ development	All C/C++ based libraries.
Setup file	SUSE package	Needed by library
<i>Tcl_Basic.bawt</i>	libx11-devel	Tk
	cups-devel	Tk (printing)
	cairo-devel	tkpath
	alsa-devel	Snack
<i>Tcl_Extended.bawt</i>	glu-devel	tcl3dBasic
	libxrandr-devel	
	python3-devel	tclpy
	python3-numpy	
	libxcursor-devel	tkdnd
	systemd-devel	tcluvc
	libjpeg*-devel	tcluvc libgd
<i>Tcl3D.bawt</i>	libxi-devel libxinerama-devel	glfw
<i>OSG_Extended.bawt</i>	freeglut-devel	Cal3D

Fedora 38.1 (gcc 13.1.1)

- Install default Fedora 38.1 workstation distribution (ex. *Fedora-Workstation-Live-x86_64-38-1.6.iso*)
- Use `dnf` to install further packages:

Setup file	dnf groupinstall	Needed by library
<i>All</i>	Development Tools Development Libraries X Software Development	All C/C++ based libraries.
Setup file	dnf install	Needed by library
<i>Tcl_Basic.bawt</i>	cups-devel	Tk (printing)
	gcc-g++	photoresize
	cairo-devel	tkpath
	alsa-lib-devel	Snack
<i>Tcl_Extended.bawt</i>	mesa-libGLU-devel	tcl3dBasic
	jbigkit-devel	openjpeg
	liblerc-devel	
	libudev-devel	tcluvc

<i>Tcl3D.bawt</i>	libXinerama-devel	glfw
<i>OSG_Extended.bawt</i>	freeglut-devel	Cal3D

2.3 Installation on Darwin

Prerequisites:

- XCode
- curl (should be available by default on Darwin)

Downloads:

- **BAWT** framework *Bawt-3.2.0.zip*

Installation:

- Extract **BAWT** framework *Bawt-3.2.0.zip* in a directory of choice, ex. *~/Bawt*
- Open shell (Terminal window), go into created directory *~/Bawt Bawt-3.2.0* and execute:


```
> chmod u+x Build*.sh
> chmod u+x tclkit*
```

Usage examples:

Note, that Darwin does not support 32-bit programs.

- Create basic Tcl packages as universal binaries:


```
> ./Build-Darwin.sh universal Setup/Tcl_Basic.bawt update
```
- Create extended Tcl packages including simple shell-based installation script as native binaries:


```
> ./Build-Darwin.sh native Setup/Tcl_Distribution.bawt update
```
- Run the test suite of Tcl and Tk:


```
> ./Build-Darwin.sh native Setup/Tcl_MinimalDist.bawt test Tcl Tk
```

2.4 Use of Batch Scripts

As the **BAWT** framework is generic and has lots of command line options (see chapter [7 Command Line Options](#)), a batch or shell script for each supported platform is included in the distribution for ease of usage in the most common use cases:

- *Build-Windows.bat*
- *Build-Linux.sh*
- *Build-Darwin.sh*

These batch scripts have been used in the examples of the previous chapters and may serve as starting point for your own batch scripts suited exactly to your needs.

Batch script *Build-Windows.bat*

```
@echo off
setlocal

rem Default values for some often used options.
set OUTROOTDIR=./BawtBuild
set NUMJOBS=%NUMBER_OF_PROCESSORS%
```

```

rem First 5 parameters are mandatory.
if "%1" == "" goto ERROR
if "%2" == "" goto ERROR
if "%3" == "" goto ERROR
if "%4" == "" goto ERROR
if "%5" == "" goto ERROR

set PROCESSOR=%1
set BITS=%2
set COMPILER=%3
set SETUPFILE=%4
set ACTION=%5
shift
shift
shift
shift
shift

rem If no target is given, use target "all".
if "%1"==" " goto BUILDALL

rem Loop through the rest of the parameter list for targets.
set TARGETS=
:PARAMLOOP
rem There is a trailing space in the next line. It's there for formatting.
set TARGETS=%TARGETS%%1
shift
if not "%1"==" " goto PARAMLOOP
goto BUILD

:BUILDALL
if "%ACTION%"=="clean" goto WARNING
if "%ACTION%"=="complete" goto WARNING

set TARGETS=all

:BUILD

if "%BITS%"=="32" set ARCH=x86
if "%BITS%"=="64" set ARCH=x64
if "X%ARCH%"=="X" goto ERROR

if "X%TCLKIT%"=="X" set TCLKIT=tclkit-win32-intel.exe

set ACTION=--%ACTION%
set BAWTOPTS=--rootdir %OUTROOTDIR% ^
             --architecture %ARCH% ^
             --compiler %COMPILER% ^
             --numjobs %NUMJOBS% ^
             --logviewer

rem Build all libraries as listed in Setup file.
CALL %TCLKIT% Bawt.tcl %BAWTOPTS% %ACTION% %SETUPFILE% %TARGETS%

goto EOF

:WARNING
echo Warning: This may clean or rebuild everything.
echo Use "clean all" or "complete all" to allow this operation.

:ERROR
echo.
echo Usage: %0 Processor Bits Compiler SetupFile Action [Target1] [TargetN]
echo Processor : intel
echo Bits : 32 64

```

```
echo Compiler      : gcc vs2013 vs2015 vs2017 vs2019 vs2022
echo               gcc+vs20XX vs20XX+gcc
echo Actions      : list clean extract configure compile distribute
echo               finalize complete update simulate touch test
echo Default target : all
echo.
echo Output directory: %OUTROOTDIR%
echo.
echo Specify variable TCLKIT on the command line to use a separate bootstrap program.
echo Example:
echo set TCLKIT=tclsh ^&^& Build-Windows.bat intel 64 gcc Setup\Tcl_Basic.bawt update
echo.
:EOF
```

See also chapter [5.5 Advanced Batch Scripts](#) for examples of more complex batch scripts.

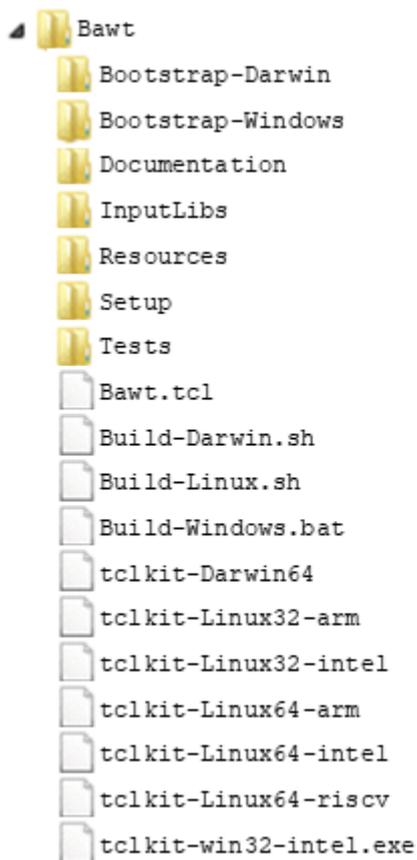
3 Directory and File Structure

This chapter explains the directory structure of the input and output files as well as the contents and structure of the *Setup* and *Build* files.

3.1 Directory Structure

3.1.1 Structure of the input directories

If **BAWT** was downloaded and installed according to the instructions in chapter 2 *Installation and Usage*, the following directory structure should exist.



The *Bootstrap* directories contain zipped versions of the `7-zip` program for Windows and Darwin and zipped versions of the `zip` program for Windows and Linux.

In directory *Bootstrap-Windows* there should be at least one version of the MSYS/MinGW distributions, which you must have downloaded manually.

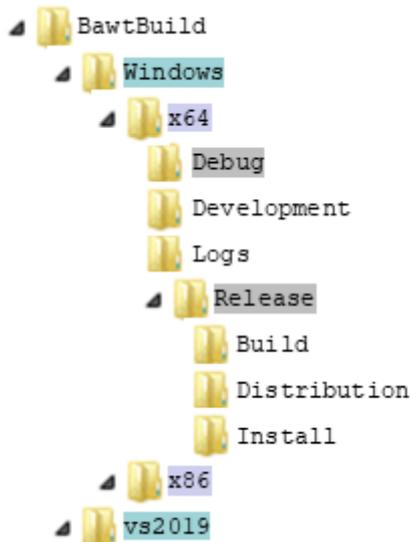
Directory *InputLibs* contains the zipped source code versions of the libraries and the associated *Build* files, see chapter 0

Build Files for a detailed description of *Build* files. Note, that this directory is empty after a fresh installation of **BAWT**, because the corresponding files are downloaded on demand at the first start of a **BAWT** build by default. See chapter 5 *Build Process* on how to avoid automatic downloads and updates. The *Setup* files (see chapter 3.2 *Setup Files*) supplied with **BAWT** are located in directory *Setup*.

Directory *Tests* contains several simple test scripts for checking correct compilation and installation of Tcl related packages.

For each supported platform there is also a Tclkit executable supplied, which is needed to run the **BAWT** framework, if no Tcl interpreter is available on your machine (Bootstrapping). A Tclkit is a single-file Tcl interpreter executable.

3.1.2 Structure of the output directories



The root directory of the output files of a **BAWT** build (*BawtBuild* in the above example) can be specified with command line option `--rootdir`. In a *Build* script this directory can be queried with Tcl procedure *GetOutputRootDir*.

Beneath the root build directory there can be several directories named according to the build environment used: *Windows*, *Linux*, *Darwin* for builds with *gcc* or *vs2013*, *vs2015*, *vs2017*, *vs2019* or *vs2022*, if a Visual Studio version was used for building.

Beneath these environment specific directories two directory names can appear, depending on the build architecture: *x86* for 32-bit or *x64* for 64-bit builds.

In these architecture specific directories 3 to 4 subdirectories are contained.

The *Logs* directory contains the overall build log file *_BawtBuild.log* as well as the library specific build log files. See chapter 0

Logging for an in-depth explanation of **BAWT** logging functionality.

The *Development* directory contains all the files needed for a developer using the built libraries.

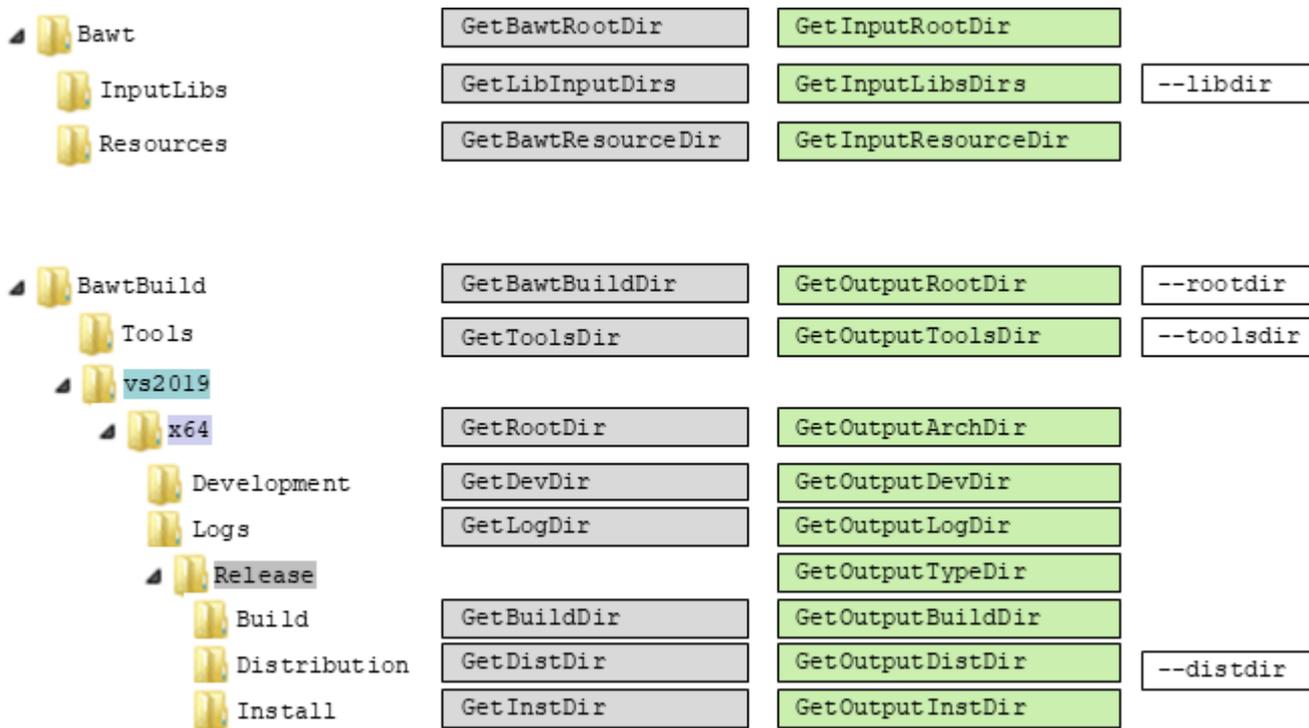
Depending on the specified build types, directories called *Release* and *Debug* will be created. These directories contain the *Build* and *Install* subdirectories, where the actual sources are extracted and built as well as a *Distribution* subdirectory, which will contain all files needed for a software distribution of the compiled libraries.

The *Distribution* and *Development* directories contain mostly identical content. The *Development* directory typically contains additional library include files and import files (**.lib*). It is the task of the library specific *Build* file to copy the needed files into the *Distribution* and *Development* directories.

3.1.3 Directory access

The next figure shows the input and output directory hierarchy together with the procedures which can be used to get the path to the corresponding directory. The first procedure column (grey boxes) shows the names used in BAWT versions prior to 1.0, the second column (green boxes) shows the names as used by BAWT 1.0 and newer.

The last column shows the available command line options to change the location of a specific input or output directory.



The library search paths, which can be obtained with procedure `GetInputLibsDirs` are set at BAWT start-up to the following values:

- `file join [GetInputRootDir] "InputLibs"`
- `file join [pwd] "InputLibs"`

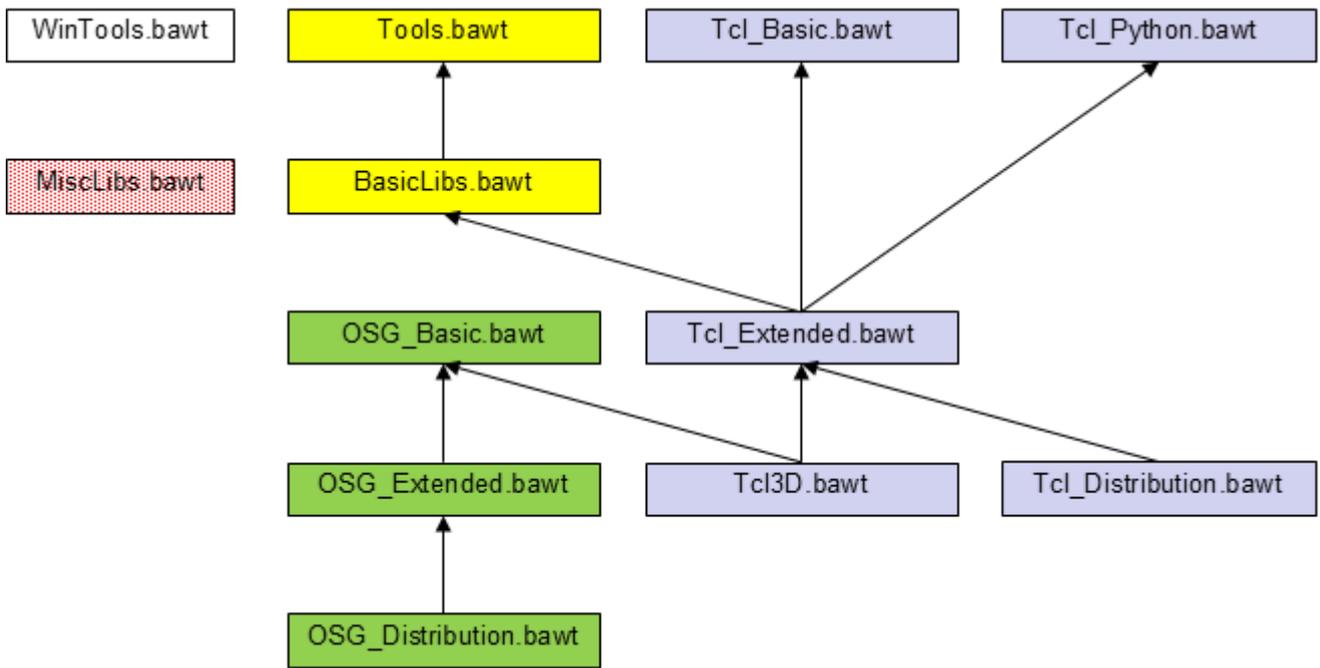
This list can be extended by using command line option `--libdir`.

If command line option `--nosubdirs` is specified, procedures `GetOutputArchDir` and `GetOutputRootDir` return the same directory path.

See chapter 4 *Build Stages* for an in-depth tour through the directory structure of **BAWT** in conjunction with the different build stages.

3.2 Setup Files

The following figure shows all available *Setup* files and their dependencies.



For the **Tcl** ecosystem the following *Setup* files are currently supported.

<i>Tcl_MinimalDist.bawt</i>	Builds Tcl, Tk and creates an InnoSetup based setup file on Windows or an installation shell script on Unix.
<i>Tcl_Basic.bawt</i>	Builds Tcl, Tk, Tckit and Tcl/Tk packages, which do not depend on 3rd party libraries. On Windows all libraries can be compiled with MSYS/MinGW.
<i>Tcl_Python.bawt</i>	Extracts the binary Python distribution on Windows and builds the tclpy package.
<i>Tcl_Extended.bawt</i>	Builds all libraries of <i>Tcl_Basic.bawt</i> , <i>Tcl_Python.bawt</i> and Tcl/Tk packages which depend on 3rd party libraries, like SWIG, CMake, libressl or image libraries. On Windows all libraries can be compiled with MSYS/MinGW.
<i>Tcl3D.bawt</i>	Builds all libraries of <i>Tcl_Extended.bawt</i> and the extended version of Tcl3D, which depends on 3rd party libraries like OpenSceneGraph, SDL, FTGL.
<i>Tcl_Distribution.bawt</i>	Builds all libraries of <i>Tcl_Extended.bawt</i> and creates an InnoSetup based setup file on Windows or an installation shell script on Unix.

For the **OpenSceneGraph** ecosystem the following *Setup* files are currently supported.

<i>OSG_Basic.bawt</i>	Builds OpenSceneGraph with basic plugin libraries as needed by Tcl3D. On Windows all libraries can be compiled with MSYS/MinGW.
<i>OSG_Extended.bawt</i>	Builds all libraries of <i>OSG_Basic.bawt</i> and builds OpenSceneGraph with extended plugin libraries, as well as libraries depending on OpenSceneGraph like osgEarth.
<i>OSG_Distribution.bawt</i>	Builds all libraries of <i>OSG_Extended.bawt</i> and creates an InnoSetup based setup file on Windows or an installation shell script on Unix.

Both the **OpenSceneGraph** ecosystem as well as the extended **Tcl** versions need special tools for building or basic libraries they depend upon.

<i>Tools.bawt</i>	Builds tools needed for building of libraries, like CMake or SWIG.
<i>BasicLibs.bawt</i>	Builds basic libraries needed by other libraries like several image libraries, zlib, freetype, ffmpeg and libressl.

There are two other *Setup* files not directly related to one of the above-mentioned ecosystems.

<i>WinTools.bawt</i>	Convenience tools for Windows supplied as precompiled binaries like Vim or Doxygen.
<i>MiscLibs.bawt</i>	Builds miscellaneous libraries not directly related to Tcl or OpenSceneGraph like mathematical, geographical or XML libraries.
Note, that the libraries listed in this file are not updated or supported since BAWT version 3.	

See the tables at the end of this chapter for the detailed content of the *Setup* files.

Setup files are standard Tcl script files. They must have one or more calls to the **BAWT** *Setup* procedure for each library being built. Optionally one or more calls to the **BAWT** *Include* procedure can be specified to add dependent libraries.

The *Setup* procedure has the following signature:

```
proc Setup { libName zipFile buildFile args }
```

The following 3 mandatory parameters must be specified:

- *libName*: Library name.
- *zipFile*: Zipped library source file or library source directory.
- *buildFile*: File containing build script for the library (see next chapter).

The following optional build parameters are currently supported:

<i>Release</i>	Build the Release version of the library. This is the default.
<i>Debug</i>	Build the Debug version of the library. Note, that not all libraries may support Debug mode.
<i>NoTcl8 NoTcl9</i>	Do not build the library with Tcl 8 resp. Tcl 9.
<i>NoTk8 NoTk9</i>	Do not build the library with Tk 8 resp. Tk 9.
<i>NoGccX</i>	Do not build the library with gcc major version X.
<i>NoUniversal</i>	Do not build the library as MacOS universal binary.
<i>NoWindows</i>	Do not build the library on Windows.
<i>NoLinux</i>	Do not build the library on Linux.
<i>NoDarwin</i>	Do not build the library on Darwin.
<i>NoWindows-arm</i>	Do not build the library on ARM based Windows.
<i>NoWindows-intel</i>	Do not build the library on Intel based Windows.
<i>NoLinux-arm</i>	Do not build the library on ARM based Linux.
<i>NoLinux-intel</i>	Do not build the library on Intel based Linux.
<i>NoLinux-riscv</i>	Do not build the library on RISC-V based Linux.
<i>NoDarwin-arm</i>	Do not build the library on ARM based Darwin.
<i>NoDarwin-intel</i>	Do not build the library on Intel based Darwin.
<i>WinCompiler=winCompiler</i>	Specify the Windows compiler to use. Valid Windows compiler names are: <i>gcc</i> , <i>vs</i> . Note, that the <i>Build</i> file must have support for both Visual Studio and MSYS/MinGW instructions.
<i>ForceVS (Deprecated)</i>	Force using Visual Studio instead of using MSYS/MinGW. Note, that the <i>Build</i> file must have support for both Visual Studio and MSYS/MinGW instructions.
<i>Version=X.Y.Z</i>	Specify or override a version string for the library. Use this option, if building a library from a directory (ex. your repository

	workspace), which does not have a version number included in the directory name.
<code>MaxParallel=Platform:NumJobs</code>	Specify the number of parallel build jobs for a specific platform. Some build systems do not work correctly with lots of parallel builds. Valid platform names are: Windows, Linux, Darwin. The platform name may be optionally appended by the compiler type <code>vs</code> or <code>gcc</code> . Example: <code>MaxParallel=Windows-gcc:2</code>
<code>NoParallel=Platforms</code> <i>(Deprecated)</i>	Specify platforms as comma separated list for which parallel builds should be disabled. Valid platform names are: Windows, Linux, Darwin.
<code>All other strings</code>	Strings not matching any of the above patterns are interpreted as a user configuration string. User configuration strings are either appended to the <code>CMake</code> or <code>configure</code> commands of the library or can be evaluated by the Build script. See chapter 3.3.2 User configurable build files for a description of user configuration strings.

The next tables list the contents of the currently available *Setup* files.

Setup file Tools.bawt			
# Builds tools needed for building of libraries, like CMake or SWIG.			
# Setup	LibName	ZipFile	BuildFile BuildOptions
if {	[UseVisualStudio "primary"]	&& [GetVisualStudioVersion] <= 2010	{
Setup	CMake	CMake-3.21.4.7z	CMake.bawt
}	elseif {	[IsDarwin]	{
# Use newest	CMake	only on new Darwin systems.	
Setup	CMake	CMake-3.31.7.7z	CMake.bawt
}	else {		
# Use this	CMake	version, so that Visual 2013 still works.	
Setup	CMake	CMake-3.25.2.7z	CMake.bawt
}			
Setup	pandoc	pandoc-3.5.7z	pandoc.bawt NoLinux-riscv
Setup	pkgconfig	pkgconfig-0.29.2.7z	pkgconfig.bawt
Setup	SWIG	SWIG-4.3.1.7z	SWIG.bawt
Setup	yasm	yasm-1.3.0.7z	yasm.bawt
Setup	nasm	nasm-2.16.3.7z	nasm.bawt

Setup file BasicLibs.bawt			
# Builds basic libraries needed by several other libraries.			
Include "Tools.bawt"			
# All of the following libraries can be compiled on Linux or Darwin, # but it is better to use the system provided libraries.			
# Setup	LibName	ZipFile	BuildFile BuildOptions
# Basic library needed by most other libraries.			
Setup	ZLib	ZLib-1.3.1.7z	ZLib.bawt NoLinux NoDarwin
Setup	xz	xz-5.4.1.7z	xz.bawt NoLinux NoDarwin
# Basic Image libraries.			
Setup	giflib	giflib-5.2.1.7z	giflib.bawt NoLinux
Setup	libwebp	libwebp-1.2.4.7z	libwebp.bawt NoLinux
Setup	JPEG	JPEG-9.f.7z	JPEG.bawt NoLinux NoDarwin
Setup	openjpeg	openjpeg-2.5.3.7z	openjpeg.bawt
Setup	PNG	PNG-1.6.48.7z	PNG.bawt NoLinux MaxParallel=Windows-gcc:1
Setup	TIFF	TIFF-4.7.0.7z	TIFF.bawt NoLinux NoDarwin
if {	[IsGccCompilerNewer "4.9.0"]	{	{
Setup	ffmpeg	ffmpeg-7.1.2.7z	ffmpeg.bawt NoLinux32-arm
}	else {		

```

    Setup ffmpeg ffmpeg-4.4.4.7z ffmpeg.bawt NoLinux32-arm
}

Setup Freetype Freetype-2.13.3.7z Freetype.bawt NoLinux NoDarwin
if { [GetTlsVersion] == 1 } {
    Setup libressl libressl-2.9.2.7z libressl.bawt
} else {
    Setup openssl openssl-3.5.1.7z openssl.bawt
}

if { [UseVisualStudio "primary"] && [GetVisualStudioVersion] <= 2008 } {
    # Visual Studio 2008
    Setup SDL SDL-2.0.4.7z SDL.bawt
} elseif { [UseVisualStudio "primary"] && [GetVisualStudioVersion] == 2010 } {
    # Visual Studio 2010
    Setup SDL SDL-2.0.8.7z SDL.bawt
} else {
    Setup SDL SDL-2.26.2.7z SDL.bawt
}
}

```

Setup file Tcl_MinimalDist.bawt

```

# Builds just Tcl and Tk and creates a distribution setup file.

# Setup LibName ZipFile BuildFile BuildOptions

# Tcl and Tk.
Setup Tcl Tcl-[GetTclVersion].7z Tcl.bawt
Setup Tk Tk-[GetTkVersion].7z Tk.bawt

# Tcl/Tk distribution as InnoSetup installer.
Setup InnoSetup InnoSetup-6.2.2.7z InnoSetup.bawt
Setup SetupTcl SetupTcl.7z SetupTcl.bawt

```

Setup file Tcl_Basic.bawt

```

# Builds Tcl, Tk, Starkit and Tcl/Tk packages, which do not depend on 3rd party libraries.
# On Windows all libraries can be compiled with MSys/MinGW.

# Setup LibName ZipFile BuildFile BuildOptions

# Tcl/Tk, stubs and manual.
Setup Tcl Tcl-[GetTclVersion].7z Tcl.bawt
Setup TclStubs Tcl-[GetTclVersion].7z TclStubs.bawt
Setup Tk Tk-[GetTkVersion].7z Tk.bawt
Setup TkStubs Tk-[GetTkVersion].7z TkStubs.bawt
Setup TclTkManual TclTkManual.7z TclTkManual.bawt

# Compiled Tcl packages.
Setup argparse argparse-0.61.7z argparse.bawt
Setup critcl critcl-3.3.7z critcl.bawt
Setup expect expect-5.45.4.1.7z expect.bawt NoLinux32-arm
Setup DiffUtil DiffUtil-0.4.3.7z DiffUtil.bawt
Setup memchan memchan-2.3.1.7z memchan.bawt NoTcl9
Setup Mpexpr Mpexpr-1.2.1.7z Mpexpr.bawt
Setup nacl nacl-1.1.1.7z nacl.bawt
Setup nsf nsf-2.4.0.7z nsf.bawt NoTcl9
Setup oratcl oratcl-4.6.1.7z oratcl.bawt
Setup parse_args parse_args-0.5.1.7z parse_args.bawt
Setup poMemory poMemory-1.0.0.7z poMemory.bawt
Setup rl_json rl_json-0.16.0.7z rl_json.bawt
Setup tbcload tbcload-2.0a0.7z tbcload.bawt
Setup tclcompiler tclcompiler-2.0a0.7z tclcompiler.bawt
Setup tcldebugger tcldebugger-2.0.1.7z tcldebugger.bawt
Setup tclcsv tclcsv-2.4.3.7z tclcsv.bawt
Setup tclparser tclparser-1.9.7z tclparser.bawt
Setup tclvfs tclvfs-1.5.0.7z tclvfs.bawt
Setup tclx tclx-9.0.0.7z tclx.bawt
Setup tdom tdom-0.9.6.7z tdom.bawt
Setup trofs trofs-0.4.9.7z trofs.bawt NoTcl9
Setup tserialport tserialport-1.1.1.7z tserialport.bawt MaxParallel=Windows-gcc:1
Setup udp udp-1.0.12.7z udp.bawt NoTcl9
Setup vectcl vectcl-0.2.1.7z vectcl.bawt NoTcl9

# Compiled Tk packages.
Setup Canvas3d Canvas3d-1.2.3.7z Canvas3d.bawt NoTcl9
Setup Img Img-[GetImgVersion].7z Img.bawt

```

Setup imgtools	imgtools-0.3.1.7z	imgtools.bawt	
Setup itk	itk-4.2.5.7z	itk.bawt	
Setup iwidgets	iwidgets-4.1.2.7z	iwidgets.bawt	
Setup photoresize	photoresize-0.2.1.7z	photoresize.bawt	
Setup poImg	poImg-3.0.1.7z	poImg.bawt	
Setup rtext	rtext-0.1.1.7z	rtext.bawt	NoTcl8
Setup Snack	Snack-2.2.12.7z	Snack.bawt	
Setup Tix	Tix-8.4.4.7z	Tix.bawt	NoDarwin NoTk9
Setup Tkhtml	Tkhtml-3.0.2.7z	Tkhtml.bawt	
Setup tkpath	tkpath-0.4.2.7z	tkpath.bawt	
Setup tko	tko-0.4.7z	tko.bawt	NoDarwin
Setup tksvg	tksvg-0.14.7z	tksvg.bawt	
Setup Tktable	Tktable-2.12.7z	Tktable.bawt	
Setup treectrl	treectrl-2.5.2.7z	treectrl.bawt	Shellicon=ON
# Compiled Tcl and Tk packages. Windows only.			
Setup iocp	iocp-2.0.2.7z	iocp.bawt	
if { [GetMajor [GetTclVersion]] < 9 } {			
Setup rbc	rbc-0.2.0.7z	rbc.bawt	NoTk9
} else {			
Setup rbc	rbcTk9-0.2.0.7z	rbc.bawt	NoTk8
}			
Setup shellicon	shellicon-0.2.0.7z	shellicon.bawt	
if { [Is32Bit] && ! [IsGccCompilerNewer "8.1.0"] } {			
Setup twapi	twapi-5.1.1.7z	twapi.bawt	
} else {			
Setup twapi	twapi-5.2.0.7z	twapi.bawt	
}			
Setup winhelp	winhelp-1.1.1.7z	winhelp.bawt	
# Compiled Tcl packages. Darwin only.			
Setup Tcladdressbook	Tcladdressbook-1.2.4.7z	Tcladdressbook.bawt	NoTcl9
Setup Tclapplescript	Tclapplescript-2.2.7z	Tclapplescript.bawt	NoTcl9
Setup tclAE	tclAE-2.0.7.7z	tclAE.bawt	NoTcl9
# Pure Tcl/Tk packages.			
Setup apave	apave-4.4.10.7z	apave.bawt	
Setup awthemes	awthemes-10.4.0.7z	awthemes.bawt	
Setup BWidget	BWidget-1.10.1.7z	BWidget.bawt	
Setup cawt	cawt-3.1.1.7z	cawt.bawt	
Setup materialicons	materialicons-0.2.7z	materialicons.bawt	
Setup mentry	mentry-4.5.7z	mentry.bawt	
Setup mqtt	mqtt-4.0.7z	mqtt.bawt	
Setup ooxml	ooxml-1.10.7z	ooxml.bawt	
Setup pdf4tcl	pdf4tcl-0.9.4.7z	pdf4tcl.bawt	
Setup pgintcl	pgintcl-3.5.2.7z	pgintcl.bawt	
Setup poLibs	poApps-3.2.0.7z	poLibs.bawt	
Setup publisher	publisher-2.0.7z	publisher.bawt	
Setup puppyicons	puppyicons-0.1.7z	puppyicons.bawt	
Setup ruff	ruff-2.6.0.7z	ruff.bawt	
Setup scrollutil	scrollutil-2.6.7z	scrollutil.bawt	
Setup thtmlview	thtmlview-2.0.0.7z	thtmlview.bawt	
Setup tablelist	tablelist-7.7.7z	tablelist.bawt	
Setup tcl9migrate	tcl9migrate-1.0.7z	tcl9migrate.bawt	
Setup tclargp	tclargp-0.2.7z	tclargp.bawt	
Setup tclfpdf	tclfpdf-1.7.2.7z	tclfpdf.bawt	
Setup tcllib	tcllib-2.1.7z	tcllib.bawt	
Setup tclws	tclws-3.5.0.7z	tclws.bawt	
Setup tkcon	tkcon-2.8.7z	tkcon.bawt	
Setup tklib	tklib-0.9.7z	tklib.bawt	
Setup tsw	tsw-1.2.7z	tsw.bawt	
Setup ukaz	ukaz-2.1.7z	ukaz.bawt	
Setup wcb	wcb-4.2.7z	wcb.bawt	
Setup windetect	windetect-2.0.1.7z	windetect.bawt	
Setup tkwintrack	tkwintrack-2.1.1.7z	tkwintrack.bawt	
# Tclkits.			
Setup vlerq	vlerq-4.1.7z	vlerq.bawt	
Setup Tclkit	Tclkit.7z	Tclkit.bawt	
# Tcl programs wrapped as starpacks.			
Setup cawtapp	cawt-3.1.1.7z	cawtapp.bawt	
Setup jigsaw	jigsaw-2.0.7z	jigsaw.bawt	
Setup gorilla	gorilla-1.6.1.7z	gorilla.bawt	
Setup tclssg	tclssg-3.0.1.7z	tclssg.bawt	
Setup tkchat	tkchat-1.482.7z	tkchat.bawt	
Setup tksqlite	tksqlite-0.5.14.7z	tksqlite.bawt	

```

Setup file Tcl_Python.bawt
# Builds binary Python distribution for Windows and tclpy package.

Include "Tcl_Basic.bawt"

# Setup LibName      ZipFile                BuildFile          BuildOptions
Setup Python        Python-3.7.7-[GetBits].7z  Python.bawt       Version=3.7.7
Setup tclpy         tclpy-0.4.1.7z            tclpy.bawt
    
```

```

Setup file Tcl_Extended.bawt
# Builds Tcl/Tk packages which depend on 3rd party libraries,
# like SWIG, CMake, openssl or image libraries.

Include "Tools.bawt"
Include "BasicLibs.bawt"
Include "Tcl_Basic.bawt"
Include "Tcl_Python.bawt"

# Setup LibName      ZipFile                BuildFile          BuildOptions
#Setup mawt          mawt-0.5.0            mawt.bawt         NoLinux32-arm
Version=0.5.0
Setup mawt           mawt-0.5.0.7z         mawt.bawt         NoLinux32-arm
Setup FTGL           FTGL-2.1.3.7z         FTGL.bawt         NoDarwin
Setup tcl3dExtended  tcl3d-1.0.1.7z        tcl3dExtended.bawt NoDarwin
Setup OglInfo        tcl3d-1.0.1.7z        OglInfo.bawt      NoDarwin

Setup tkdnd          tkdnd-2.9.5.7z        tkdnd.bawt
Setup tkribbon       tkribbon-1.2.7z       tkribbon.bawt

if { [GetTlsVersion] == 1 } {
  Setup tcltls       tcltls-1.7.23.7z      tcltls.bawt
} else {
  Setup tcltls       tcltls-2.0b1.7z      tcltls2.bawt
}
Setup Trf            Trf-2.1.4.7z          Trf.bawt           NoDarwin NoTcl9

Setup imgjp2         imgjp2-0.1.1.7z       imgjp2.bawt
Setup tzint          tzint-1.1.1.7z        tzint.bawt         NoUniversal

Setup libgd          libgd-2.3.2.7z        libgd.bawt
Setup tclgd          tclgd-1.4.1.7z        tclgd.bawt         NoUniversal

Setup tcluvc         tcluvc-0.1.7z         tcluvc.bawt

Setup cfitsio        cfitsio-4.1.0.7z      cfitsio.bawt
Setup fitsTcl        fitsTcl-2.5.1.7z      fitsTcl.bawt
Setup pawt           pawt-1.2.0.7z         pawt.bawt

Setup libffi         libffi-3.4.8.7z       libffi.bawt        NoUniversal
Setup cffi            cffi-2.0.3.7z         cffi.bawt          NoUniversal
Setup Ffidl          Ffidl-0.9.1.7z        Ffidl.bawt         NoUniversal

# MuPDF (and therefore dependent libraries tclMuPdf and MuPDFWidget)
# are not available with VisualStudio < 2019.
if { ( [UseVisualStudio "primary"] && [GetVisualStudioVersion] < 2019 ) || \
      ![IsGccCompilerNewer "4.9.2"] } {
  Setup mupdf         mupdf-1.18.2.7z       mupdf.bawt
  Setup tclMuPdf     tclMuPdf-2.1.1.7z    tclMuPdf.bawt
} else {
  Setup mupdf         mupdf-1.26.0.7z       mupdf.bawt
  Setup tclMuPdf     tclMuPdf-2.5.1.7z    tclMuPdf.bawt
}
Setup MuPDFWidget    MuPDFWidget-2.4.7z    MuPDFWidget.bawt

Setup hdc            hdc-0.2.0.1.7z         hdc.bawt
Setup gdi            gdi-0.9.9.15.7z      gdi.bawt
Setup printer        printer-0.9.6.16.7z   printer.bawt

# Tcl programs wrapped as starpacks.
Setup BawtLogViewer  BawtLogViewer-[GetVersion].7z BawtLogViewer.bawt
Setup poApps         poApps-3.2.0.7z       poApps.bawt
Setup poClipboardViewer poApps-3.2.0.7z     poClipboardViewer.bawt
    
```

```

Setup file Tcl3D.bawt
# Builds the full version of Tcl3D, which depends on
# 3rd party libraries (OpenSceneGraph, SDL, FTGL).

Include "Tcl_Extended.bawt"
Include "OSG_Basic.bawt"

# Setup LibName      ZipFile          BuildFile      BuildOptions

Setup glfw           glfw-3.3.8.7z   glfw.bawt
Setup tcl3dFull      tcl3d-1.0.1.7z  tcl3dFull.bawt
    
```

```

Setup file Tcl_Distribution.bawt
# Use this Setup file to create a Tcl/Tk distribution.

# Builds Tcl/Tk with basic package libraries.
# Include "Tcl_Basic.bawt"

# Builds Tcl/Tk with extended package libraries including Tcl3D.
# Include "Tcl3D.bawt"

# Builds Tcl/Tk with extended package libraries.
Include "Tcl_Extended.bawt"

# Setup LibName      ZipFile          BuildFile      BuildOptions

# Tcl/Tk distribution as InnoSetup installer.
Setup InnoSetup      InnoSetup-6.2.2.7z  InnoSetup.bawt
Setup Redistributables  Redistributables.7z  Redistributables.bawt
Setup SetupTcl       SetupTcl.7z        SetupTcl.bawt
Setup SetupPython    SetupPython.7z     SetupPython.bawt
    
```

```

Setup file OSG_Basic.bawt
# Builds OpenSceneGraph with basic plugin libraries as needed by Tcl3D.

# Setup LibName      ZipFile          BuildFile      BuildOptions

# Tools needed for compilation of libraries.
Setup CMake          CMake-3.25.2.7z   CMake.bawt
Setup yasm           yasm-1.3.0.7z     yasm.bawt

# The following libraries can be compiled on Linux, but for OpenSceneGraph
# we use the libraries installed by the Linux distribution.
# Basic library needed by most other libraries.
Setup ZLib           ZLib-1.2.13.7z    ZLib.bawt      NoLinux
Setup xz            xz-5.4.1.7z       xz.bawt        NoLinux NoDarwin

# Image libraries needed by OpenSceneGraph.
Setup giflib        giflib-5.2.1.7z    giflib.bawt    NoLinux
Setup JPEG          JPEG-9.e.7z        JPEG.bawt      NoLinux
Setup PNG           PNG-1.6.39.7z     PNG.bawt       NoLinux
Setup TIFF          TIFF-4.5.0.7z     TIFF.bawt      NoLinux

Setup freeglut      freeglut-3.2.2.7z  freeglut.bawt  NoLinux NoDarwin
Setup Freetype      Freetype-2.10.4.7z  Freetype.bawt  NoLinux
Setup SDL           SDL-2.26.2.7z     SDL.bawt
Setup ffmpeg        ffmpeg-4.4.4.7z    ffmpeg.bawt    NoLinux32-arm
Setup openjpeg      openjpeg-2.5.3.7z  openjpeg.bawt

if { [UseVisualStudio "primary"] && [GetVisualStudioVersion] <= 2010 } {
    Setup jasper      jasper-2.0.14.7z   jasper.bawt    NoLinux NoDarwin
} else {
    Setup jasper      jasper-2.0.25.7z   jasper.bawt    NoLinux NoDarwin
}
Setup libressl      libressl-2.9.2.7z  libressl.bawt

# OpenSceneGraph 3rd party libraries.
if { [IsGccCompilerNewer "13.0.0"] && ! [IsWindows] } {
    Setup curl        curl-7.88.1.7z     curl.bawt
} else {
    Setup curl        curl-7.70.0.7z     curl.bawt
}

# OpenSceneGraph
    
```

Setup OpenSceneGraph	OpenSceneGraph-[GetOsgVersion].7z	OpenSceneGraph.bawt
Setup OpenSceneGraphData	OpenSceneGraphData-3.4.0.7z	OpenSceneGraphData.bawt

Setup file OSG_Extended.bawt

```
# Builds OpenSceneGraph with extended plugin libraries, as
# well as libraries depending on OpenSceneGraph like osgEarth.

Include "OSG_Basic.bawt"

# Setup LibName ZipFile BuildFile BuildOptions

# Extended OpenSceneGraph 3rd party libraries.
Setup Cal3D Cal3D-0.120.7z Cal3D.bawt NoLinux-arm NoLinux-riscv
if { [UseVisualStudio "primary"] && [GetVisualStudioVersion] <= 2013 } {
    Setup gdal gdal-2.2.0.7z gdal.bawt
    Setup geos geos-3.6.3.7z geos.bawt
} else {
    Setup gdal gdal-2.4.4.7z gdal.bawt NoLinux-riscv
    Setup geos geos-3.7.2.7z geos.bawt
}
Setup GLEW GLEW-2.2.0.7z GLEW.bawt
Setup Gl2ps Gl2ps-1.4.2.7z Gl2ps.bawt

# Libraries based on OpenSceneGraph.
Setup osgcal osgcal-0.2.1.7z osgcal.bawt NoLinux-arm NoLinux-riscv MaxParallel=Linux:1
MaxParallel=Windows-gcc:1

if { [UseVisualStudio "primary"] && [GetVisualStudioVersion] <= 2008 } {
    Setup osgearth osgearth-2.8.7z osgearth.bawt
} else {
    Setup osgearth osgearth-2.10.1.7z osgearth.bawt NoLinux-arm NoLinux-riscv
}
}
```

Setup file OSG_Distribution.bawt

```
# Use this Setup file to create an OpenSceneGraph distribution.

# Builds OpenSceneGraph with basic plugin libraries.
# Include "OSG_Basic.bawt"

# Builds OpenSceneGraph with extended plugin libraries, as
# well as libraries depending on OpenSceneGraph like osgEarth.
Include "OSG_Extended.bawt"

# Setup LibName ZipFile BuildFile BuildOptions

# OpenSceneGraph distribution as InnoSetup installer.
Setup InnoSetup InnoSetup-6.2.2.7z InnoSetup.bawt
Setup Redistributables Redistributables.7z Redistributables.bawt
Setup SetupOsg SetupOsg.7z SetupOsg.bawt
```

Setup file MiscLibs.bawt

```
# Builds miscellaneous libraries not related to Tcl or OpenSceneGraph.
# Note, that the libraries listed in this file are not updated or
# supported anymore.

Include "Tools.bawt"
Include "BasicLibs.bawt"

# Setup LibName ZipFile BuildFile BuildOptions

if { ( ! [UseVisualStudio "primary"] && [IsWindows] && [IsGccCompilerNewer "12.0.0"] ) || \
    ( [UseVisualStudio "primary"] && [GetVisualStudioVersion] >= 2022 ) } {
    # This boost version can only be compiled with
    # Windows: VS 2022 or newer.
    # Windows: gcc 12.0.0 or newer
    Setup Boost Boost-1.78.0.7z Boost.bawt
} elseif { ( [UseVisualStudio "primary"] && [GetVisualStudioVersion] >= 2015 ) || \
    ( ! [UseVisualStudio "primary"] && [IsWindows] ) || \
    ( ! [IsWindows] && [IsGccCompilerNewer "4.9.0"] ) } {
    # This boost version can only be compiled with
    # Windows: VS 2015 or newer.
    # Unix : gcc 4.9.0 or newer
}
```

```

Setup Boost          Boost-1.75.0.7z          Boost.bawt
} else {
  # This boost version cannot be compiled with MinGW gcc.
  Setup Boost        Boost-1.58.0.7z          Boost.bawt
}

Setup ccl             ccl-4.0.6.7z           ccl.bawt
Setup Eigen           Eigen-3.3.9.7z         Eigen.bawt
Setup fftw            fftw-3.3.9.7z         fftw.bawt
if { [UseVisualStudio "primary"] && [GetVisualStudioVersion] <= 2013 } {
  Setup GeographicLib GeographicLib-1.50.1.7z GeographicLib.bawt
} else {
  Setup GeographicLib GeographicLib-1.52.7z   GeographicLib.bawt
}
Setup GeographicLibData GeographicLibData.7z   GeographicLibData.bawt
Setup KDIS            KDIS-2.9.0.7z          KDIS.bawt
Setup libxml2         libxml2-2.10.3.7z       libxml2.bawt
Setup sqlite3         sqlite3-3.47.1.7z      sqlite3.bawt
Setup tinyxml2        tinyxml2-9.0.0.7z      tinyxml2.bawt
Setup Xerces          Xerces-3.2.4.7z        Xerces.bawt

```

Setup file WinTools.bawt			
# Builds miscellaneous tools for Windows.			
# Setup LibName	ZipFile	BuildFile	BuildOptions
Setup Blender	Blender-3.0.0.7z	Blender.bawt	
Setup DirectXTex	DirectXTex-2021_11.7z	DirectXTex.bawt	
Setup Doxygen	Doxygen-1.8.15.7z	Doxygen.bawt	
Setup Vim	Vim-9.0.0.7z	Vim.bawt	

3.3 Build Files

Build files include the logic needed to extract, configure, compile and distribute a library. They must define the following two procedures, where `libName` is replaced with the name of the library as specified as first parameter of the `Setup` procedure:

- `Init_libName { libName libVersion }`
- `Build_libName { libName libVersion buildDir instDir devDir distDir }`

The parameter values for these procedures are supplied by the **BAWT** framework.

<i>libName</i>	Library name as supplied with first parameter of procedure <code>Setup</code> .
<i>libVersion</i>	Library version extracted from source file name as supplied with second parameter of procedure <code>Setup</code> .
<i>buildDir</i>	<code>[file join [GetOutputBuildDir] \$libName]</code>
<i>instDir</i>	<code>[file join [GetOutputInstDir] \$libName]</code>
<i>devDir</i>	<code>[GetOutputDevDir]</code>
<i>distDir</i>	<code>[GetOutputDistDir]</code>

The logic of a `Build` file will be explained with the following excerpt of the `Build` file of Tcl package **udp**:

Build file udp.bawt	
# Copyright: 2016-2025 Paul Obermeier (obermeier@tcl3d.org)	
# Distributed under BSD license.	
#	
# BuildType: MSys / gcc	
<pre> proc Init_udp { libName libVersion } { SetScriptAuthor \$libName "Paul Obermeier" "obermeier@tcl3d.org" SetLibHomepage \$libName "https://core.tcl-lang.org/tcludp/" SetLibDependencies \$libName "Tcl" SetPlatforms \$libName "All" SetWinCompilers \$libName "gcc" </pre>	

```

}

proc Build_udp { libName libVersion buildDir instDir devDir distDir } {
    if { [UseStage "Extract" $libName] } {
        ExtractLibrary $libName $buildDir
    }

    if { [UseStage "Configure" $libName] } {
        set flags ""
        set cflags ""
        append cflags [GetPermissiveCFlags] " "
        append cflags [GetDarwinCFlags] " "
        if { [string trim $cflags] ne "" } {
            append flags "CFLAGS='$cflags' "
        }
        TeaConfig $libName $buildDir $instDir $flags
    }

    if { [UseStage "Compile" $libName] } {
        MSysBuild $libName $buildDir "install-binaries"
    }

    if { [UseStage "Distribute" $libName] } {
        StripLibraries "$instDir"
        LibFileCopy "$instDir" "$devDir/[GetTclDir]" "*" true
        LibFileCopy "$instDir" "$distDir/[GetTclDir]" "*" true
    }
    return true
}

```

The `Init_libName` procedure must call the following **BAWT** framework procedures:

<code>SetScriptAuthor</code>	Specify name and mail address of the build script author. This information is used for command line option <code>--authors</code> .
<code>SetLibHomepage</code>	Specify the homepage of the library. This information is used for command line option <code>--homepages</code> .
<code>SetLibDependencies</code>	Specify the dependencies of the library. If the library has no dependencies, specify <code>"None"</code> as parameter. Otherwise, a variable number of library names can be given. This information is used for command line option <code>--dependencies</code> .
<code>SetPlatforms</code>	Specify the supported platforms. Valid keywords are: <code>"Windows"</code> <code>"Linux"</code> <code>"Darwin"</code> <code>"All"</code> . This information is used for command line option <code>--platforms</code> .
<code>SetWinCompilers</code>	Specify the supported compilers on Windows. Optional. The first specified compiler is used as default. Valid keywords are: <code>"gcc"</code> <code>"vs"</code> . This information is used for command line option <code>--wincompilers</code> .

The `Build_libName` procedure must check, which stage or stages should be executed (using procedure `UseStage`) and supply appropriate Tcl commands for each stage.

The following four stages can be handled in a build file:

- Extract
- Configure
- Compile
- Distribute

See chapter 4 *Build Stages* for details on these stages and typical commands executed for each stage.

Errors can be indicated by calling the **BAWT** procedure `SetErrorMessage` and returning `false`.

Optionally a procedure named `Env_libName` may be specified in a build file. This procedure has the same signature as the `Build_libName` procedure and may be used to specify library specific

environment variables (using **BAWT** procedure `SetEnvVar`) or to add a value to the system environment variable `Path` (using **BAWT** procedure `AddToPathEnv`).

The following excerpt from the `Tcl` build file shows a usage example:

```
proc Env_Tcl { libName libVersion buildDir instDir devDir distDir } {
    SetEnvVar      "TCLLIBPATH" "$devDir/[GetTclDir]/lib"
    AddToPathEnv  "$devDir/opt/$libName/bin"
}
```

Another optional procedure named `Test_libName` was introduced in **BAWT 3.0**. This procedure has the same signature as the `Build_libName` procedure and may be used to execute test scripts using command line option `--test`.

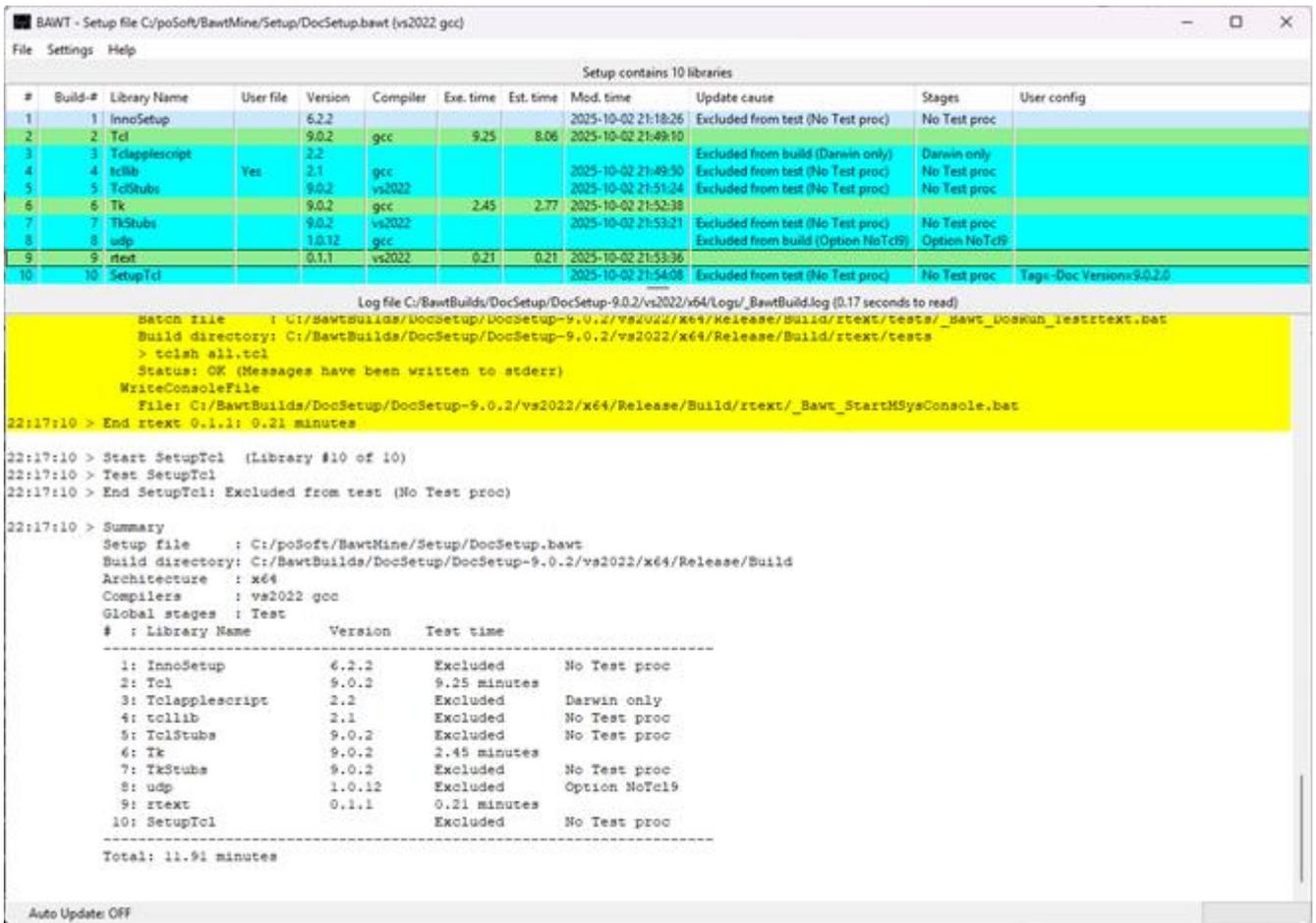
The following excerpt from the `Tcl` build file shows how to execute the `Tcl` test suite by calling **BAWT** procedure `MSysTest`:

```
proc Test_Tcl { libName libVersion buildDir instDir devDir distDir } {
    if { [UseStage "Test" $libName] } {
        MSysTest $libName $buildDir "test"
    }
    return true
}
```

It is also possible to execute a shell test script instead of `MSysTest`:

```
proc Test_poApps { libName libVersion buildDir instDir devDir distDir } {
    if { [UseStage "Test" $libName] } {
        if { [IsUnix] } {
            MSysRun $libName "Test$libName" "$buildDir/TestPrograms" "./RunTests.sh"
        } else {
            DosRun  $libName "Test$libName" "$buildDir/TestPrograms" "RunTests.bat"
        }
    }
    return true
}
```

If using the [graphical log viewer](#), the test mode is automatically detected and the display of test results is switched on, as shown in the next figure.



3.3.1 User supplied build files

BAWT version 2.0 introduced the functionality of user supplied build files, which allows to add custom build files for existing libraries without the need to change the default build files.

To create a user supplied build file, make a copy of the build file (ex. *tcllib.bawt*) and give the copied file the name *tcllib_User.bawt*. By appending the string *_User* to the root file name, BAWT automatically detects the file as a user supplied build file and uses this file instead of the original build file.

You can then edit the user supplied build file according to your needs, ex. do not create the *critcl* based modules for *tcllib*.

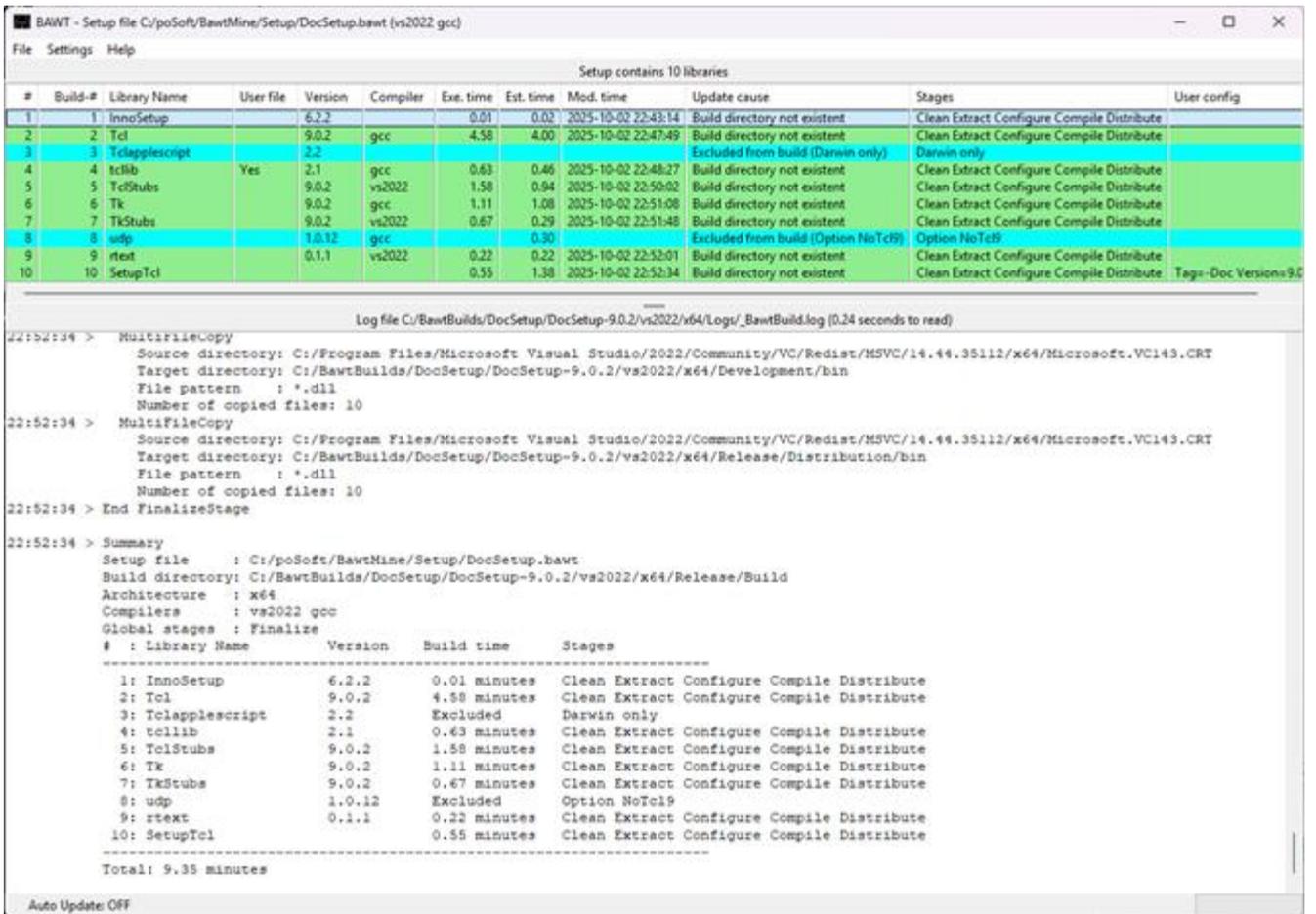
The user supplied build scripts must be located in directories from the library search paths, see chapter 3.1.3 *Directory access*.

Note, that user supplied build scripts are not considered in action `--update`, see chapter 5 Build Process.

You may also give the user supplied build file any name you like. Then you have to notify BAWT to use that file for a specific library by using command line option `--user`.

If you do not want to use the user supplied files, there is no need to delete or rename them. Specify command line option `--nouserbuilds` to disable all user build files.

If using the [graphical log viewer](#), the application of a user supplied build file is indicated in column *User*, see library *tcllib* in next figure.



3.3.2 User configurable build files

Some of the library build files are already setup to supply user configuration options. These configuration options can be supplied using the following methods:

As command line option `--copt`

As option string of the `Setup` procedure, see chapter [3.2 Setup Files](#)

The following build scripts currently support user configuration options:

Build script	User options
Tcl.bawt	Build static tclsh: Static=ON OFF. Default: OFF.
Tk.bawt	Build static wish: Static=ON OFF. Default: OFF.

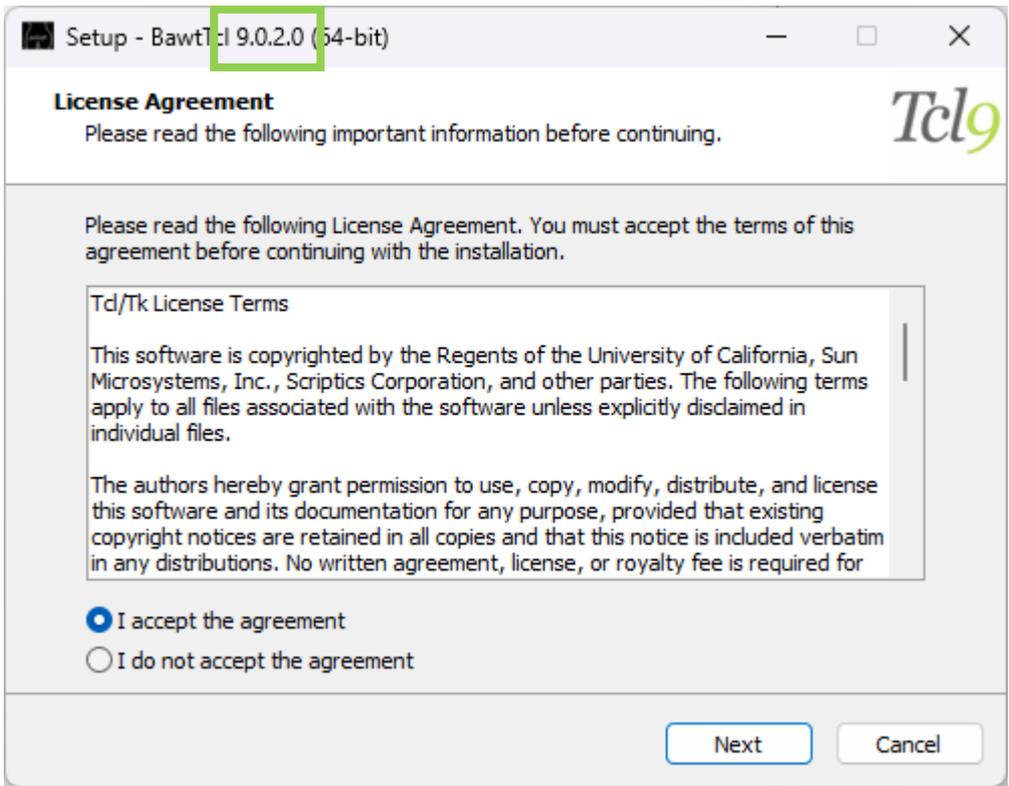
Build script	User options
SetupOsg.bawt	Tag string for generated Setup file name: <code>Tag=XXX</code> Version string used for InnoSetup: <code>Version=XXX</code>
SetupPython.bawt	
SetupTcl.bawt	

The following example using Tcl version 9.0.2

```
--copt SetupTcl 'Tag=-BI' --copt SetupTcl 'Version=9.0.2.0'
```

generates an InnoSetup file with the following name:

`SetupTcl-BI-9.0.2-x64_Bawt-3.2.0.exe`



Build script	User options
SetupTcl.bawt	Setup type for Linux: Linux=tar deb. Default: tar. Setup type for Darwin: Darwin=tar dmg. Default: tar.

Example creating Debian distribution files on Linux and DMG files on Darwin :

```
--copt SetupTcl 'Linux=deb' -copt SetupTcl 'Darwin=dmg'
```

Note, that on Windows only InnoSetup distributions are supported. For a description of the different Unix setup types, see the section regarding creation of software distribution files in chapter [5.1.4 Use the generated libraries](#).

Build script	User options
tcl3dFull.bawt	Use static SDL library: StaticSDL=ON OFF. Default: OFF. Currently only supported for Visual Studio builds.

Example:

```
--copt tcl3dFull 'StaticSDL=ON'
```

Build script	User options
tcllib.bawt	Toggle critcl based compilation: Critcl=ON OFF. Default: ON. Build dtplite starpack on Windows: Dtplite=ON OFF. Default: ON.

Example:

```
--copt tcllib 'Critcl=OFF'
```

Build script	User options
tcltls.bawt	Toggle hardening: Hardening=ON OFF. Default: ON.

Example:

```
--copt tcltls 'Hardening=OFF'
```

If `tcltls` is compiled with hardening set to ON, it is compiled with option `-fstack-protector-all`, which needs the `libssp-0.dll` library. That library is automatically copied into the `Tcl/bin` directory. Starting with gcc 13, the stack protector library is automatically linked statically. If hardening is set to OFF, `tcltls` does not need this external dependency.

Build script	User options
SWIG.bawt	Add Tcl dependency for SWIG test-suite: <code>AddTcl=ON OFF</code> . Default: OFF.

Example:

```
--copt SWIG 'AddTcl=ON'
```

Build script	User options
OpenSceneGraph.bawt	Toggle example compilation: <code>-DBUILD_OSG_EXAMPLES=ON OFF</code> . Default: OFF. Keep the plugin directory structure: <code>KeepPluginFolder=ON OFF</code> . Default: OFF.

Example:

```
--copt OpenSceneGraph '-DBUILD_OSG_EXAMPLES=ON'
```

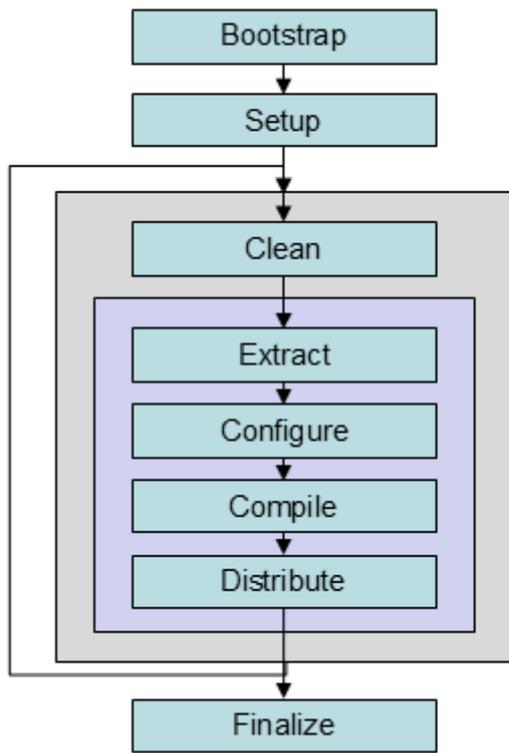
Build script	User options
osgearth.bawt	Toggle example compilation: <code>-DBUILD_OSGEARTH_EXAMPLES=ON OFF</code> . Default: OFF.

Example:

```
--copt osgearth '-DBUILD_OSGEARTH_EXAMPLES=ON'
```

4 Build Stages

This chapter describes the stages used in the **BAWT** framework to build the libraries specified in a *Setup* file.



The stages are grouped into global and library specific ones. The global stages `Bootstrap`, `Setup` and `Finalize` are called only once per **BAWT** execution, the library specific stages are called once for each library.

Four of the library specific stages (`Extract`, `Configure`, `Compile`, `Distribute`) are user configurable. Actions for these stages must be specified in the library *Build* files.

4.1 Stage Bootstrap

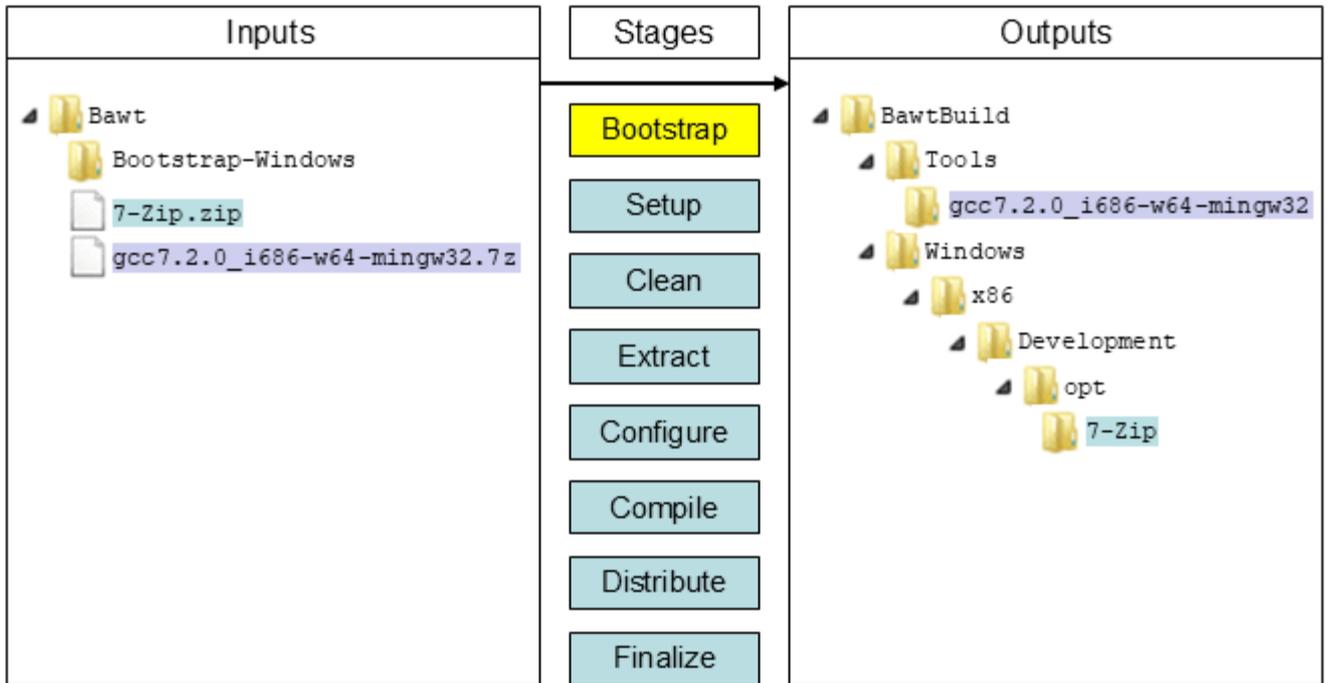
Extract and copy bootstrap tools.

This stage is executed automatically on each invocation of *Bawt.tcl*.

It is not executed, if command line option `--list` is specified.

BAWT needs the **7-Zip** program to extract the library source distributions. For Windows and Darwin, a version of the **7-Zip** program is included in the **BAWT** framework. On Linux **7-Zip** is typically already available with the operating system or can be installed as Linux package `p7zip` or `p7zip-full`.

On Windows lots of the libraries are built with the MSYS/MinGW suite. Different versions of MSYS/MinGW are available on the **BAWT** download site.



Command line options influencing this stage:

- [--gccversion](#)
- [--architecture](#)
- [--toolsdir](#)

The 7-Zip distribution itself must be compressed with standard ZIP, so that it can be extracted with the `vfs::zip` package contained in the `tclkit`. All other tools and libraries are compressed in 7-Zip format because of better compression rates (Example: MSYS/MinGW is 2 times smaller with 7z).

4.2 Stage Setup

Read and execute the specified *Setup* file.

This stage is executed automatically on each invocation of *Bawt.tcl*.

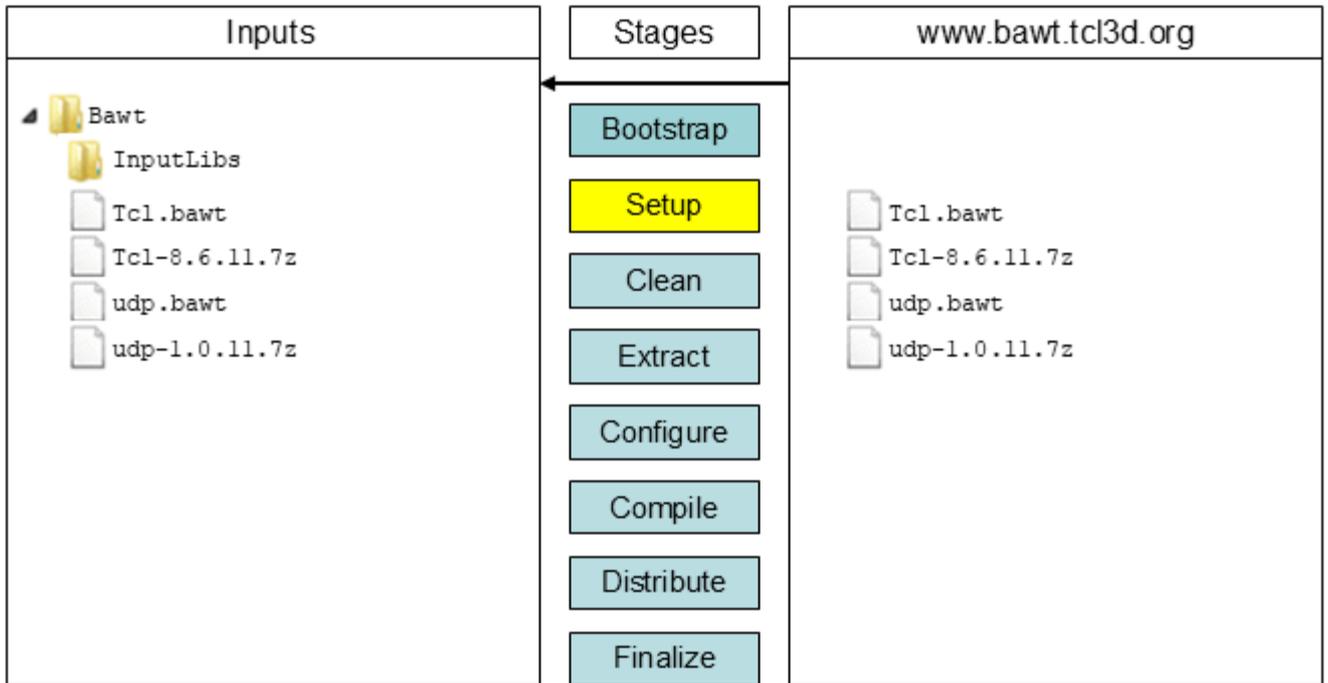
Check for existence of the library source code (either as a 7z file or directory) as well as the according *Build* file. If these do not exist in the library directory *InputLibs* of the current working directory (additional directories can be added with command line option [--libdir](#)) or are older than those available on the **BAWT** website, they are downloaded from the **BAWT** website.

If this fails, a fatal error is thrown and the build process is stopped.

The version number of the library is extracted from the file or directory name of the library.

If build action is set to *Update*, the necessary build stages are determined according to the existence of the library source and *Build* files as well as to the modification times of the corresponding build and install directories.

Checking for newer versions and automatic downloading may be skipped by specifying command line option [--noonline](#).

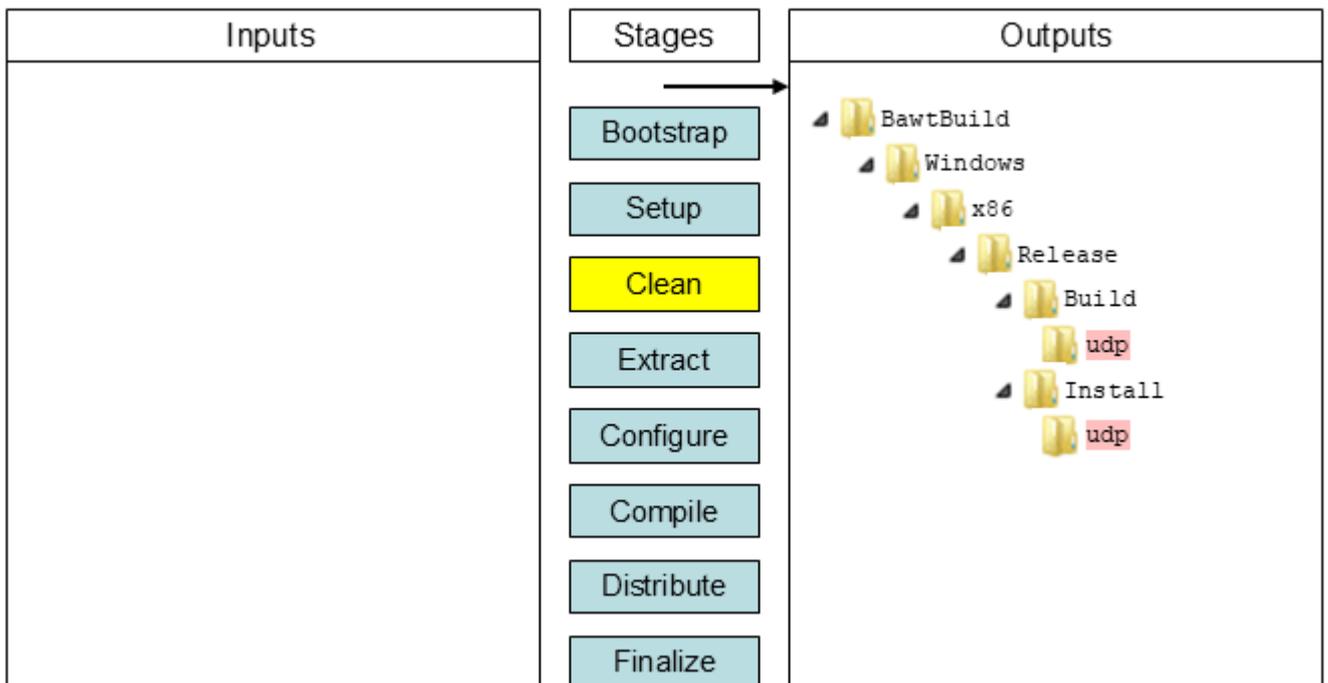


Command line options influencing this stage:

- [--noonline](#)
- [--norecursive](#)
- [--sort](#)
- [--url](#)

4.3 Stage Clean

Remove library specific build and install directory.



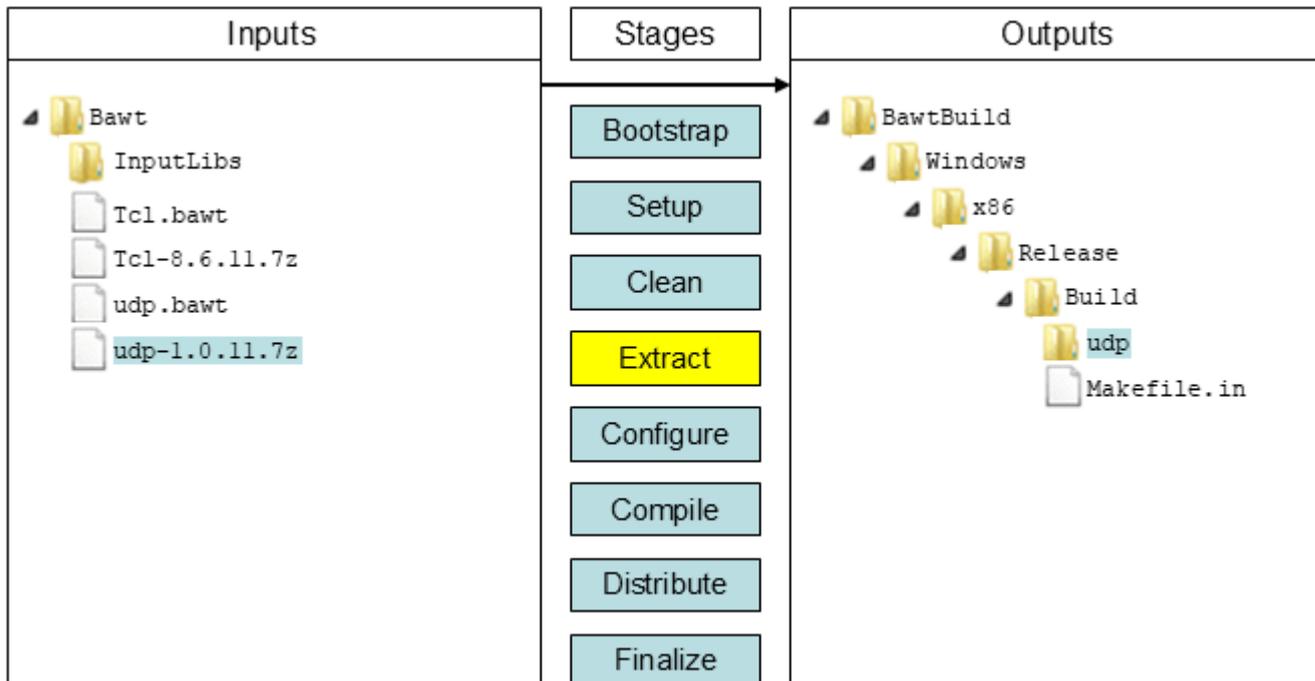
Command line options influencing this stage:

- [--clean](#)

[--timeout](#)

4.4 Stage Extract

Extract library source code into build directory.



In stage `Extract` the library source code will be extracted and copied into the build directory. This is achieved by calling the **BAWT** procedure `ExtractLibrary`, which cares about having either a source directory or a compressed source file.

Ideally the source code can be compiled without any changes. If changes have to be done, it is preferred not to edit the source code manually, but make the changes in the build script after extraction.

BAWT has two utility procedures for this purpose:

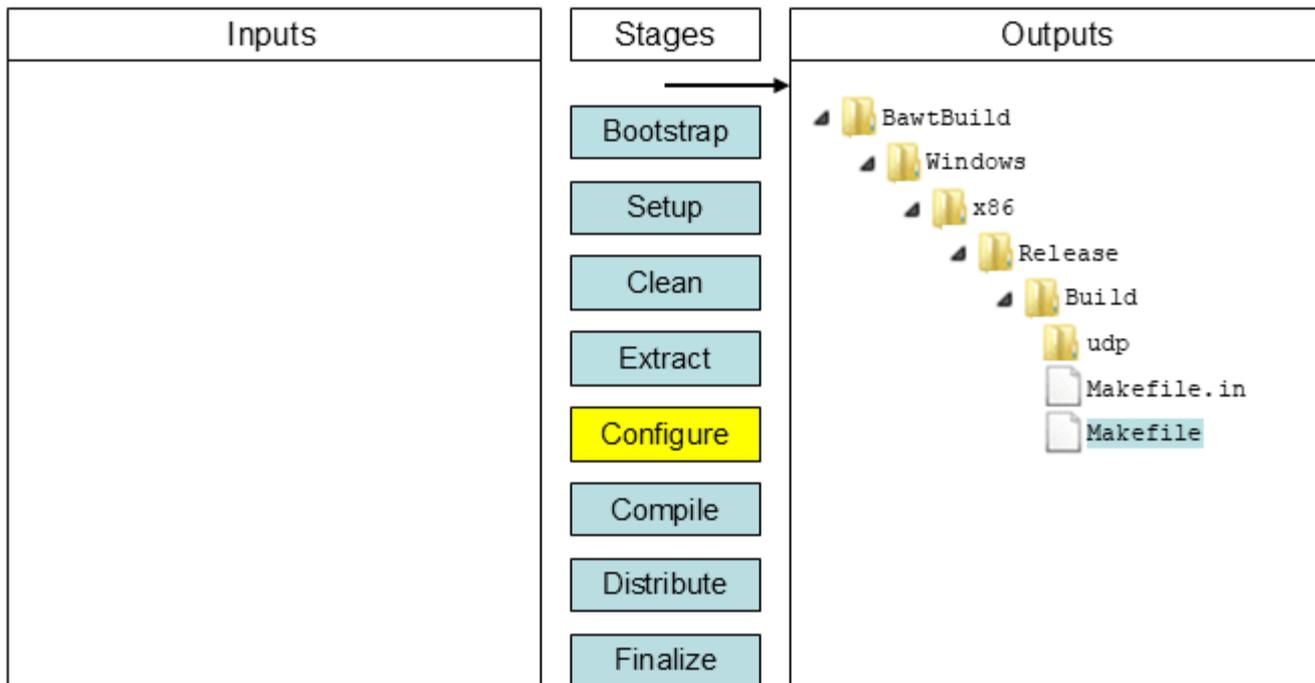
- `ReplaceLine`
- `ReplaceKeywords`

Command line options influencing this stage:

[--extract](#)

4.5 Stage Configure

Configure library for compilation.



In stage `Configure` the library will be configured, which generates the appropriate make files for the chosen compiler and platform.

The following high-level **BAWT** procedures are available for configuration tasks:

- `CMakeConfig` when using the CMake build infrastructure.
- `MsysConfig` when using a configure script with “standard” options.
- `TeaConfig` when using the Tcl Extension Architecture for Tcl packages.

See the source code of `Bawt.tcl` to get the default options set by these procedures.

If the build infrastructure does not fit any of the mentioned one above, the configuration command must be built up as a Tcl string and executed with the generic **BAWT** procedure `MsysRun`.

See the miscellaneous build scripts for usage examples.

The following **BAWT** procedures are typically used for configuration tasks:

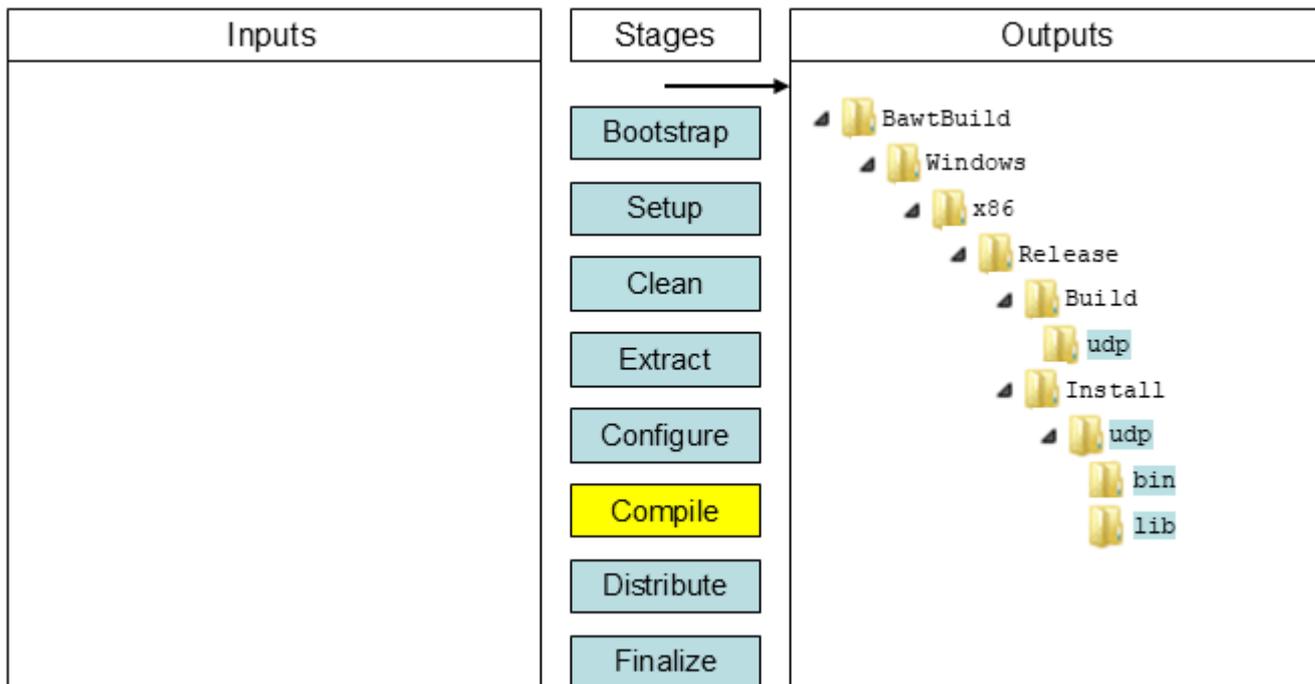
- `IsIntel`
- `IsArm`
- `IsRiscV`
- `IsDebugBuild`
- `IsReleaseBuild`
- `IsWindows`
- `IsLinux`
- `IsDarwin`
- `IsUnix`

Command line options influencing this stage:

- [`--configure`](#)
- [`--architecture`](#)
- [`--compiler`](#)
- [`--gccversion`](#)
- [`--buildtype`](#)
- [`--copt`](#)

4.6 Stage Compile

Compile and install library.



In stage `Compile` the library will be compiled and installed.

The following high-level **BAWT** procedures are available for compilation tasks:

- `CMakeBuild` when using the CMake build infrastructure.
- `MSysBuild` when using the Tcl Extension Architecture for Tcl packages.

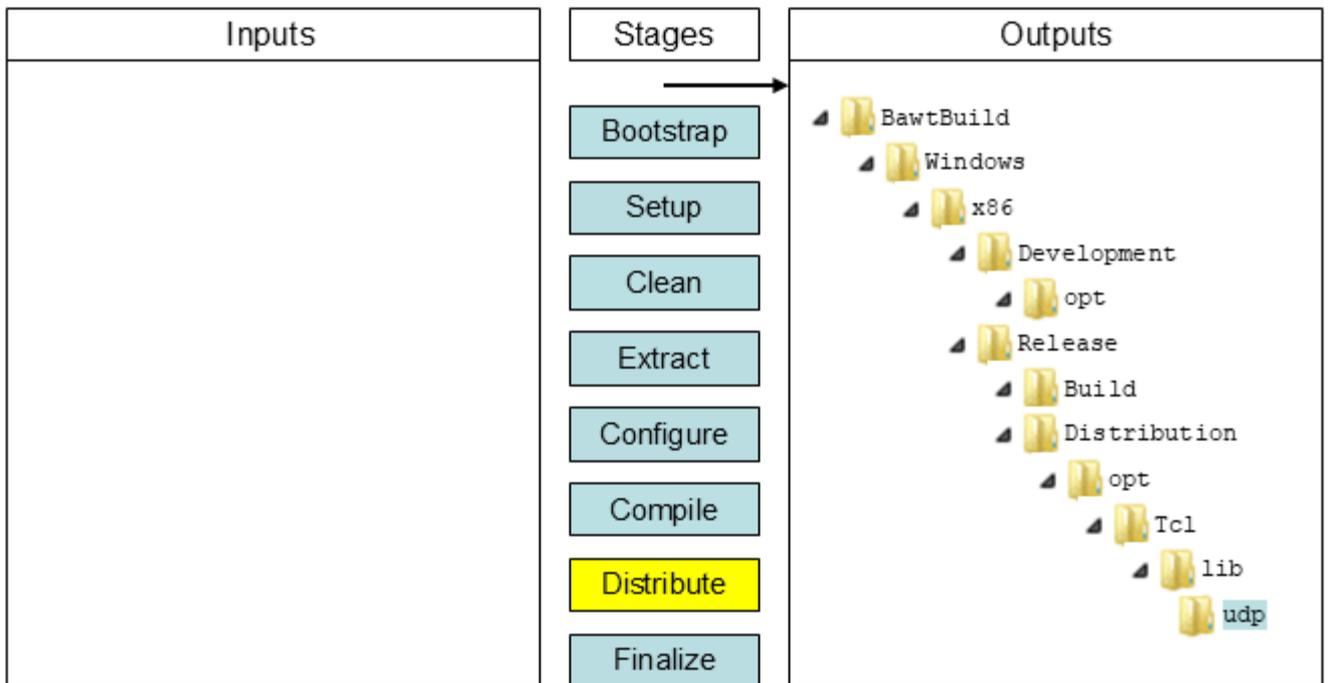
If the build infrastructure does not fit any of the two mentioned above, the compilation command must be built up as a Tcl string and executed with either **BAWT** procedure `MSysRun` or `DosRun`.

Command line options influencing this stage:

[--compile](#)
[--numjobs](#)
[--nostrip](#)
[--noimportlibs](#)

4.7 Stage Distribute

Copy relevant files into developer and user distribution directories.



In stage `Distribute` the library will be copied into the distribution and development directories. The following **BAWT** procedures are typically used for distribution tasks:

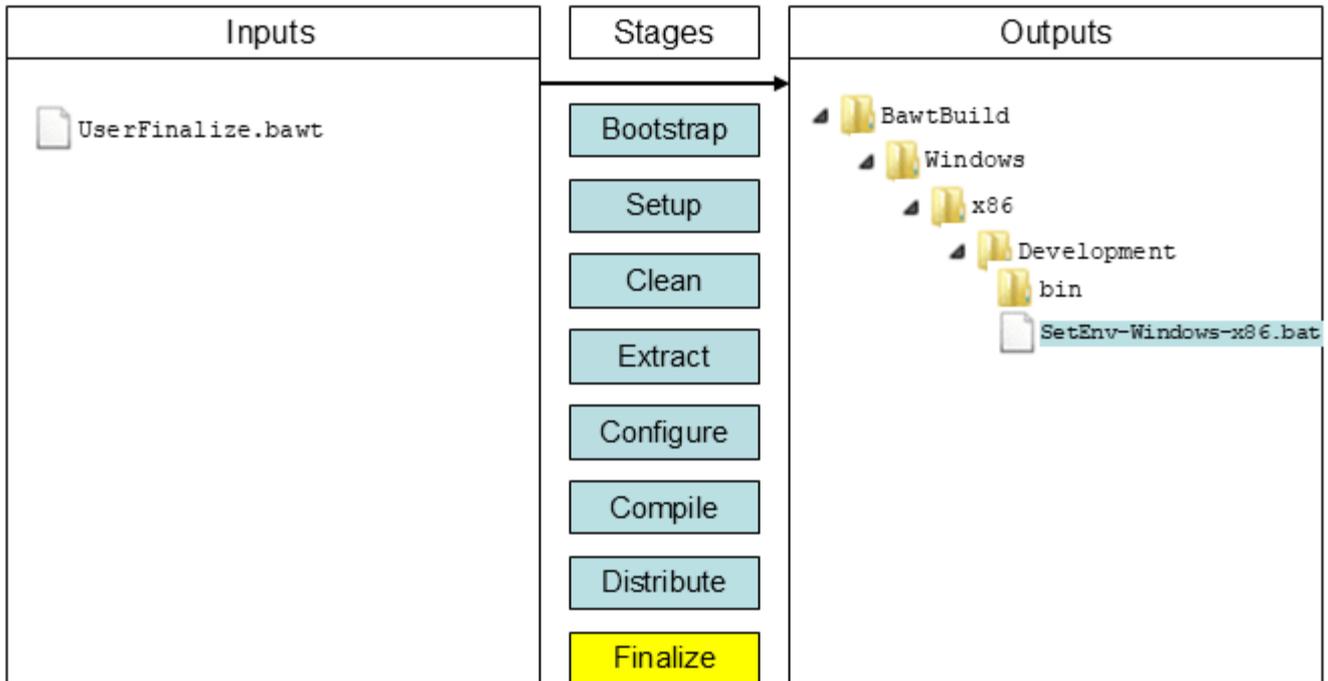
- `SingleFileCopy`
- `MultiFileCopy`
- `LibFileCopy`
- `FileRename`
- `UseTclPkgVersion`
- `IsDebugBuild`
- `IsReleaseBuild`
- `IsWindows`
- `IsLinux`
- `IsDarwin`
- `IsUnix`
- `ErrorAppend`

Command line options influencing this stage:

[--distribute](#)
[--noversion](#)

4.8 Stage Finalize

Perform final actions, optionally call user supplied `Finalize` procedure and print summary.



The Finalize stage is performed automatically at the end of the build process or can be manually selected with command line option [--finalize](#).

The Finalize stage creates an environment file in the *Development/bin* directory called *SetEnv-*.bat* or *SetEnv-*.sh*. It contains all the environment variables set by the *Env_libName* procedures of the libraries.

If running on Windows with Visual Studio it also copies the appropriate Visual Studio runtime libraries into the *Development/bin* directory. If you do not want to copy these runtime libraries, use command line option [--noruntimelibs](#).

To supply a user defined finalize action to **BAWT**, create a file containing a procedure named *Finalize*. See the file *UserFinalize.tcl* in **BAWT** directory *Setup* as an example.

You can use any standard Tcl procedure or one of the **BAWT** procedures like *Log* or *MultiFileCopy* in the *Finalize* procedure.

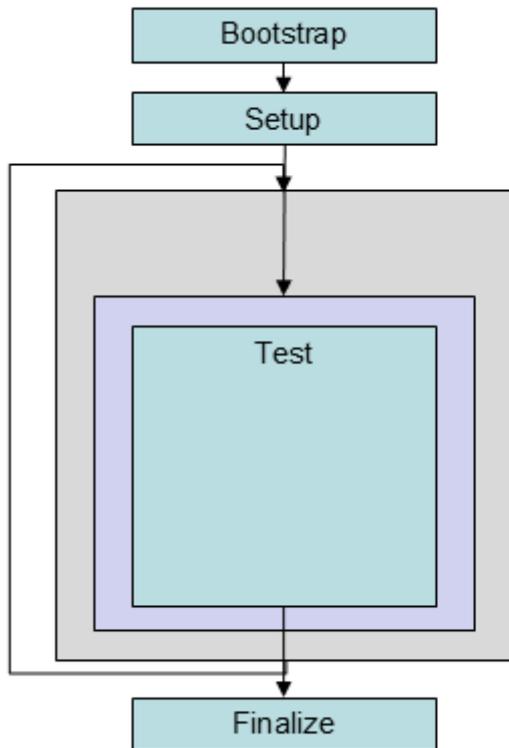
To make the file containing your Finalize procedure available for the **BAWT** build process, use command line option [--finalizefile](#).

Command line options influencing this stage:

- [--finalize](#)
- [--finalizefile](#)
- [--noruntimelibs](#)

4.9 Stage Test

Perform test scripts as specified in procedure *Test_libName*.



See the section regarding test procedures `Test_libName` in chapter [0 Build Files](#).

The Test stage must be manually selected with command line option `--test`.

5 Build Process

This chapter gives insight into the BAWT build process from the perspective of a user of BAWT as well as from the perspective of a developer, who wants to extend BAWT with new libraries.

5.1 User Perspective

As described in the previous chapter a specific stage can be executed with one of the following command line action options. These specific action options are typically only used when integrating a new library into BAWT.

```
--clean      : Clean library specific build and install directories.
--extract    : Extract library source from a ZIP file or a directory.
--configure  : Perform the configure stage of the build process.
--compile    : Perform the compile stage of the build process.
--distribute : Perform the distribution stage of the build process.
--finalize   : Generate environment file and call user supplied Finalize procedure.
```

The following global command line action options are typically used for building or updating the BAWT libraries.

```
--complete  : Perform the following stages in order:
               clean, extract, configure, compile, distribute, finalize.
--update    : Perform necessary stages depending on modification times.
               Note: Global stage finalize is always executed.
--simulate  : Simulate update action without actually building libraries.
--touch     : Set modification times of library build directories to current time.
```

Option `--complete` makes a complete rebuild of the specified libraries, while `--update` checks, which libraries have to be rebuilt.

The necessary build stages are determined according to the existence of the library source and *Build* files as well as to the modification times of the corresponding build directories.

It is also checked, if the build of a library has been cancelled or stopped by checking for the existence of a so-called *Progress File*, which is created in the *Logs* directory at the start of a library build and deleted after a successful library build.

Additionally, a check is performed, if a library is dependent of another library, which has been rebuilt. This recursive dependency checking can be switched off with command line option `--norecursive`.

The `--simulate` option performs the same actions as the `--update` option, but does not build anything. It just prints out, which libraries would be rebuilt, if you would execute the `--update` command line option.

It often happens, that only cosmetic changes are done to a *Build* file, which would cause this library (and all dependent libraries) to be rebuilt. To avoid rebuilding of these libraries, specify the option `--touch`, which sets the modification times of the build directories to the current date and time.

5.1.1 Use Case: Cosmetic change of Build file *Tcl.bawt*

Due to the number of dependencies, a change of *Build* file *Tcl.bawt* would cause a lot of libraries to be rebuilt, as the next screenshot of the *BawtLogViewer* shows, when executing a `--simulate` run.

BAWT - Setup file C:/poSoft/BawtMine/Setup/DocSetup.bawt (vs2022 gcc)

File Settings Help

Setup contains 10 libraries

#	Build-#	Library Name	User file	Version	Compiler	Exe. time	Est. time	Mod. time	Update cause	Stages	User config
1	1	InnoSetup		6.2.2		Simulation mode	0.02	2025-10-02 21:18:26			
2	2	Tcl		9.0.2	gcc	Simulation mode	4.00	2025-10-02 21:41:40	Build file newer than build dir		
3	3	Tclapplescript		2.2					Excluded from build (Darwin only)		
4	4	tcllib	Yes	2.1	gcc	Simulation mode	0.46	2025-10-02 21:23:40	Recursive dependency on Tcl		
5	5	TclStubs		9.0.2	vs2022	Simulation mode	0.94	2025-10-02 21:25:15	Build directory not existent		
6	6	Tk		9.0.2	gcc	Simulation mode	1.08	2025-10-02 21:26:21	Recursive dependency on Tcl		
7	7	TkStubs		9.0.2	vs2022	Simulation mode	0.29	2025-10-02 21:26:56	Recursive dependency on TclStubs		
8	8	udp		1.0.12	gcc		0.30		Excluded from build (Option NoTcl9)		
9	9	rtext		0.1.1	vs2022	Simulation mode	0.22	2025-10-02 21:27:09	Recursive dependency on TclStubs		
10	10	SetupTcl				Simulation mode	1.38	2025-10-02 21:27:40	Recursive dependency on All		Tag=-Doc Version=9.0.2.0

Log file C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Logs/_BawtBuild.log (0.16 seconds to read)

```

21:43:54 > End udp: Excluded from build (Option NoTcl9)
21:43:54 > Start rtext 0.1.1 (Library #9 of 10)
      Build types : Release
      Update cause: Recursive dependency on TclStubs
21:43:54 > End rtext 0.1.1: Simulation mode
21:43:54 > Start SetupTcl (Library #10 of 10)
      Build types : Release
      Update cause: Recursive dependency on All
21:43:54 > End SetupTcl : Simulation mode
21:43:54 > Summary
Setup file      : C:/poSoft/BawtMine/Setup/DocSetup.bawt
Build directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build
Architecture   : x64
Compilers      : vs2022 gcc
Global stages  : None
# : Library Name      Version      Build action  Build cause
-----
1: InnoSetup          6.2.2       None          None
2: Tcl                 9.0.2       Update        Build file newer than build dir
3: Tclapplescript     2.2         None          Darwin only
4: tcllib              2.1         Update        Recursive dependency on Tcl
5: TclStubs           9.0.2       Update        Build directory not existent
6: Tk                  9.0.2       Update        Recursive dependency on Tcl
7: TkStubs            9.0.2       Update        Recursive dependency on TclStubs
8: udp                 1.0.12     None          Option NoTcl9
9: rtext               0.1.1       Update        Recursive dependency on TclStubs
10: SetupTcl          0.1.1       Update        Recursive dependency on All
-----
Total: 0.00 minutes
  
```

Auto Update: OFF

To avoid the rebuild of all of these libraries, which may take a lot of time, we execute a `--touch` run. Note the execution of the `DirTouch` procedure of the BAWT framework shown in the text widget in the lower half of the window.

BAWT - Setup file C:/poSoft/BawtMine/Setup/DocSetup.bawt (vs2022 gcc)

File Settings Help

Setup contains 10 libraries

#	Build-#	Library Name	User file	Version	Compiler	Exe. time	Est. time	Mod. time	Update cause	Stages	User config
1	1	InnoSetup		6.2.2		0.00	0.02	2025-10-02 21:18:26			
2	2	Tcl		9.0.2	gcc	0.00	4.00	2025-10-02 21:49:10			
3	3	Tclapplescript		2.2					Excluded from build (Darwin only)	Darwin only	
4	4	tcllib	Yes	2.1	gcc	0.00	0.46	2025-10-02 21:49:50			
5	5	TclStubs		9.0.2	vs2022	0.00	0.94	2025-10-02 21:51:24			
6	6	Tk		9.0.2	gcc	0.00	1.08	2025-10-02 21:52:38			
7	7	TkStubs		9.0.2	vs2022	0.00	0.29	2025-10-02 21:53:21			
8	8	udp		1.0.12	gcc		0.30		Excluded from build (Option NoTcl9)	Option NoTcl9	
9	9	rtext		0.1.1	vs2022	0.00	0.22	2025-10-02 21:53:36			
10	10	SetupTcl				0.00	1.38	2025-10-02 21:54:08			Tag=Doc Version=9.0.2.0

Log file C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Logt/_BawtBuild.log (0.18 seconds to read)

```

22:32:12 > Build rtext 0.1.1 (Release)
  DirTouch
  Directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/rtext
22:32:27 > End rtext 0.1.1: 0.00 minutes

22:32:27 > Start SetupTcl (Library #10 of 10)
  Build types : Release
22:32:27 > Build SetupTcl (Release)
  DirTouch
  Directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/SetupTcl
22:32:27 > End SetupTcl : 0.00 minutes

22:32:27 > Summary
  Setup file      : C:/poSoft/BawtMine/Setup/DocSetup.bawt
  Build directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build
  Architecture   : x64
  Compilers      : vs2022 gcc
  Global stages  : Touch
  # : Library Name      Version      Build time
  -----
  1: InnoSetup          6.2.2       0.00 minutes
  2: Tcl                9.0.2       0.00 minutes
  3: Tclapplescript    2.2         Excluded      Darwin only
  4: tcllib             2.1         0.00 minutes
  5: TclStubs          9.0.2       0.00 minutes
  6: Tk                9.0.2       0.00 minutes
  7: TkStubs           9.0.2       0.00 minutes
  8: udp               1.0.12      Excluded      Option NoTcl9
  9: rtext             0.1.1       0.00 minutes
 10: SetupTcl          0.00 minutes
  -----
  Total: 0.00 minutes
  
```

Auto Update: OFF

If we now perform an `--update` run, none of the libraries are rebuilt.

#	Build-#	Library Name	User file	Version	Compiler	Exe. time	Est. time	Mod. time	Update cause	Stages	User config
1	1	InnoSetup		6.2.2		0.00	0.02	2025-10-02 21:18:26		None	
2	2	Tcl		9.0.2	gcc	0.00	4.00	2025-10-02 21:49:10		None	
3	3	Tclapplescript		2.2					Excluded from build (Darwin only)	Darwin only	
4	4	tcllib	Yes	2.1	gcc	0.00	0.46	2025-10-02 21:49:50		None	
5	5	TclStubs		9.0.2	vs2022	0.00	0.94	2025-10-02 21:51:24		None	
6	6	Tk		9.0.2	gcc	0.00	1.08	2025-10-02 21:52:38		None	
7	7	TkStubs		9.0.2	vs2022	0.00	0.29	2025-10-02 21:53:21		None	
8	8	udp		1.0.12	gcc		0.30		Excluded from build (Option NoTcl9)	Option NoTcl9	
9	9	rtext		0.1.1	vs2022	0.00	0.22	2025-10-02 21:53:36		None	
10	10	SetupTcl				0.00	1.38	2025-10-02 21:54:08		None	Tag=Doc Version=9.0.2.0

Log file C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Log/_BawtBuild.log (0.18 seconds to read)

```

22:03:10 > MultiFileCopy
  Source directory: C:/Program Files/Microsoft Visual Studio/2022/Community/VC/Redist/MSVC/14.44.35112/x64/Microsoft.VC143.CRT
  Target directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Development/bin
  File pattern : *.dll
  Number of copied files: 10
22:03:10 > MultiFileCopy
  Source directory: C:/Program Files/Microsoft Visual Studio/2022/Community/VC/Redist/MSVC/14.44.35112/x64/Microsoft.VC143.CRT
  Target directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Distribution/bin
  File pattern : *.dll
  Number of copied files: 10
22:03:10 > End FinalizeStage
22:03:10 > Summary
  Setup file : C:/poSoft/BawtMine/Setup/DocSetup.bawt
  Build directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build
  Architecture : x64
  Compilers : vs2022 gcc
  Global stages : Finalize
  # : Library Name      Version      Build time   Stages
  -----
  1: InnoSetup          6.2.2       0.00 minutes None
  2: Tcl                9.0.2       0.00 minutes None
  3: Tclapplescript    2.2         Excluded    Darwin only
  4: tcllib            2.1         0.00 minutes None
  5: TclStubs          9.0.2       0.00 minutes None
  6: Tk                9.0.2       0.00 minutes None
  7: TkStubs           9.0.2       0.00 minutes None
  8: udp               1.0.12      Excluded    Option NoTcl9
  9: rtext             0.1.1       0.00 minutes None
  10: SetupTcl         0.00 minutes None
  -----
  Total: 0.00 minutes
  
```

Auto Update: Off

5.1.2 Compiler selection on Windows

On Linux and Darwin only the gcc compiler suite is supported.

On Windows gcc and Visual Studio are supported. Some packages can be compiled only with gcc or only with Visual Studio. More and more libraries can be compiled with either gcc or Visual Studio.

Starting with version 2.0, **BAWT** supports the notion of primary and secondary compilers on Windows. Which compilers are supported by a build script is indicated with BAWT procedure `SetWinCompilers`.

```

proc Init_tkdnd { libName libVersion } {
  SetScriptAuthor    $libName "Paul Obermeier" "obermeier@tcl3d.org"
  SetLibHomepage     $libName "https://github.com/petasis/tkdnd"
  SetLibDependencies $libName "CMake" "Tk"
  SetPlatforms       $libName "All"
  SetWinCompilers    $libName "gcc" "vs"
}
  
```

The above call of `SetWinCompilers` indicates, that the library can be compiled by both Visual Studio and gcc.

To see, which Windows compilers are supported, use the `--wincompilers` command line option or look for that information in the corresponding build files.

To determine, which compiler should be used in an actual compilation, there is the possibility to specify the compiler using command line option `--compiler`.

This option has been extended to not only accept `gcc` or `vs20XX` as arguments, but also a combination of both using a plus sign as separator, ex. `gcc+vs2019`.

If a library does not support the Windows compiler selected when calling BAWT, then that library is excluded from the build. The log file contains a message like the following:

```
15:02:30 > Start Boost 1.58.0 (Library #2 of 137)
           Build types : Release
15:02:30 > End Boost: Excluded from build (Compiler gcc not supported)
```

Behaviour before BAWT version 2.0:

If the chosen Windows compiler is Visual Studio, but the package only supports gcc, the gcc compiler was automatically chosen as secondary compiler, as the MSYS/MinGW suite is part of BAWT and therefore always available. The other way is not supported, as a Visual Studio compiler may not be available.

The following 3 options of choosing a compiler on Windows were available up to BAWT version 1.3.0.

BAWT 1.3.0	Command line option --compiler	SetWinCompilers		
		gcc	vs	gcc vs
Option 1	Not specified	gcc	Excluded	gcc
Option 2	--compiler gcc	gcc	Excluded	gcc
Option 3	--compiler vs20XX	gcc	vs	vs

Behaviour since BAWT version 2.0:

With BAWT 2.0 two new options have been added, which specify the primary and secondary compiler.

BAWT 2.0.0	Command line option --compiler	SetWinCompilers		
		gcc	vs	gcc vs
Option 1	Not specified	gcc	Excluded	gcc
Option 2	--compiler gcc	gcc	Excluded	gcc
Option 3	--compiler vs20XX	Excluded	vs	vs
Option 4	--compiler gcc+vs20XX	gcc	vs	gcc
Option 5	--compiler vs20XX+gcc	gcc	vs	vs

Options 1 and 2 work the same way as they did in BAWT versions before 2.0. Option 3 now does not compile packages supporting only gcc. This behaviour can now be achieved by specifying Option 4 (vs20XX+gcc).

To support this new functionality, several incompatible changes had to be implemented:

New procedures	Removed procedures
<i>SetCompilerVersions</i>	<i>GetVSCompilerVersionNumber</i>
<i>GetCompilerVersions</i>	<i>IsVSCompilerNewer</i>
<i>UseVisualStudio</i>	<i>IsVSCompiler</i>
<i>GetVisualStudioVersion</i>	<i>SetForceVSCompiler</i>
<i>NeedDll2Lib</i>	<i>ForceVSCompiler</i>

Procedure *GetCompilerVersion* now has a changed and extended signature.

Compilation of **Tcl/Tk** and all supported Tcl packages (everything included in *Setup* files *Tcl_Basic.bawt* and *Tcl_Extended.bawt*) is possible without using Visual Studio with the exception of building Visual Studio compatible Tcl and Tk stub libraries. Those stub libraries can only be compiled using Visual Studio.

To generate Visual Studio compatible Tcl and Tk import libraries (*.lib) the **BAWT** procedure `Dll2Lib` is used. It creates the import library from the DLL by using the **link.exe** program, which is part of Visual Studio.

If Visual Studio is not available, a warning message like the following is issued:

```
Warning > Dll2Lib tk86.lib: Creating import libraries needs VisualStudio
```

To avoid these warnings, add command line option `--noimportlibs`, if Visual Studio is not available or import libraries are not needed.

5.1.3 Online updates of libraries

If using the online update functionality, it is recommended that the local BAWT version is identical to the remote version on the BAWT server. If the local major or minor version is older than the remote version, a fatal error is generated:

```
FATAL > Remote major version 3.2.0 different to major local version 2.3.0
```

If only the patch version differs, a warning is issued.

You are able to download with different local and remote versions by specifying the `--noexit` command line option, but this is not recommended.

To have a consistent set of library versions or if using **BAWT** on a computer without internet connection, use the command line option `--noonline` to avoid checking for updates and automatic downloading of new libraries.

5.1.4 Use the generated libraries

To use the generated libraries, the following possibilities exist:

- Manually copy the appropriate directory.
- Use the `Finalize` procedure.
- Create a software distribution setup file

Manually copy the appropriate directories

Copy the appropriate directories from either the *Distribution* or *Development* directory to a suitable location on your computer.

For example, after executing the Setup file `Tcl_Basic.bawt` to generate a **Tcl** distribution for Windows, copy output directory `Development\opt\Tcl` to `C:\Tcl` and set the environment variables `PATH` and `TCLLIBPATH`.

Note, that the entries of the `PATH` variable on Windows are separated by semicolons (;). The entries of variable `TCLLIBPATH` are separated by spaces and directory paths must use slashes (/) instead of backslashes (\).

On Unix the environment variables are typically set in the shell resource file, ex. `.bashrc`:

Use the Finalize procedure

Instead of doing the copy manually, it is easier and faster to do the copying in the Finalize stage. The **BAWT** framework contains a template Finalize file `Setup/UserFinalize.bawt`, which is shown below. Adapt the installation paths according to your local needs.

```
# Example script for user supplied Finalize procedure.
```

```

#
# The procedure copies the generated Tcl distribution
# from the Development folder into a folder specified
# in your Path environment variable.
#
# You have to adapt the installation paths (tclRootDir)
# according to your needs.
#
# To execute the Finalize procedure, the name of this file
# must be specified on the BAWT command line with option
# "--finalizefile".

proc Finalize {} {
    Log "Finalize (User defined)"

    # For safety reasons this is just a dummy mode.
    # Remove the next lines to enable functionality.
    if { 1 } {
        Log "Finalize Dummy mode" 2 false
        return
    }

    if { [IsWindows] } {
        set tclRootDir "C:/opt"
    } elseif { [IsLinux] } {
        set tclRootDir "~/opt"
    } elseif { [IsDarwin] } {
        set tclRootDir "~/opt"
    } else {
        ErrorAppend "Finalize: Cannot determine operating system" "FATAL"
    }

    set tclInstDir [file join $tclRootDir "Tcl"]

    Log "Installing Tcl into $tclInstDir" 2 false
    DirDelete $tclInstDir

    MultiFileCopy [file join [GetOutputDevDir] [GetTclDir]] $tclInstDir "*" true
}

```

Create a software distribution setup file

There are currently three *Build* files to create software distribution setup files:

- *SetupTcl.bawt* to create a **Tcl** Batteries-Included software distribution
- *SetupPython.bawt* to create a **Python** software distribution
- *SetupOsg.bawt* to create an **OpenSceneGraph** software distribution

These scripts take all contents of the *Release/Distribution* directory and create a software distribution setup file. This setup file is created with **InnoSetup** on Windows and as a simple, self-extracting shell script for Unix platforms.

The *SetupTcl.bawt* script also supports creation of Debian distribution files using user configuration option `Linux=deb` on Debian and Ubuntu. On Darwin a DMG software distribution file can be created using user configuration option `Darwin=dmg`.

The software distribution setup file itself is generated in the *Release/Distribution* directory.

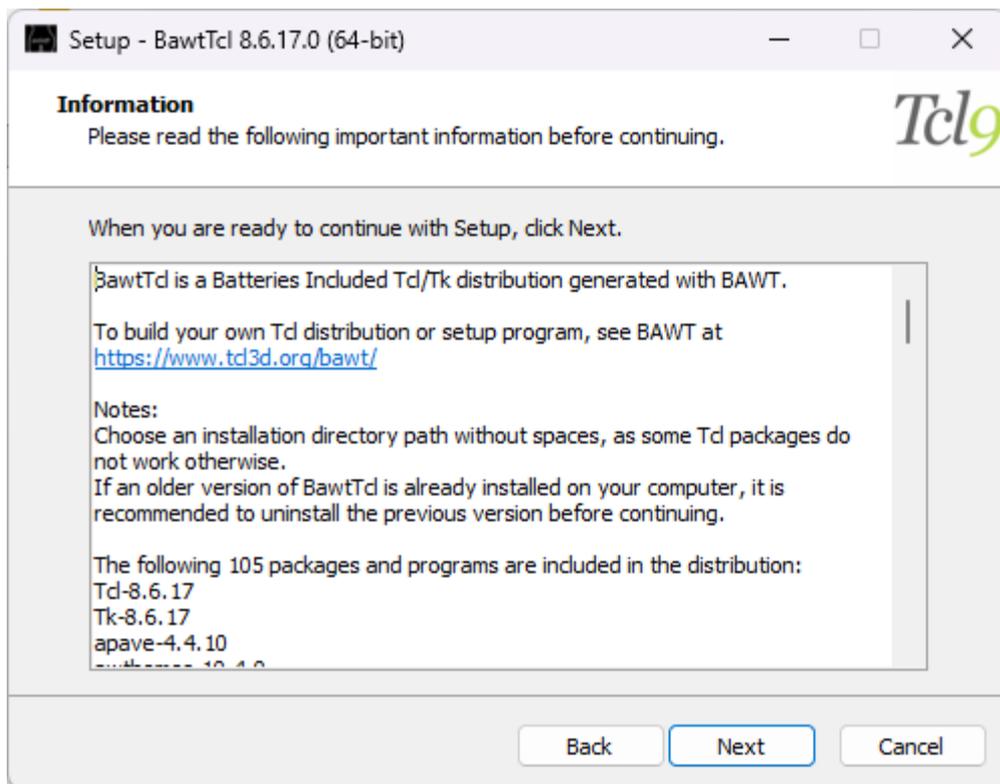
The software distribution setup file name for **Tcl/Tk** has the Tcl version, the architecture and the BAWT version used to build the distribution encoded into the file name.

Example: `SetupTcl-BI-8.6.17-x64_Bawt-3.2.0.exe`

The software distribution setup file name for **OpenSceneGraph** has the OSG version, the compiler version, the architecture and the BAWT version used to build the distribution encoded into the file name. Example: `SetupOsg-3.4.1-vs2013-x64_Bawt-3.2.0.exe`

In the same directory as the distribution setup files, text files named `SetupTcl-8.6.17.txt` resp. `SetupOsg-3.4.1.txt` are created, which list the contents of the software distribution setup file.

This list is used to display the contents of the **InnoSetup** based distribution setup file, see the following screenshot for an example.



For Unix (Linux and Darwin) a simple shell script-based distribution setup file is generated. If called without arguments, a simple usage message is displayed.

```
> ./SetupTcl-BI-8.6.17-x64_Bawt-3.2.0.sh

Usage: SetupTcl-BI-8.6.17-x64_Bawt-3.2.0.sh InstallationDirectory
Install folder Tcl into specified installation directory
```

If called with a not existing installation directory path, an error message is printed onto standard output.

```
> ./SetupTcl-BI-8.6.17-x64_Bawt-3.2.0.sh asdf

Installation directory asdf does not exist.
Check name or create manually.
```

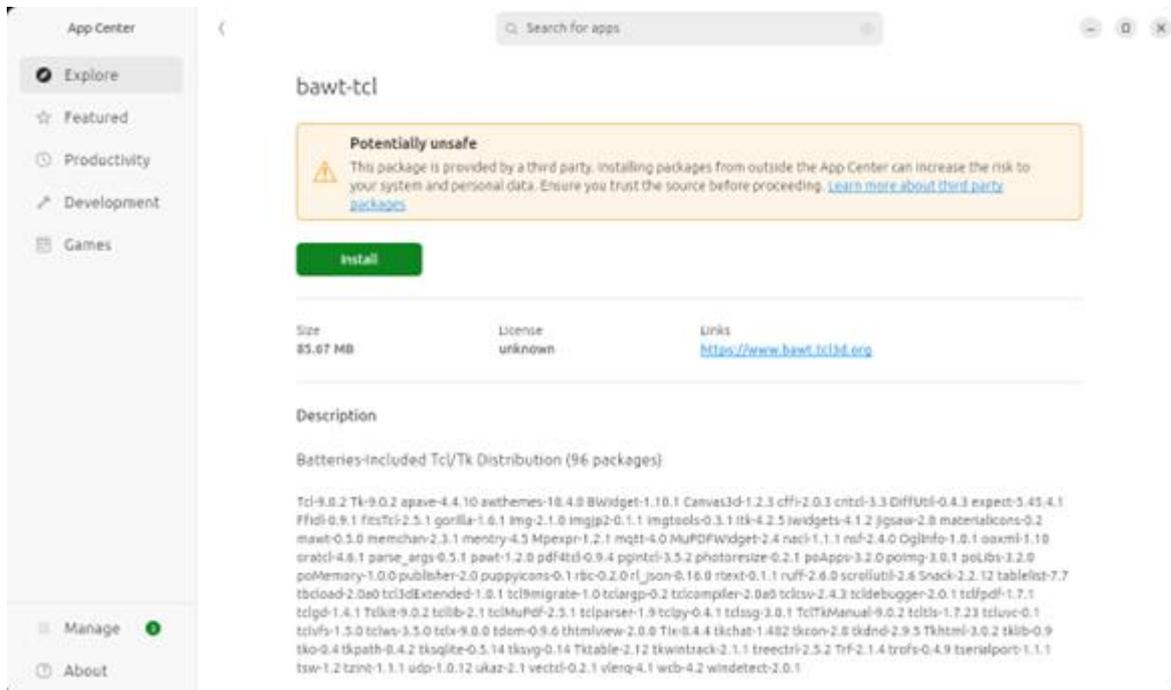
If called with a valid installation directory, the contents are extracted into that directory and a message on how to set the needed environment variables is printed onto standard output.

```
> ./SetupTcl-BI-8.6.17-x64_Bawt-3.2.0.sh ~/bin
Extracting Tcl into /home/obermeier/bin ...

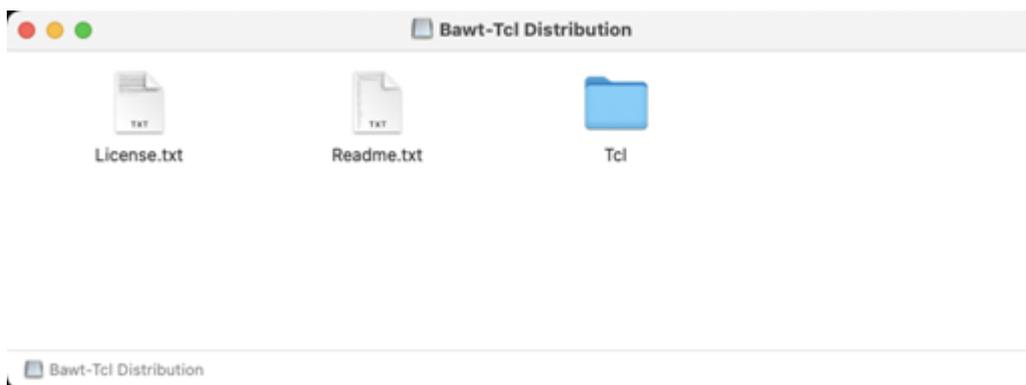
Add the following lines to your shell resource file (ex. ~/.bashrc):
```

```
export PATH="/home/obermeier/bin/Tcl/bin:$PATH"
export TCLLIBPATH="/home/obermeier/bin/Tcl/lib $TCLLIBPATH"
```

Debian *.deb* distribution files can be used to install a Bawt-Tcl distribution on Linux system supporting the dpkg format. The Tcl distribution will be installed into directory */opt*, which conforms to the Debian standards for custom packages. If you want to change that path, adapt the corresponding line in script *SetupTcl.bawt*. The control file for the Debian package manager is contained in file *SetupTcl.7z (DEBIAN/control)*.



Disk Image Files (DMG) can be used on MacOS to install a Bawt-Tcl distribution by dragging the *Tcl* distribution directory from the mounted DMG image and following the instructions in the *Readme.txt* file.



5.1.5 Change icons of executables

To change the icon of the generated *tclkits* and *starpacks* as well as the information shown about an executable on Windows (Resource), two command line options exist in the **BAWT** framework:

- [--iconfile](#)
- [--resourcefile](#)

The user supplied icon and resource files can be either located in the *Resources* directory. Then it is sufficient to just specify the name of the files. If the files are located at other places, the path name of the files must be absolute.

Use the icon file *poSoft.ico* and resource file *poSoft.rc* supplied by **BAWT** in directory *Resources* as starting point for your adapted ones.

If specifying your own resource file, do not change the name of the icon file in the following line of your resource file:

```
tk ICON DISCARDABLE "tclkit.ico"
```

The name must always be *tclkit.ico*.

If specifying a user supplied icon file with command line option `--iconfile`, the icon file will be copied into the build directory *Tclkit/kbskit/win* and renamed to *tclkit.ico*, so that it is possible to only specify an icon file without specifying a resource file.

Changes to the used icon and resource file are not considered by the BAWT update check process, so if using these options, it is necessary to at least rebuild package *tclkit* and its dependencies.

5.1.6 Sign executables

Signing executables is currently only available on Windows and needs program *signtool.exe*, which is part of Visual Studio.

Signing of executables is done by calling procedure *SignExecutables* in a build script. This call is typically done in the Distribution stage. See build file *BawtLogViewer.bawt* for an example.

There are two command line options controlling the signature process:

`--certfile` (mandatory)
`--timestampurl` (optional)

Directory *Resources* contains a self-signed certification file *poSoft.cer*, which is used for the BAWT distributions.

5.1.7 Parallel builds

All build environments used by BAWT support parallel compilation. The number of parallel build jobs can be specified globally for all libraries with command line option `--numjobs`.

Alternatively, the number of parallel build jobs can be restricted for specific libraries as additional parameter `MaxParallel` in the Setup procedure. See chapter [3.2 Setup Files](#) for a description of the Setup procedure and its parameters.

The following libraries consistently produce deadlocks when executed in parallel, so the number of parallel jobs is already limited in the corresponding BAWT Setup files by specifying option `MaxParallel=Windows-gcc:1`.

- CERTI
- PNG
- osgcal
- tserialport

Other libraries which occasionally tend to deadlock are the following:

- freeglut

- gdal
- geos
- openjpeg
- OpenSceneGraph
- osgearth
- SDL

Deadlocks have occurred until now only on Windows using the gcc compiler.

As reference point, the next table shows typical build times on my laptop for libraries needing 2 minutes or more. The laptop is equipped with an Intel QuadCore i7-4700 2.4Ghz with HyperThreading. 8 parallel compile jobs have been used.

Estimated build time	Libraries
~ 2 minutes	ccl libgd libwebp SetupTcl xz
~ 3 minutes	geos kdis TIFF
~ 4 minutes	SWIG tcltcl tcl3dFull
~ 5 minutes	gdal Tclkit Xerces
~ 6 minutes	curl gdal libressl Tcl
~ 7 minutes	boost ffmpeg Img
~ 9 minutes	fftw
~ 25 minutes	osgearth
~ 35 minutes	OpenSceneGraph

5.2 Developer Perspective

5.2.1 Upgrade a library

If you want to use a new version of a library already supported by **BAWT**, chances are high, that the existing build scripts still work with the new version.

So just pack the sources of the new version into a 7z file and edit the corresponding entry in the *Setup* file. Also check the comments of the library build script regarding manual changes to the source code.

If the library is a **Tcl** package, you might get warnings from the **Starpack** build scripts. This indicates, that you will have 2 different versions in the **Tcl** library directory, which might lead to troubles.

The following warnings are issued, when upgrading library tablelist 6.20 to tablelist 6.22:

```
MakeStarpack: Found more than 1 package with prefix tablelist*:
TclBasic-8.6.17/vs2019/x64/Development/opt/Tcl/lib/tablelist6.20
TclBasic-8.6.17/vs2019/x64/Development/opt/Tcl/lib/tablelist6.22
```

So, when upgrading one or more libraries, you should either remove the development and distribution directories and do a fresh rebuild. The other possibility is to search for the directories of the old version (*tablelist6.10* in the above example) and just remove these directories from the development and distribution directory.

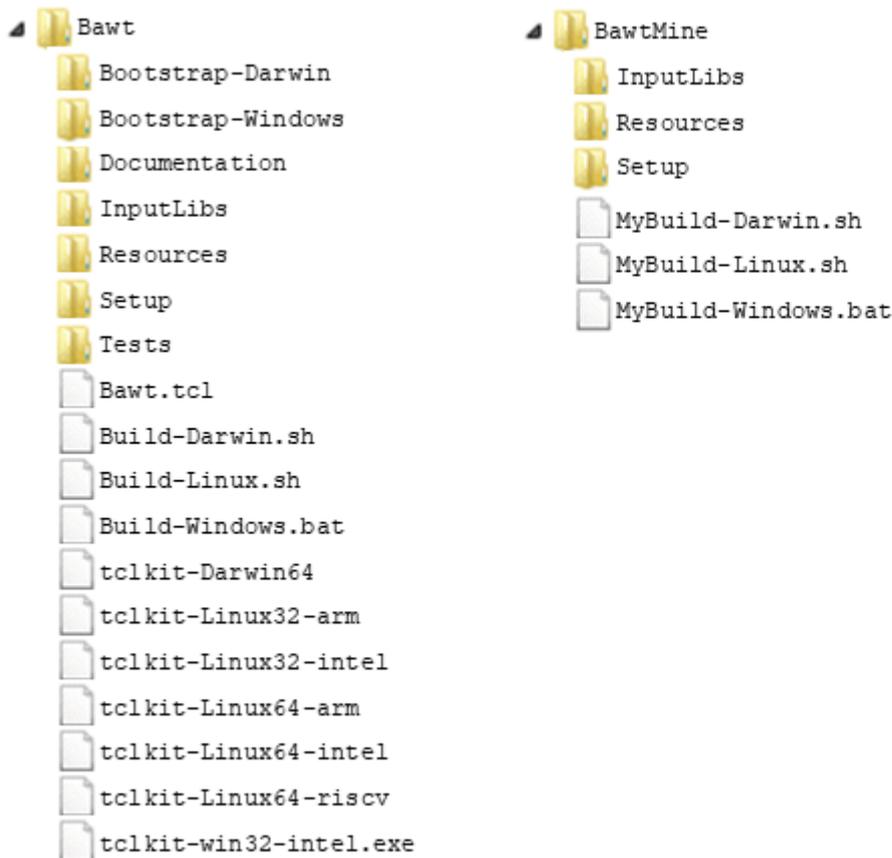
Another option is to use command line option `--noversion`, which strips the version number from the names of Tcl package directories.

5.2.2 Add a library

Library sources should be specified either as a directory named *\$libName-\$libVersion* or as a compressed file named *\$libName-\$libVersion.7z*.

libName must not contain a "-" character, because this character is used to separate the library name from the version string.

It is easily possible to extend the libraries compiled by **BAWT** with COTS software, ex. company specific libraries. One possibility is to just add these libraries into the *InputLibs* directory of the standard **BAWT** distribution. The better solution is to create a separate directory (ex. *BawtMine*), which holds your libraries in a similar structure like **BAWT** does. In this directory you create adapted versions of the batch scripts (ex. *MyBuild-Windows.bat*) and add *Setup* files, which reference your libraries as well as libraries of the standard BAWT distribution.



If you want to use a library, which is currently under development, it is possible to add the directory containing the local checkout of the library.

The following example shows the *Setup* file *mawtSvn.bawt* used to compile the current version of **MAWT** from my local SVN checkout.

```

Include "Tools.bawt"
Include "BasicLibs.bawt"
Include "Tcl_Basic.bawt"

if { [IsWindows] } {
    set dirName C:/poSoft/Mawt
} elseif { [IsLinux] } {
    set dirName /home/obermeier/poSoft/Mawt
} else {
    set dirName /Users/obermeier/poSoft/Mawt
}

Setup mawt $dirName mawt.bawt Version=0.4.0

```

Note, that the checkout directory typically has no version number in it, so the version number is specified as optional argument of the Setup procedure.

5.2.3 Add a Tcl program

Adding a Tcl program is similar to adding a library, i.e. the sources must be supplied as a compressed file as well as a corresponding *Build* script.

The Tcl program will be created as a starpack, i.e. a standalone executable containing the Tcl interpreter (tclkit), the program scripts as well as needed Tcl packages.

To ease the generation of starpacks, the BAWT framework offers procedures *MakeStarpackTcl* and *MakeStarpackTk* for this purpose. Use *MakeStarpackTcl*, if you want to create a console program, and *MakeStarpackTk*, if you want to create a program with a graphical Tk user interface.

```
proc MakeStarpackTcl { appScript appName starpackName buildDir args }
```

<i>appScript</i>	Full path to the startup script of the Tcl program.
<i>appName</i>	The name of the application. Typically <code>\$libName</code> .
<i>starpackName</i>	The name of the starpack executable. Typically <code>\$libName[GetExeSuffix]</code> .
<i>buildDir</i>	The name of the output directory. Typically <code>\$instDir</code> .
<i>args</i>	A list of files and directories to be included in the starpack. The path names of the files and directories must be absolute paths. The files of the Tcl program are typically located in <code>\$buildDir</code> . Needed Tcl packages are located in <code>[GetDevTclLibDir]</code> .

Example Build files using these procedures are:

- *BawtLogViewer.bawt*
- *gorilla.bawt*
- *poApps.bawt*
- *tclsg.bawt*
- *tksqlite.bawt*

The signature of procedure *MakeStarpackTk* is identical to procedure *MakeStarpackTcl*.

A starpack on Darwin is a directory using the extension *.app*.

5.2.4 Manually compile a library

To configure and compile a library, the **BAWT** framework uses shell (**.sh*) or batch files (**.bat*). These batch files are created in the *Configure* and *Compile* phases and stored in the *Build* directory (or a suitable subdirectory like eg. *win*) of the library.

You can use these batch files to configure or compile a library manually. This is especially useful while developing the build file for a new **BAWT** library.

Before running one of the shell or batch files on the command line, you have to remove the last line of the script containing the `exit` command or replace the `exit` command with an `echo` command.

You can easily open a library specific DOS or MSys shell window via the context menu of the *BawtLogViewer*, see chapter 6.1 Graphical Log Viewer.

The first part of the file name defines the configure and compile environment and corresponds to the general **BAWT** procedures for executing commands with the same name:

`_Bawt_DosRun:`

- The commands will be executed in a standard Windows command line environment.
- If running the command manually on Windows, it must be executed from a DOS command shell.
- **Example:** `> _Bawt_DosRun_CMakeBuild.bat`

`_Bawt_MSysRun:`

- The commands will be executed in the MSYS/MinGW environment or a standard shell environment on Unix systems.
- If running the command manually on Windows, it must be executed from a MSYS/MinGW shell.
- Note, that on Unix systems all files are prefixed with `_Bawt_MSys`.
- **Example:** `> sh _Bawt_MSysRun_MSysBuild.bat`

The second part specifies the caller of the `DosRun` or `MSysRun` command. This is typically one of the following standard BAWT procedures:

- `NMakeBuild`
- `MsBuild`
- `CMakeConfig`
- `CMakeBuild`
- `MSysConfig`
- `TeaConfig`
- `MSysBuild`

For libraries, which cannot be built with one of the above standard procedures, it is common usage to specify the caller in the form:

- `_Bawt_LibName_Configure`
- `_Bawt_LibName_Compile`

One example is the Boost library, which has special configure and compile commands:

- `_Bawt_DosRun_Boost_Configure.bat`
- `_Bawt_DosRun_Boost_Compile.bat`

When using `NMakeBuild` or `MsBuild`, there is no need to specify commands for the configuration phase.

- `_Bawt_DosRun_MsBuild.bat`
- `_Bawt_DosRun_NMakeBuild.bat`

All other commands typically come in pairs, so you will see the following combination of configure and compile batch scripts:

- `_Bawt_DosRun_CMakeConfig.bat`
- `_Bawt_DosRun_CMakeBuild.bat`
- `_Bawt_MSysRun_TeaConfig.bat`
- `_Bawt_MSysRun_MSysBuild.bat`
- `_Bawt_MSysRun_MSysConfig.bat`
- `_Bawt_MSysRun_MSysBuild.bat`
- `_Bawt_MSysRun_CMakeBuild.bat`
- `_Bawt_MSysRun_CMakeConfig.bat`

5.3 Known issues

5.3.1 Build deadlock

Problem:

The build process does not continue with specific libraries.

Workaround or solution:

This is due to errors in the build infrastructure of the corresponding library in conjunction with parallel builds. See chapter [5.1.6 Sign executables](#)

Signing executables is currently only available on Windows and needs program `signtool.exe`, which is part of Visual Studio.

Signing of executables is done by calling procedure `SignExecutables` in a build script. This call is typically done in the Distribution stage. See build file `BawtLogViewer.bawt` for an example.

There are two command line options controlling the signature process:

```
--certfile           (mandatory)
--timestampurl      (optional)
```

Directory `Resources` contains a self-signed certification file `poSoft.cer`, which is used for the BAWT distributions.

Parallel builds for details.

5.3.2 BawtLogViewer shows incorrect build time

Problem:

If the build of a library starts before midnight and extends over midnight, the build time of this package will be negative in the BawtLogViewer table display, as the log file only stores time values as HH:MM:SS.

Workaround or solution:

None.

5.3.3 Package SWIG

Problem:

SWIG build fails occasionally on Windows due to problems renaming files. This behavior was noticed on systems running Sophos AntiVirus only.

Workaround or solution:

No real solution, other than retrying the build until it succeeds.

5.3.4 Package Trf

Problem:

The CRC module of Tcl package `Trf` crashes when compiled in x86 mode on Windows.

Workaround or solution:

None.

5.3.5 Package tcllib/crc32

Problem:

The `crc32` module of Tcl package `tcllib` crashes when compiled in x86 mode on Windows.

Workaround or solution:

The crash is not the fault of module `crc32` itself, but of the CRC module of package `Trf`, which gets called, if the `Trf` extension is available.

Either remove package `Trf` or remove loading of accelerator `trf` in file `crc32.tcl`

```
foreach e {trf critcl} {
    if {[LoadAccelerator $e]} break
}
```

5.4 Tips and Tricks

5.4.1 Tips for Windows

Check generated library

To check the architecture of a generated dynamic library, execute the following command in a Visual Studio developer command prompt:

```
> dumpbin /headers XXX.dll | more
```

The architecture of the library is contained in the file header section of the output:

```
FILE HEADER VALUES
      machine (x64)
```

5.4.2 Tips for Linux

Check generated library

To check, if a library has been stripped, the commands `nm` or `file` can be used. To check the architecture of a generated library, the command `file` can be used.

A library built for Release should have no symbols and thus should generate the following outputs:

```
> nm libjpeg.so.9.1.0
nm: libjpeg.so.9.1.0: no symbols

> file libjpeg.so.9.1.0
libjpeg.so.9.1.0: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV),
dynamically linked, stripped
```

A Debug build should have symbols and thus should generate the following outputs:

```
> nm libjpeg.so.9.1.0 | more
0002ffa0 r aanscalefactor.4133
0002fa60 r aanscalefactor.4178
0002ffe0 r aanscales.4125

> file libjpeg.so.9.1.0
libjpeg.so.9.1.0: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV),
dynamically linked, not stripped
```

5.5 Advanced Batch Scripts

This section contains the batch scripts, which are used to generate the **Tcl-Pure** (minimal Tcl/Tk distribution) as well as the **Tcl-BI** (Batteries Included Tcl/Tk distribution) distributions.

5.5.1 Build Tcl-Pure distributions

The following batch scripts are used to create the **Tcl-Pure** distributions for all supported Tcl versions. A separate directory (`C:/BawtBuilds/TclMinimal/TclMinimal-%TCLVERS%-%TkVERS%-%LINKTYPE%`) is created for each Tcl version containing both the x86 and x64 versions.

The needed MSYS/MinGW versions are located in directory `C:/BawtBuildTools` (using option `--toolsdir`) to avoid extracting these for each Tcl version.

Batch script `UpdateTclMinimal.bat`

```
@echo off
setlocal

rem Architecture, TclVersion, TkVersion, TclString, LinkType and Finalize flag are
rem mandatory parameters
if "%1" == "" goto ERROR
if "%2" == "" goto ERROR
if "%3" == "" goto ERROR
if "%4" == "" goto ERROR
if "%5" == "" goto ERROR
if "%6" == "" goto ERROR

set ARCH=%1
set TCLVERS=%2
set TKVERS=%3
set TCLSTRING=%4
set LINKTYPE=%5
set FINALIZE=%6
shift
shift
shift
shift
shift
shift

rem If no target is given, use target "all".
if "%1"=="" goto BUILDALL

rem Loop through the rest of the parameter list for targets.
set TARGETS=
:PARAMLOOP
rem There is a trailing space in the next line. It's there for formatting.
set TARGETS=%TARGETS% %1
shift
if not "%1"=="" goto PARAMLOOP
goto BUILD

:BUILDALL
set TARGETS=all

:BUILD

set LINKTYPEOPT=
if "%LINKTYPE%"=="Dynamic" goto NOSTATIC
set LINKTYPEOPT=--copt Tcl Static=ON --copt Tk Static=ON
:NOSTATIC

set BAWTROOT=..\Bawt
```

```

set SETUPFILE=%BAWTROOT%\Setup\Tcl_MinimalDist.bawt
set FINALIZEFILE=Setup\UserFinalize.bawt
set OUTROOTDIR=C:/BawtBuilds/TclMinimal/TclMinimal-%TCLVERS%-%TKVERS%-%LINKTYPE%
set TOOLSDIR=C:/BawtBuildTools
set TCLKIT=%BAWTROOT%\tclkit-win32-intel.exe
set NUMJOBS=%NUMBER_OF_PROCESSORS%
set ACTION=--update

set BAWTOPTS=--rootdir %OUTROOTDIR% ^
             --toolsdir %TOOLSDIR% ^
             --architecture %ARCH% ^
             --compiler gcc+vs2022 ^
             --numjobs %NUMJOBS% ^
             --noonline ^
             --nouserbuilds ^
             --iconfile poSoft.ico ^
             --resourcefile poSoft.rc ^
             --certfile poSoft.cer ^
             --tclversion %TCLVERS% ^
             --tkversion %TKVERS% ^
             --copt SetupTcl "Version=%TCLSTRING%"

set FINALIZEOPT=--logviewer
if "%FINALIZE%"=="0" goto NOFINALIZE
set FINALIZEOPT=--finalizefile %FINALIZEFILE%
:NOFINALIZE

rem Build all libraries as listed in build configuration file.
CALL %TCLKIT% %BAWTROOT%\Bawt.tcl %BAWTOPTS% %LINKTYPEOPT% %FINALIZEOPT% %ACTION%
%SETUPFILE% %TARGETS%

goto EOF

:ERROR
echo.
echo Usage: %0 Architecture TclVersion TkVersion TclString UseFinalizeScript [Target1]
[TargetN]
echo Architecture      : x86 x64
echo TclVersion        : 8.6.17  9.0.2
echo TkVersion         : 8.6.17  9.0.2
echo TclString         : 8.6.17.0 9.0.2.0
echo LinkType          : Static Dynamic
echo UseFinalizeScript: 0 1
echo Default target   : all
echo.

:EOF

```

You might need to adapt the pathes specified in `OUTROOTDIR` and `TOOLSDIR` as well as the used Visual Studio version specified in command line option `--compiler`.

Batch script `UpdateTclMinimals.bat`

```

@echo off
setlocal

REM Architecture TclVersion TkVersion TclString LinkType UseFinalizeScript

CALL UpdateTclMinimal x64 8.6.17 8.6.17 8.6.17.0 Dynamic 0
CALL UpdateTclMinimal x64 9.0.2 9.0.2 9.0.2.0 Dynamic 0

CALL UpdateTclMinimal x86 8.6.17 8.6.17 8.6.17.0 Dynamic 0
CALL UpdateTclMinimal x86 9.0.2 9.0.2 9.0.2.0 Dynamic 0

```

CALL	UpdateTclMinimal	x64	9.0.2	9.0.2	9.0.2.0	Static	0
CALL	UpdateTclMinimal	x86	9.0.2	9.0.2	9.0.2.0	Static	0

5.5.2 Build Tcl-BI distributions

The following batch scripts are used to create the **Tcl-BI** distributions for all supported Tcl versions. A separate directory (*C:/BawtBuilds/TclDistribution/TclDistribution-%TCLVERS%*) is created for each Tcl version containing both the x86 and x64 versions.

The needed MSYS/MinGW versions are located in directory *C:/BawtBuildTools* (using option `--toolsdir`) to avoid extracting these for each Tcl version.

Batch script *UpdateTclDistribution.bat*

```
@echo off
setlocal

rem Architecture, TclVersion, TkVersion, TclString, ImgVersion, Action and Finalize flag
rem are mandatory parameters
if "%1" == "" goto ERROR
if "%2" == "" goto ERROR
if "%3" == "" goto ERROR
if "%4" == "" goto ERROR
if "%5" == "" goto ERROR
if "%6" == "" goto ERROR
if "%7" == "" goto ERROR

set ARCH=%1
set TCLVERS=%2
set TKVERS=%3
set TCLSTRING=%4
set IMGVERS=%5
set ACTION=%6
set FINALIZE=%7
shift
shift
shift
shift
shift
shift
shift

rem If no target is given, use target "all".
if "%1"==" " goto BUILDALL

rem Loop through the rest of the parameter list for targets.
set TARGETS=
:PARAMLOOP
rem There is a trailing space in the next line. It's there for formatting.
set TARGETS=%TARGETS% %1
shift
if not "%1"==" " goto PARAMLOOP
goto BUILD

:BUILDALL
set TARGETS=all

:BUILD

set BAWTROOT=..\Bawt
set SETUPFILE=%BAWTROOT%\Setup\Tcl_Distribution.bawt
set FINALIZEFILE=Setup\UserFinalize.bawt
set OUTROOTDIR=C:/BawtBuilds/TclDistribution/TclDistribution-%TCLVERS%-%TKVERS%
set TOOLSDIR=C:/BawtBuildTools
set TCLKIT=%BAWTROOT%\tclkit-win32-intel.exe
set NUMJOBS=%NUMBER_OF_PROCESSORS%
```

```

set ACTION=--%ACTION%

set BAWTOPTS=--rootdir %OUTROOTDIR% ^
             --toolsdir %TOOLSDIR% ^
             --architecture %ARCH% ^
             --compiler gcc+vs2022 ^
             --numjobs %NUMJOBS% ^
             --noonline ^
             --nouserbuilds ^
             --iconfile poSoft.ico ^
             --resourcefile poSoft.rc ^
             --certfile poSoft.cer ^
             --tclversion %TCLVERS% ^
             --tkversion %TKVERS% ^
             --imgversion %IMGVERS% ^
             --copt SWIG "AddTcl=ON" ^
             --copt SetupTcl "Tag=-BI" ^
             --copt SetupTcl "Version=%TCLSTRING%"

set FINALIZEOPT=--logviewer
if "%FINALIZE%"=="0" goto NOFINALIZE
set FINALIZEOPT=--finalizefile %FINALIZEFILE%
:NOFINALIZE

rem Build all libraries as listed in build configuration file.
CALL %TCLKIT% %BAWTROOT%\Bawt.tcl %BAWTOPTS% %FINALIZEOPT% %ACTION% %SETUPFILE%
%TARGETS%

goto EOF

:ERROR
echo.
echo Usage: %0 Architecture TclVersion TkVersion TclString ImgVersion Action
UseFinalizeScript [Target1] [TargetN]
echo Architecture      : x86 x64
echo TclVersion        : 8.6.17  9.0.2
echo TkVersion         : 8.6.17  9.0.2
echo TclString         : 8.6.17.0 9.0.2.0
echo ImgVersion        : 1.4.17 2.1.0
echo Action            : update test
echo UseFinalizeScript: 0 1
echo Default target    : all
echo.

:EOF

```

You might need to adapt the pathes specified in `OUTROOTDIR` and `TOOLSDIR` as well as the used Visual Studio version specified in command line option `--compiler`.

Batch script `UpdateTclDistributions.bat`

```

@echo off
setlocal

REM Architecture TclVersion TkVersion TclString ImgVersion Action UseFinalizeScript

CALL UpdateTclDistribution x64 8.6.17 8.6.17 8.6.17.0 2.1.0 update 0
CALL UpdateTclDistribution x86 8.6.17 8.6.17 8.6.17.0 2.1.0 update 0

CALL UpdateTclDistribution x64 9.0.2 9.0.2 9.0.2.0 2.1.0 update 0
CALL UpdateTclDistribution x86 9.0.2 9.0.2 9.0.2.0 2.1.0 update 0

```

6 Logging

The *Logs* output directory contains the overall build log file *_BawtBuild.log* as well as the library specific build log files.

Library specific log files contain the output of the configuration and compile process. They also contain the error messages, if the build of a library does not succeed.

The overall log file contains the messages, which are printed onto standard output during the BAWT build process. The amount of log messages can be set by specifying the log level with command line option [--loglevel](#). Level 0 does not produce any log messages, while level 4 produces lots of log messages. The default value for the log level is 3.

Each stage or executed command is prefixed with a time code like shown in the next line:

```
21:35:30 > Build tclcompiler 1.7.1 (Release)
```

If log files of different configurations should be compared, these time codes may be disturbing. BAWT therefore allows to remove the time codes from the log messages by specifying command line option [-nologtime](#).

When rerunning a build, existing log files are renamed by appending *.bak* to the corresponding files before creating the new log files.

To view the build process online in a graphical window, the command line option [--logviewer](#) can be specified. See the next chapter for a detailed description of the graphical log file viewer **BawtLogViewer**.

Logging functionality is realized in namespace `BawtLog`. The most important procedure is `Log`, which may be used in build scripts, too.

Command line options influencing logging:

[--loglevel](#)
[--nologtime](#)
[--logviewer](#)

6.1 Graphical Log Viewer

The **BawtLogViewer** is a separate program to view and analyse the log output of BAWT. It is a Tcl script, which is wrapped as a Starpack and is included as a Windows executable in directory *Bootstrap-Windows*. For other platforms it can be built with BAWT.

To generate log messages usable by **BawtLogViewer**, the log level must be set to at least 3.

The graphical log viewer can either be used to analyse log files after a build process has finished (offline mode) or it can be used to interactively view the build process (online mode). Viewing the log messages online can be done by either using command line option [--logviewer](#) when starting the BAWT build process or by opening the log file *_BawtBuild.log* anytime during the build process.

Log files can be opened by using the `File` menu or by dragging and dropping the icon of the log file onto the **BawtLogViewer** window.

The following figure shows the layout of the log viewer window, which has 2 main parts. In the upper part all libraries of the Setup file are listed in a scrollable table, while in the lower part the log messages of the build process are displayed in a scrollable text widget.

#	Build-#	Library Name	User file	Version	Compiler	Exe. time	Est. time	Mod. time	Update cause	Stages	User config
1	1	InnoSetup		6.2.2		0.03	0.02	2025-10-03 00:01:49	Build directory not existent		
2	2	Tcl		9.0.2	gcc	4.62	4.00	2025-10-03 00:06:26	Build directory not existent		
3	3	Tclapplescript		2.2					Excluded from build (Darwin only)		
4	4	tcllib	Yes	2.1	gcc	0.64	0.46	2025-10-03 00:07:05	Build directory not existent		
5	5	TclStubs		9.0.2	vs2022	1.59	0.94	2025-10-03 00:08:40	Build directory not existent		
6	6	Tk		9.0.2	gcc	0.50	1.08	2025-10-03 00:08:56	Build directory not existent		
7	7	TkStubs		9.0.2	vs2022		0.29	2025-10-02 23:05:27			
8	8	udp		1.0.12	gcc		0.30				
9	9	ntext		0.1.1	vs2022		0.22	2025-10-02 23:05:40			
10	10	SetupTcl					1.38	2025-10-02 23:06:10			Tags: Doc Version=9.0.2.0

```

Log file C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Log/_BawtBuild.log (0.17 seconds to read)
Source directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Install/tcllib/lib
Target directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Distribution/opt/Tcl/lib
File pattern : *
Number of copied files: 732
00:07:04 > FileRename
Source: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Development/opt/Tcl/lib/tcllib2.1
Target: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Development/opt/Tcl/lib/tcllib
00:07:05 > FileRename
Source: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Distribution/opt/Tcl/lib/tcllib2.1
Target: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Distribution/opt/Tcl/lib/tcllib
WriteConsoleFile
File: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/tcllib/_Bawt_StartMSysConsole.bat
00:07:05 > End tcllib 2.1: 0.64 minutes

00:07:05 > Start TclStubs 9.0.2 (Library #5 of 10)
Build types : Release
Update cause: Build directory not existent
00:07:05 > Clean TclStubs (Release)
00:07:05 > DirDelete
Directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/TclStubs
00:07:05 > DirDelete
Directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Install/TclStubs
00:07:05 > Build TclStubs 9.0.2 (Release)
DirCreate
Directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/TclStubs
DirCreate
Directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Install/TclStubs
00:07:05 > ExtractLibrary
ZIP file : C:/poSoft/Bawt/InputLibs/Tcl-9.0.2.7z
Target directory: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/TclStubs
00:07:08 > FileRename
Source: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/Tcl-9.0.2
Target: C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/TclStubs
Auto Update: ON (Library Tk running since 0.50 minutes. Estimated build time 1.08 minutes)
    
```

Different row background colors indicate the build status of a library. A green background indicates a successful build of a library, a blue background indicates an excluded library, a yellow background shows the library currently under build and an orange background indicates a library, where the current build time is greater than the estimated build time. See below for an explanation of estimated build times for deadlock detection.

A red text color is displayed for libraries which issued a warning during the build process.

The table can be sorted by any of the columns except the first one, which just shows the row number. For example, you may want to view the libraries sorted by library names instead of the build number. Selecting a table row scrolls to the beginning of the corresponding section in the text widget. The section is also marked with a yellow background.

By double clicking onto a table row, a simple editor window is opened showing the contents of the library specific build log file, see next figure for an example. Your favourite editor may be specified by setting the environment variable `EDITOR`.

```

Library Build Log: Tk.log (0 warnings)
C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Logs/Tk.log
> Start Build_Tk

~ /c/Users/anton
configure: loading site script /etc/config.site
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.exe
checking for suffix of executables... .exe
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether the compiler supports GNU C... yes
checking whether gcc accepts -g... yes
checking for gcc option to enable C11 features... none needed
checking for inline... inline
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for strings.h... yes
checking for sys/stat.h... yes
checking for sys/types.h... yes
checking forunistd.h... yes
checking for ar... ar
checking for ranlib... ranlib
checking for windres... windres
checking whether make sets $(MAKE)... yes
checking how to build libraries... shared
checking for Tcl configuration... found /C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/Tcl/tclConf
checking for existence of /C:/BawtBuilds/DocSetup/DocSetup-9.0.2/vs2022/x64/Release/Build/Tcl/tclConf

```

Pressing the right mouse button opens a context menu with the following functionalities:

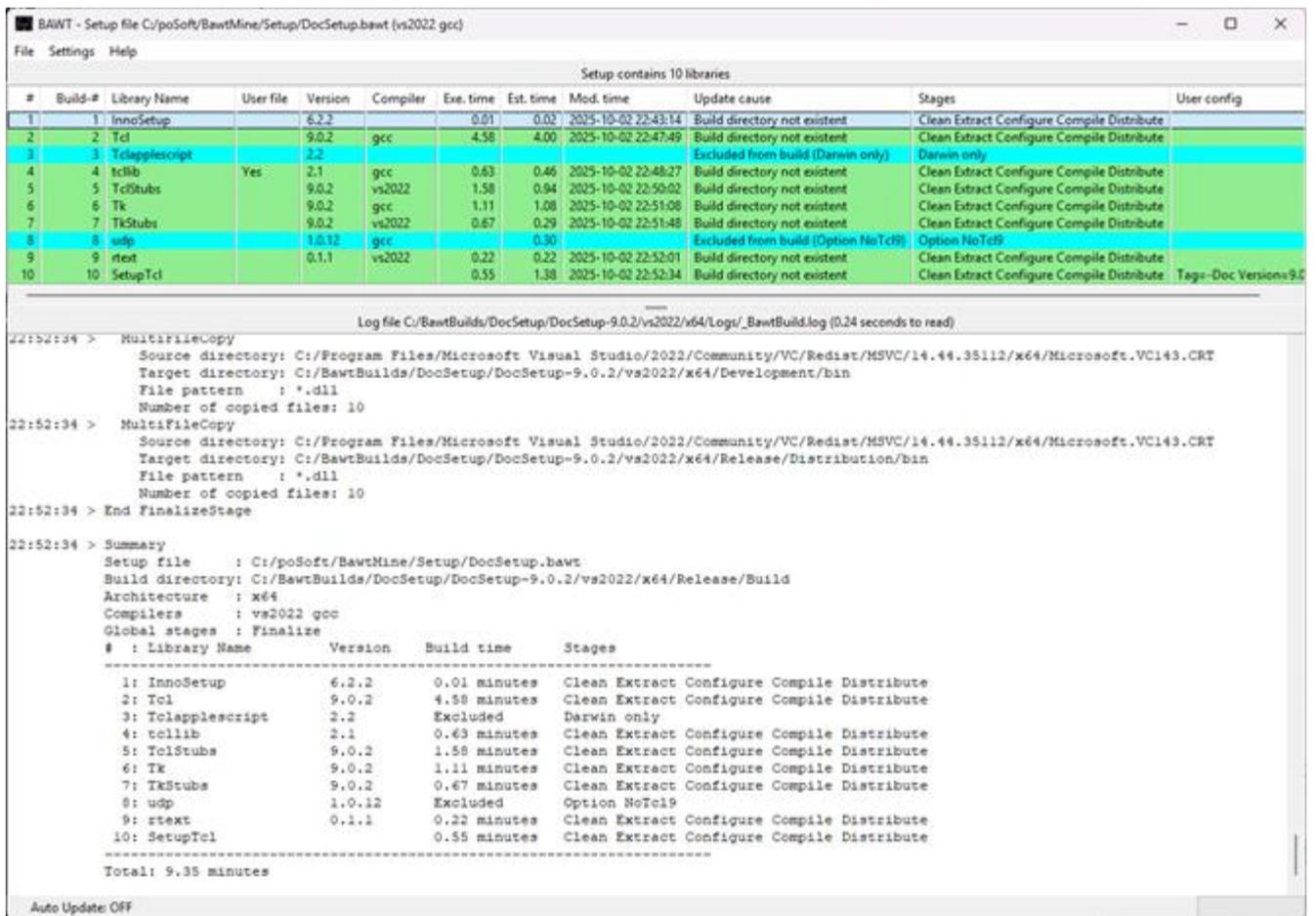
- Save library specific build or test time.
- Open library specific directories in an Explorer window.
- Open library specific Log, Setup or Build file.
- Open library specific DOS or MSYS shell window.

Pressing a key while the table has focus, selects the next library, which has this key as its first letter. Pressing other keys within the *Key Repeat Time* extends the search string similar to the behaviour of the Windows Explorer. The *Key Repeat Time* can be specified in the *Settings* menu.

Pressing the `Return` key has the same functionality as double clicking the selected table row.

Note the following features, which are only available in online mode:

- **BawtLogViewer** starts in *Auto Update* mode, where it reloads the log file every 3 seconds. The *Auto Update* mode is automatically switched off when the end of the build process is detected in the log file or it can be switched on or off by selecting the appropriate entry in the *File* menu.
- When reloading the log file, the table row order is always reset to the library build order.
- The accumulated time of the library currently being built is displayed in the status bar of the viewer window and in the corresponding table cell.
- Column *Stages* is not filled before the end of the build process, see next figure.



The program can be used to detect library build deadlocks by comparing the current build time against an estimated build time. To generate estimated build times, at least one BAWT build has to be performed. After loading the corresponding log files, the build times of this run can be saved in the settings file by selecting File menu entry Save build times.

These build times are then used as estimated build times in future BAWT builds to compare the current build time of a library against these estimated build times. If the current build time exceeds the estimated time by a specific threshold value (which can be specified in the Settings menu), both a visual warning (corresponding row background is set to orange) as well as an acoustic warning (beep) is issued. The acoustic warning can be disabled in the Settings menu.

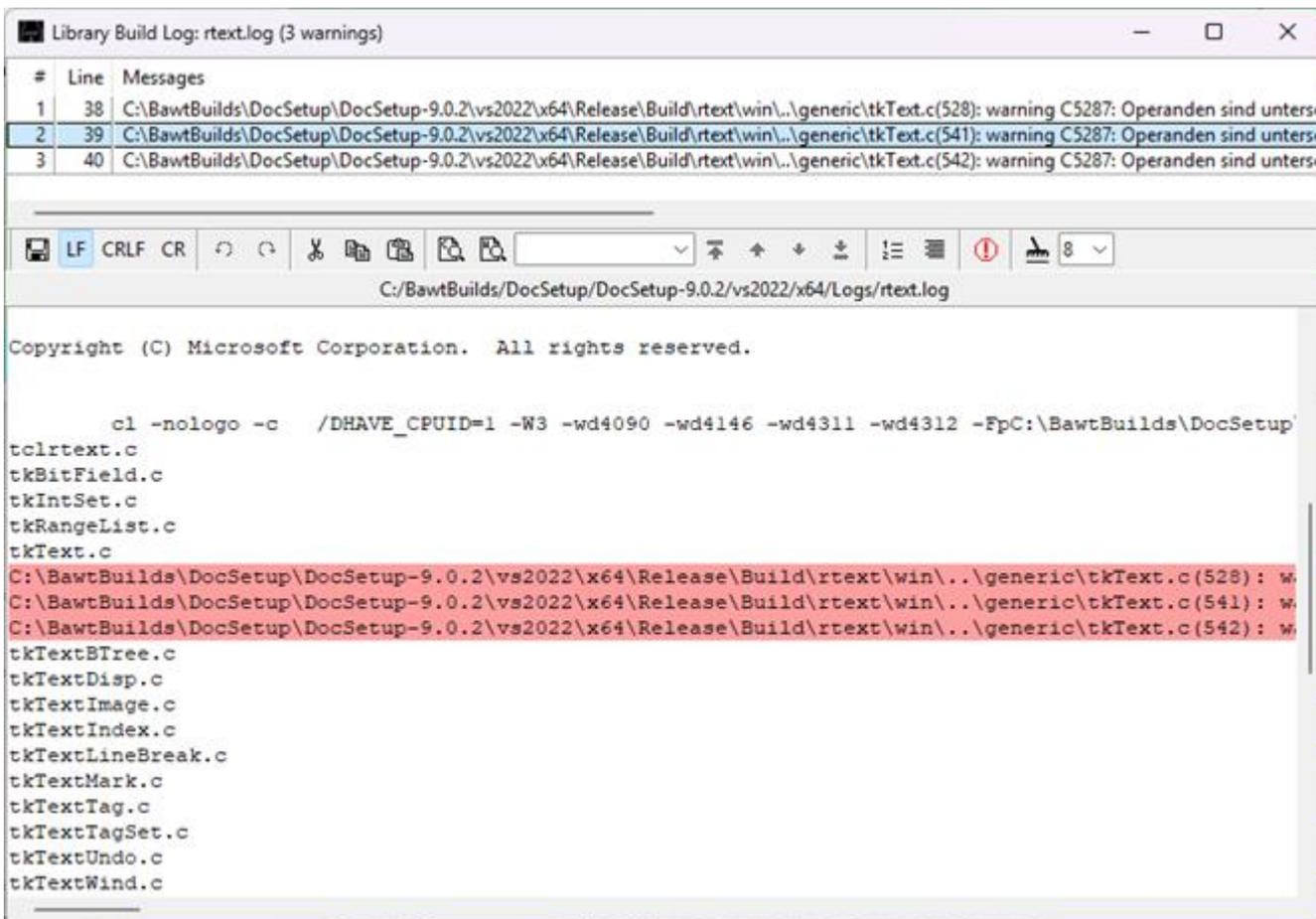
Estimated build times, deadlock parameters and other values like window size and position are stored in the settings files `~/BawtLogViewer/BawtLogViewer.cfg`.

Warnings occurring during the build process of libraries are logged by the BAWT framework and can be inspected by selecting File menu entry View compiler warnings.

#	Library Name	Warnings
1	Tcl	22
2	rtext	3
3	InnoSetup	0
4	tcllib	0
5	TclStubs	0
6	Tk	0
7	TkStubs	0
8	SetupTcl	0
9	Tclapplescript	-1
10	udp	-1

The table can be sorted and searched using key presses as described for the main window. Double clicking a row or pressing the `Return` key opens a simple editor window showing the contents of the library specific build log file.

If warnings occurred during the build process, these are shown in a table in the upper part of the editor window. Clicking onto a row selects the corresponding message in the editor window, see next figure.



The following glob-style patterns are used to detect errors and warnings in the log messages:

#	gcc	VisualStudio	gcc-German
SetWarningDetectPatterns	"*: warning:*"	"*: warning*:*"	"*: Warnung:*"
SetErrorDetectPatterns	"*: error:*"	"*: error C*:*"	"*: Fehler:*"

This list of patterns can be extended by editing the settings file `~/BawtLogViewer.cfg`.

```
# SetWarningDetectPatterns args
catch {::BawtLogViewer::SetWarningDetectPatterns {*: warning:*} {*: warning*:*} {*: Warnung:*}}

# SetErrorDetectPatterns args
catch {::BawtLogViewer::SetErrorDetectPatterns {*: error:*} {*: error C*:*} {*: Fehler:*}}
```

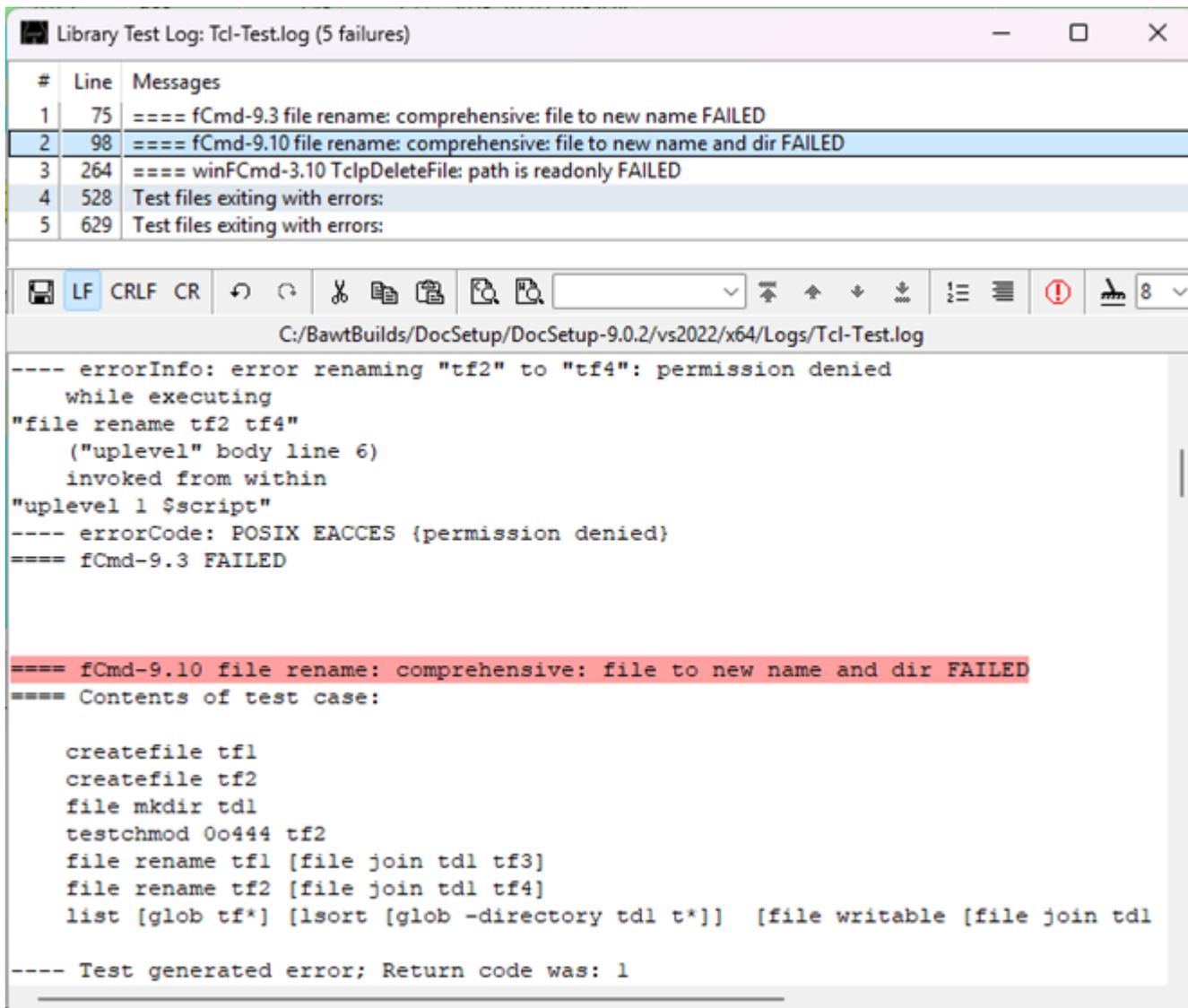
Failures occurring during the test process of libraries are logged by the BAWT framework and can be inspected by selecting `File` menu entry `View test failures`. See the section regarding test procedures in chapter 0 [Build Files](#) for more information about test execution.

-1 means no tests performed		
#	Library Name	Failures
1	Tcl	5
2	Tk	0
3	rtext	0
4	InnoSetup	-1
5	Tclapplescript	-1
6	tcllib	-1
7	TclStubs	-1
8	TkStubs	-1
9	udp	-1
10	SetupTcl	-1

A green background indicates an executed test, while a blue background indicates, that the library does not have a test procedure. Column `Failures` shows the number of failures during test execution of a library.

The table can be sorted and searched using key presses as described for the main window. Double clicking a row or pressing the `Return` key opens a simple editor window showing the contents of the library specific test log file.

If failures occurred during the test process, these are shown in a table in the upper part of the editor window. Clicking onto a row selects the corresponding message in the editor window, see next figure.



Matching test failures in the test log files uses `tcltest` specific strings, as well as user defined glob-style patterns similar to detecting build warnings.

The following user defined glob-style patterns are used to detect failures in the test log messages.

```
# SetFailureDetectPatterns args
catch {::BawtLogViewer::SetFailureDetectPatterns
  Error:* ERROR:* Test*failed:* ErrorCode*}
```

This list of patterns can be extended by editing settings file `~/BawtLogViewer.cfg`.

7 Command Line Options

Calling the **BAWT** framework script with command line option `--help` prints the following help message:

```
Usage: Bawt.tcl [Options] SetupFile LibraryName [LibraryNameN]
```

Start the BAWT automatic library build process.

When using "all" as library name, all libraries specified in the setup file are built.

It is also possible to specify the numbers of the libraries as printed by option "--list" or specify a range of numbers (ex: 2-5).

Note, that at least either a list or build action option must be specified.

7.1 General Options

Option	Description
<code>--help</code>	Print this help message and exit.
<code>--version</code>	Print BAWT version and copyright and exit. Use in combination with <code>--loglevel 0</code> to just print the version number.
<code>--procs</code>	Print all available procedures and exit.
<code>--proc <str></code>	Print documentation of specified procedure and exit.
<code>--loglevel <int></code>	Specify log message verbosity. Choices: 0 - 4. Default: 3.
<code>--nologtime</code>	Do not write time strings with log messages. Default: Write time strings. Use this option when comparing log files to have less differences.
<code>--logviewer</code>	Start graphical log viewer program BawtLogViewer . Only working, if log level is greater than 1. Default: No.

7.2 List Action Options

Option	Description
<code>--list</code>	Print all available library names and versions and exit.
<code>--platforms</code>	Print library names, versions and supported platforms.
<code>--wincompilers</code>	Print library names, versions and supported Windows compilers.
<code>--authors</code>	Print library names, versions and script authors.
<code>--homepages</code>	Print library names, versions and homepages.
<code>--excludes</code>	Print library names, versions and platform specific excluded libraries.
<code>--options</code>	Print library names, versions and library build options.
<code>--havetest</code>	Print library names, versions and test procedure availability.
<code>--dependencies</code>	Print library names, versions and dependencies.
<code>--dependency</code>	Print dependencies of specified target libraries.

The list action options may be accumulated to print several library informations at once.

7.3 Build Action Options

Option	Description
--clean	Clean library specific build and install directories.
--extract	Extract library source from a ZIP file or a directory.
--configure	Perform the configure stage of the build process.
--compile	Perform the compile stage of the build process.
--distribute	Perform the distribution stage of the build process.
--finalize	Generate environment file and call user supplied Finalize procedure.
--complete	Perform the following stages in order: clean, extract, configure, compile, distribute, finalize.
--update	Perform necessary stages depending on modification times. Note: Global stage finalize is always executed.
--simulate	Simulate update action without actually building libraries.
--touch	Set modification times of library build directories to current time.
--test	Perform library specific test scripts.

7.4 Build Configuration Options

Option	Description
--architecture <str>	Build for specified processor architecture. Choices: x86 for 32-bit, x64 for 64-bit. Default: Architecture of the calling tclkit or tclsh.
--compiler <str>	Build with specified compiler version. Choices: gcc vs2013 vs2015 vs2017 vs2019 vs2022. Specify primary and secondary compiler by adding a plus sign in between. Example: gcc+vs2022. Default: gcc.
--gccversion <str>	Build with specified MinGW gcc version. Windows only. Choices: 7.2.0 8.1.0 11.2.0 12.2.0 13.2.0 14.2.0. Default: 7.2.0.
--msysversion <str>	Build with specified MSYS version. Windows only. Choices: 1 2. Default: Version 2 if available, otherwise version 1.
--tclversion <str>	Build Tcl, Tk and Tclkit for specified version. Choices: 8.6.7 - 8.6.17, 9.0.1 - 9.0.2. Default: 8.6.17.
--tkversion <str>	Build Tk and Tclkit for specified version. Choices: 8.6.7 - 8.6.17, 9.0.1 - 9.0.2. Default: 8.6.17.
--imgversion <str>	Build Img for specified version. Choices: 1.4.13 - 1.4.17, 2.0.1 - 2.1.0. Default: 2.1.0.
--tclsversion <str>	Build tcltcl for specified version. Choices: 1 2. Default: Version 1.
--osgversion <str>	Build OpenSceneGraph for specified version. Choices: 3.4.1 3.6.5. Default: 3.6.5.
--libversion <lib> <str>	Build library for specified version.

	Overwrites values specified in Setup file. Default: As specified in Setup file.
--zipfile <lib> <str>	Build library from specified file or directory. Overwrites values specified in Setup file. Default: As specified in Setup file.
--buildtype <str>	Use specified build type. Choices: Release Debug. Default: Release or as specified in setup file.
--universal	Enable universal binary builds. Available for Darwin only. Default: Not enabled.
--mindarwin <str>	Specify Darwin minimum version to build for. Default: 11.0 (Big Sur).
--sanitizer	Enable Address Sanitizer libasan. Only valid with build type Debug. Default: Not enabled.
--exclude <lib>	Force exclusion of build for specified library name.
--wincc <lib> <str>	Use specified Windows compiler, if supported by build script. Choices: gcc vs.
--sdk <lib> <str>	Use specified Microsoft SDK version. To use the SDK version for all libraries, specify "all" as library name.
--copt <lib> <str>	Specify library specific user configuration option.
--user <lib> <str>	Specify library specific user build file.
--url <str>	Specify BAWT download server. Default: https://www.tcl3d.org/bawt/download
--cacert <str>	Use specified certificate file as parameter to curl calls. Default: None.
--toolsdir <str>	Specify directory containing MSYS/MinGW. Default: [GetOutputToolsDir]
--rootdir <str>	Specify build output root directory. Default: [GetOutputRootDir]
--libdir <str>	Add a directory containing library source and build files. This option can be called multiple times and adds the new directory to the beginning of the directory list. Default search list: [file join [pwd] "InputLibs"] [file join [GetInputRootDir] "InputLibs"]
--distdir <str>	Specify distribution root directory. Default: [file join [GetOutputTypeDir] "Distribution"]
--finalizefile <str>	Specify file with user supplied Finalize procedure. Default: None.
--sort <str>	Sort libraries according to specified sorting mode. Choices: dependencies dictionary none. Default: dependencies
--noversion	Do not use version number for Tcl package directories. Default: Library name and version number.
--noexit	Do not exit build process after fatal error, but try to continue. Default: Exit build process after a fatal error.
--nosetupwarning	Do not print warnings regarding multiple versions in Setup files. Default: Print warnings.
--noimportlibs	Do not create import libraries on Windows. Default: Create import libraries. Needs VisualStudio.

<code>--noruntimelibs</code>	Do not copy VisualStudio runtime libraries. Default: Copy runtime libraries. Needs VisualStudio.
<code>--nostrip</code>	Do not strip libraries in distribution directory. Default: Strip libraries.
<code>--noonline</code>	Do not check or download from online repository. Default: Use https://www.tcl3d.org/bawt/download
<code>--norecursive</code>	Do not check recursive dependencies. Default: Use recursive dependencies.
<code>--nosubdirs</code>	Do not create compiler and architecture sub directories. Default: Create compiler and architecture sub directories.
<code>--nouserbuilds</code>	Do not consider user build files. Default: Consider user build files named <i>LibraryName_User.bawt</i> .
<code>--iconfile <str></code>	Use specified icon file for tclkits and starpacks. Default: Standard tclkit icon. Windows only.
<code>--resourcefile <str></code>	Use specified resource file for tclkits and starpacks. Default: Standard tclkit resource file. Windows only.
<code>--certfile <str></code>	Use specified certification file for code signing starpacks. Default: No code signing. Windows only.
<code>--timestampurl <str></code>	Use specified timestamp server for code signing starpacks. Default: http://timestamp.digicert.com . Use empty string to add no timestamp. Windows only.
<code>--numjobs <int></code>	Number of parallel compile jobs. Default: 1
<code>--libjobs <lib> <int></code>	Number of parallel compile jobs for specified library. Default: 1
<code>--timeout <float></code>	Number of seconds to try renaming or deleting directories. Default: 30.0

8 Supported Libraries

List of all libraries (using command line option [--platforms](#))

#:	Name	Version	Platforms		
1:	apave	4.4.10	Windows	Linux	Darwin
2:	argparse	0.61	Windows	Linux	Darwin
3:	awthemes	10.4.0	Windows	Linux	Darwin
4:	BawtLogViewer	3.2.0	Windows	Linux	Darwin
5:	Blender	3.0.0	Windows		
6:	Boost	1.78.0	Windows	Linux	Darwin
7:	BWidget	1.10.1	Windows	Linux	Darwin
8:	Cal3D	0.120	Windows	Linux	
9:	Canvas3d	1.2.3	Windows	Linux	
10:	cawt	3.1.1	Windows		
11:	cawtapp	3.1.1	Windows		
12:	ccl	4.0.6	Windows	Linux	
13:	cffi	2.0.3	Windows	Linux	Darwin
14:	cfitsio	4.1.0	Windows	Linux	Darwin
15:	CMake	3.25.2	Windows	Linux	Darwin
16:	critcl	3.3	Windows	Linux	Darwin
17:	curl	7.70.0	Windows	Linux	Darwin
18:	DiffUtil	0.4.3	Windows	Linux	Darwin
19:	DirectXTex	2021_11	Windows		
20:	Doxygen	1.8.15	Windows		
21:	Eigen	3.3.9	Windows	Linux	Darwin
22:	expect	5.45.4.1	Linux	Darwin	
23:	Ffidl	0.9.1	Windows	Linux	Darwin
24:	ffmpeg	7.1.2	Windows	Linux	Darwin
25:	fftw	3.3.9	Windows	Linux	Darwin
26:	fitsTcl	2.5.1	Windows	Linux	Darwin
27:	freeglut	3.2.2	Windows	Linux	Darwin
28:	Freetype	2.13.3	Windows	Linux	Darwin
29:	FTGL	2.1.3	Windows	Linux	Darwin
30:	gdal	2.4.4	Windows	Linux	Darwin
31:	gdi	0.9.9.15	Windows		
32:	GeographicLib	1.52	Windows	Linux	Darwin
33:	GeographicLibData		Windows	Linux	Darwin
34:	geos	3.7.2	Windows	Linux	Darwin
35:	giflib	5.2.1	Windows	Linux	Darwin
36:	Gl2ps	1.4.2	Windows	Linux	Darwin
37:	GLEW	2.2.0	Windows	Linux	Darwin
38:	glfw	3.3.8	Windows	Linux	Darwin
39:	gorilla	1.6.1	Windows	Linux	Darwin
40:	hdc	0.2.0.1	Windows		
41:	Img	2.1.0	Windows	Linux	Darwin
42:	imgjp2	0.1.1	Windows	Linux	Darwin
43:	imgtools	0.3.1	Windows	Linux	Darwin
44:	InnoSetup	6.2.2	Windows		
45:	iocp	2.0.2	Windows		
46:	itk	4.2.5	Windows	Linux	Darwin
47:	iwidgets	4.1.2	Windows	Linux	Darwin
48:	jasper	2.0.25	Windows	Linux	Darwin
49:	jigsaw	2.0	Windows	Linux	Darwin
50:	JPEG	9.f	Windows	Linux	Darwin
51:	KDIS	2.9.0	Windows	Linux	Darwin
52:	libffi	3.4.8	Windows	Linux	Darwin
53:	libgd	2.3.2	Windows	Linux	Darwin
54:	libressl	2.9.2	Windows	Linux	Darwin
55:	libwebp	1.2.4	Windows	Linux	Darwin
56:	libxml2	2.10.3	Windows	Linux	Darwin
57:	materialicons	0.2	Windows	Linux	Darwin
58:	mawt	0.5.0	Windows	Linux	Darwin
59:	memchan	2.3.1	Windows	Linux	Darwin
60:	mentry	4.5	Windows	Linux	Darwin
61:	Mpexpr	1.2.1	Windows	Linux	Darwin
62:	mqtt	4.0	Windows	Linux	Darwin

63:	mupdf	1.26.0	Windows	Linux	Darwin
64:	MuPDFWidget	2.4	Windows	Linux	Darwin
65:	nacl	1.1.1	Windows	Linux	Darwin
66:	nasm	2.16.3	Windows		
67:	nsf	2.4.0	Windows	Linux	Darwin
68:	OglInfo	1.0.1	Windows	Linux	Darwin
69:	ooxml	1.10	Windows	Linux	Darwin
70:	openjpeg	2.5.3	Windows	Linux	Darwin
71:	OpenSceneGraph	3.6.5	Windows	Linux	Darwin
72:	OpenSceneGraphData	3.4.0	Windows	Linux	Darwin
73:	orattcl	4.6.1	Windows	Linux	Darwin
74:	osgcal	0.2.1	Windows	Linux	
75:	osgearth	2.10.1	Windows	Linux	Darwin
76:	pandoc	3.5	Windows	Linux	Darwin
77:	parse_args	0.5.1	Windows	Linux	Darwin
78:	pawt	1.2.0	Windows	Linux	Darwin
79:	pdf4tcl	0.9.4	Windows	Linux	Darwin
80:	pgintcl	3.5.2	Windows	Linux	Darwin
81:	photoresize	0.2.1	Windows	Linux	Darwin
82:	pkgconfig	0.29.2	Darwin		
83:	PNG	1.6.48	Windows	Linux	Darwin
84:	poApps	3.2.0	Windows	Linux	Darwin
85:	poClipboardViewer	3.2.0	Windows		
86:	poImg	3.0.1	Windows	Linux	Darwin
87:	poLibs	3.2.0	Windows	Linux	Darwin
88:	poMemory	1.0.0	Windows	Linux	Darwin
89:	printer	0.9.6.16	Windows		
90:	publisher	2.0	Windows	Linux	Darwin
91:	puppyicons	0.1	Windows	Linux	Darwin
92:	Python	3.7.7	Windows		
93:	rbc	0.2.0	Windows	Linux	
94:	Redistributables		Windows		
95:	rl_json	0.16.0	Windows	Linux	Darwin
96:	rttext	0.1.1	Windows	Linux	Darwin
97:	ruff	2.6.0	Windows	Linux	Darwin
98:	scrollutil	2.6	Windows	Linux	Darwin
99:	SDL	2.26.2	Windows	Linux	Darwin
100:	SetupPython		Windows		
101:	SetupTcl		Windows	Linux	Darwin
102:	shellicon	0.2.0	Windows		
103:	Snack	2.2.12	Windows	Linux	
104:	sqlite3	3.47.1	Windows	Linux	Darwin
105:	SWIG	4.3.1	Windows	Linux	Darwin
106:	tablelist	7.7	Windows	Linux	Darwin
107:	tbcloud	2.0a0	Windows	Linux	Darwin
108:	Tcl	8.6.17	Windows	Linux	Darwin
109:	tcl3dExtended	1.0.1	Windows	Linux	Darwin
110:	tcl3dFull	1.0.1	Windows	Linux	Darwin
111:	tcl9migrate	1.0	Windows	Linux	Darwin
112:	Tcladdressbook	1.2.4	Darwin		
113:	tclAE	2.0.7	Darwin		
114:	Tclapplescript	2.2	Darwin		
115:	tclargp	0.2	Windows	Linux	Darwin
116:	tclcompiler	2.0a0	Windows	Linux	Darwin
117:	tclcsv	2.4.3	Windows	Linux	Darwin
118:	tcldebugger	2.0.1	Windows	Linux	Darwin
119:	tclfpdf	1.7.2	Windows	Linux	Darwin
120:	tclgd	1.4.1	Windows	Linux	Darwin
121:	Tclkit		Windows	Linux	Darwin
122:	tcllib	2.1	Windows	Linux	Darwin
123:	tclMuPdf	2.5.1	Windows	Linux	Darwin
124:	tclparser	1.9	Windows	Linux	Darwin
125:	tclpy	0.4.1	Windows	Linux	
126:	tclssg	3.0.1	Windows	Linux	Darwin
127:	TclStubs	8.6.17	Windows		
128:	TclTkManual		Windows	Linux	Darwin
129:	tcltts	1.7.23	Windows	Linux	Darwin
130:	tcluvc	0.1	Linux		
131:	tclvfs	1.5.0	Windows	Linux	Darwin
132:	tclws	3.5.0	Windows	Linux	Darwin
133:	tclx	9.0.0	Windows	Linux	Darwin

134:	tdom	0.9.6	Windows	Linux	Darwin
135:	thtmlview	2.0.0	Windows	Linux	Darwin
136:	TIFF	4.7.0	Windows	Linux	Darwin
137:	tinyxml2	9.0.0	Windows	Linux	Darwin
138:	Tix	8.4.4	Windows	Linux	Darwin
139:	Tk	8.6.17	Windows	Linux	Darwin
140:	tkchat	1.482	Windows	Linux	Darwin
141:	tkcon	2.8	Windows	Linux	Darwin
142:	tkdnd	2.9.5	Windows	Linux	Darwin
143:	Tkhtml	3.0.2	Windows	Linux	Darwin
144:	tklib	0.9	Windows	Linux	Darwin
145:	tko	0.4	Windows	Linux	Darwin
146:	tkpath	0.4.2	Windows	Linux	Darwin
147:	tkribbon	1.2	Windows		
148:	tksqlite	0.5.14	Windows	Linux	Darwin
149:	TkStubs	8.6.17	Windows		
150:	tksvg	0.14	Windows	Linux	Darwin
151:	Tktable	2.12	Windows	Linux	Darwin
152:	tkwintrack	2.1.1	Windows	Linux	
153:	treectrl	2.5.2	Windows	Linux	Darwin
154:	Trf	2.1.4	Windows	Linux	Darwin
155:	trofs	0.4.9	Windows	Linux	Darwin
156:	tserialport	1.1.1	Windows	Linux	Darwin
157:	tsw	1.2	Windows	Linux	Darwin
158:	twapi	5.2.0	Windows		
159:	tzint	1.1.1	Windows	Linux	Darwin
160:	udp	1.0.12	Windows	Linux	Darwin
161:	ukaz	2.1	Windows	Linux	Darwin
162:	vectcl	0.2.1	Windows	Linux	Darwin
163:	Vim	9.0.0	Windows		
164:	vlerq	4.1	Windows	Linux	Darwin
165:	wcb	4.2	Windows	Linux	Darwin
166:	windetect	2.0.1	Windows	Linux	
167:	winhelp	1.1.1	Windows		
168:	Xerces	3.2.4	Windows	Linux	Darwin
169:	xz	5.4.1	Windows	Linux	Darwin
170:	yasm	1.3.0	Windows		
171:	ZLib	1.3.1	Windows	Linux	Darwin

List of all libraries (using command line option --dependencies)		
#:	Name	Version Dependencies

1:	apave	4.4.10 Tk
2:	argparse	0.61 Tcl
3:	awthemes	10.4.0 Tk
4:	BawtLogViewer	3.2.0 Tclkit tablelist tkdnd poLibs scrollutil Img rtext
5:	Blender	3.0.0
6:	Boost	1.78.0
7:	BWidget	1.10.1 Tk
8:	Cal3D	0.120 CMake freeglut
9:	Canvas3d	1.2.3 Tk
10:	cawt	3.1.1 Tcl twapi tdom Img tablelist
11:	cawtapp	3.1.1 Tclkit cawt Img tablelist tdom twapi
12:	ccl	4.0.6 CMake
13:	cffi	2.0.3 Tcl libffi
14:	cfitsio	4.1.0
15:	CMake	3.25.2
16:	critcl	3.3 Tcl
17:	curl	7.70.0 libressl
18:	DiffUtil	0.4.3 Tcl
19:	DirectXTex	2021_11
20:	Doxygen	1.8.15
21:	Eigen	3.3.9
22:	expect	5.45.4.1 Tcl
23:	Ffidl	0.9.1 Tcl libffi
24:	ffmpeg	7.1.2 SDL yasm
25:	fftw	3.3.9
26:	fitsTcl	2.5.1 Tcl cfitsio
27:	freetype	3.2.2 CMake
28:	FreeType	2.13.3 PNG

29:	FTGL	2.1.3	FreeType
30:	gdal	2.4.4	openjpeg
31:	gdi	0.9.9.15	Tk TkStubs
32:	GeographicLib	1.52	CMake
33:	GeographicLibData		GeographicLib
34:	geos	3.7.2	CMake
35:	giflib	5.2.1	
36:	Gl2ps	1.4.2	CMake freeglut PNG ZLib
37:	GLEW	2.2.0	CMake
38:	glfw	3.3.8	CMake
39:	gorilla	1.6.1	Tcl Tclkit
40:	hdc	0.2.0.1	Tk TkStubs
41:	Img	2.1.0	Tk TkStubs tcllib
42:	imgj2p	0.1.1	Tk openjpeg
43:	imgtools	0.3.1	Tcl Tk
44:	InnoSetup	6.2.2	
45:	iocp	2.0.2	Tcl
46:	itk	4.2.5	Tk
47:	iwidgets	4.1.2	Tk itk
48:	jasper	2.0.25	CMake JPEG
49:	jigsaw	2.0	Tclkit puppyicons tcllib tklib Img
50:	JPEG	9.f	
51:	KDIS	2.9.0	CMake
52:	libffi	3.4.8	
53:	libgd	2.3.2	ZLib TIFF JPEG PNG libwebp FreeType
54:	libressl	2.9.2	
55:	libwebp	1.2.4	
56:	libxml2	2.10.3	CMake Zlib
57:	materialicons	0.2	Tk tdom tksvg
58:	mawt	0.5.0	Tk TkStubs SWIG CMake Img ffmpeg
59:	memchan	2.3.1	Tcl
60:	mentry	4.5	Tk wcb
61:	Mpexpr	1.2.1	Tcl
62:	mqt	4.0	Tcl
63:	mupdf	1.26.0	
64:	MuPDFWidget	2.4	Tk tclMuPdf
65:	nacl	1.1.1	Tcl
66:	nas	2.16.3	
67:	nsf	2.4.0	Tcl
68:	OglInfo	1.0.1	Tclkit tcl3dExtended
69:	ooxml	1.10	Tcl tclvfs tdom
70:	openjpeg	2.5.3	CMake
71:	OpenSceneGraph	3.6.5	CMake ZLib TIFF JPEG jasper giflib PNG curl FreeType ffmpeg
72:	OpenSceneGraphData	3.4.0	OpenSceneGraph
73:	oratcl	4.6.1	Tcl
74:	osgcal	0.2.1	Cal3D OpenSceneGraph
75:	osgearth	2.10.1	CMake curl gdal geos OpenSceneGraph
76:	pandoc	3.5	
77:	parse_args	0.5.1	Tcl
78:	pawt	1.2.0	Tcl fitstcl Img
79:	pdf4tcl	0.9.4	Tk
80:	pgintcl	3.5.2	Tcl
81:	photoresize	0.2.1	Tcl Tk
82:	pkgconfig	0.29.2	
83:	PNG	1.6.48	CMake ZLib
84:	poApps	3.2.0	Tclkit tcllib tablelist Img tdom publisher tclMuPdf fitsTcl poImg
85:	poClipboardViewer	3.2.0	Tclkit cawt tcllib tablelist Img twapi tksvg scrollutil
86:	poImg	3.0.1	Tk
87:	poLibs	3.2.0	Tcl Tk
88:	poMemory	1.0.0	Tcl
89:	printer	0.9.6.16	Tk TkStubs hdc
90:	publisher	2.0	Tcl
91:	puppyicons	0.1	Tk tksvg
92:	Python	3.7.7	
93:	rbc	0.2.0	Tk
94:	Redistributables		
95:	rl_json	0.16.0	Tcl
96:	rtext	0.1.1	TclStubs TkStubs
97:	ruff	2.6.0	Tcl
98:	scrollutil	2.6	Tk
99:	SDL	2.26.2	CMake
100:	SetupPython		Python
101:	SetupTcl		All
102:	shellicon	0.2.0	Tk TkStubs
103:	Snack	2.2.12	Tk TkStubs
104:	sqlite3	3.47.1	
105:	SWIG	4.3.1	
106:	tablelist	7.7	Tk
107:	tbcload	2.0a0	Tcl

108:	Tcl	8.6.17	
109:	tcl3dExtended	1.0.1	CMake Tk TkStubs SWIG Freetype FTGL SDL
110:	tcl3dFull	1.0.1	CMake Tk TkStubs SWIG Freetype FTGL SDL OpenSceneGraph
111:	tcl9migrate	1.0	Tcl
112:	Tcladdressbook	1.2.4	Tcl
113:	tclAE	2.0.7	Tcl
114:	Tclapplescript	2.2	Tcl
115:	tclargp	0.2	Tcl
116:	tclcompiler	2.0a0	Tcl
117:	tclcsv	2.4.3	Tcl
118:	tcldebugger	2.0.1	Tk Tclkit tcllib tclparser
119:	tclfpdf	1.7.2	Tk
120:	tclgd	1.4.1	Tcl libgd
121:	Tclkit		Tcl Tk
122:	tcllib	2.1	Tcl critcl Tclkit
123:	tclMuPdf	2.5.1	Tk TkStubs CMake mupdf publisher
124:	tclparser	1.9	Tcl
125:	tclpy	0.4.1	Tk TkStubs Python
126:	tclssg	3.0.1	Tcl Tclkit tcllib
127:	TclStubs	8.6.17	
128:	TclTkManual		Tcl Tk
129:	tcltls	1.7.23	Tcl libressl
130:	tcluvc	0.1	Tcl Tk
131:	tclvfs	1.5.0	Tcl
132:	tclws	3.5.0	Tcl tdom tcllib
133:	tclx	9.0.0	Tcl
134:	tdom	0.9.6	Tcl
135:	thtmlview	2.0.0	Tk
136:	TIFF	4.7.0	JPEG ZLib xz
137:	tinyxml2	9.0.0	CMake
138:	Tix	8.4.4	Tk
139:	Tk	8.6.17	Tcl
140:	tkchat	1.482	Tclkit
141:	tkcon	2.8	Tk
142:	tkdnd	2.9.5	CMake Tk TkStubs
143:	Tkhtml	3.0.2	Tcl Tk
144:	tklib	0.9	Tk
145:	tko	0.4	Tk
146:	tkpath	0.4.2	Tk
147:	tkribbon	1.2	Tk TkStubs
148:	tksqlite	0.5.14	Tcl Tclkit tablelist Tktable treectrl Img
149:	TkStubs	8.6.17	TclStubs
150:	tksvg	0.14	Tk
151:	Tktable	2.12	Tk
152:	tkwintrack	2.1.1	Tk
153:	treectrl	2.5.2	Tk
154:	Trf	2.1.4	Tcl Zlib
155:	trofs	0.4.9	Tk
156:	tserialport	1.1.1	Tcl
157:	tsw	1.2	Tk
158:	twapi	5.2.0	Tcl
159:	tzint	1.1.1	Tcl PNG
160:	udp	1.0.12	Tcl
161:	ukaz	2.1	Tk
162:	vectcl	0.2.1	Tcl
163:	Vim	9.0.0	
164:	vlerq	4.1	Tcl
165:	wcb	4.2	Tk
166:	windetect	2.0.1	Tk
167:	winhelp	1.1.1	Tcl Tk
168:	Xerces	3.2.4	CMake
169:	xz	5.4.1	
170:	yasm	1.3.0	
171:	ZLib	1.3.1	

List of all libraries (using command line option [--authors](#))

#:	Name	Version	ScriptAuthor
1:	apave	4.4.10	Paul Obermeier
2:	argparse	0.61	Paul Obermeier
3:	awthemes	10.4.0	Paul Obermeier
4:	BawtLogViewer	3.2.0	Paul Obermeier
5:	Blender	3.0.0	Paul Obermeier
6:	Boost	1.78.0	Paul Obermeier
7:	BWidget	1.10.1	Paul Obermeier

8:	Cal3D	0.120	Paul Obermeier
9:	Canvas3d	1.2.3	Paul Obermeier
10:	cawt	3.1.1	Paul Obermeier
11:	cawtapp	3.1.1	Paul Obermeier
12:	ccl	4.0.6	Paul Obermeier
13:	cfffi	2.0.3	Paul Obermeier
14:	cfitsio	4.1.0	Paul Obermeier
15:	CMake	3.25.2	Paul Obermeier
16:	critcl	3.3	Paul Obermeier
17:	curl	7.70.0	Paul Obermeier
18:	DiffUtil	0.4.3	Paul Obermeier
19:	DirectXTex	2021_11	Paul Obermeier
20:	Doxygen	1.8.15	Paul Obermeier
21:	Eigen	3.3.9	Paul Obermeier
22:	expect	5.45.4.1	Paul Obermeier
23:	Ffidl	0.9.1	Paul Obermeier
24:	ffmpeg	7.1.2	Paul Obermeier
25:	fftw	3.3.9	Paul Obermeier
26:	fitsTcl	2.5.1	Paul Obermeier
27:	freeglut	3.2.2	Paul Obermeier
28:	FreeType	2.13.3	Paul Obermeier
29:	FTGL	2.1.3	Paul Obermeier
30:	gdal	2.4.4	Paul Obermeier
31:	gdi	0.9.9.15	Paul Obermeier
32:	GeographicLib	1.52	Paul Obermeier
33:	GeographicLibData		Paul Obermeier
34:	geos	3.7.2	Paul Obermeier
35:	giflib	5.2.1	Paul Obermeier
36:	Gl2ps	1.4.2	Paul Obermeier
37:	GLEW	2.2.0	Paul Obermeier
38:	glfw	3.3.8	Paul Obermeier
39:	gorilla	1.6.1	Paul Obermeier
40:	hdc	0.2.0.1	Paul Obermeier
41:	Img	2.1.0	Paul Obermeier
42:	imgjp2	0.1.1	Paul Obermeier
43:	imgtools	0.3.1	Paul Obermeier
44:	InnoSetup	6.2.2	Paul Obermeier
45:	iocp	2.0.2	Paul Obermeier
46:	itk	4.2.5	Paul Obermeier
47:	iwidgets	4.1.2	Paul Obermeier
48:	jasper	2.0.25	Paul Obermeier
49:	jigsaw	2.0	Paul Obermeier
50:	JPEG	9.f	Paul Obermeier
51:	KDIS	2.9.0	Paul Obermeier
52:	libffi	3.4.8	Paul Obermeier
53:	libgd	2.3.2	Alexander Schoepe
54:	libressl	2.9.2	Paul Obermeier
55:	libwebp	1.2.4	Paul Obermeier
56:	libxml2	2.10.3	Paul Obermeier
57:	materialicons	0.2	Paul Obermeier
58:	mawt	0.5.0	Paul Obermeier
59:	memchan	2.3.1	Alexander Schoepe
60:	mentry	4.5	Paul Obermeier
61:	Mpexpr	1.2.1	Paul Obermeier
62:	mqtt	4.0	Paul Obermeier
63:	mupdf	1.26.0	Paul Obermeier
64:	MuPDFWidget	2.4	Paul Obermeier
65:	nacl	1.1.1	Paul Obermeier
66:	nasm	2.16.3	Paul Obermeier
67:	nsf	2.4.0	Paul Obermeier
68:	OglInfo	1.0.1	Paul Obermeier
69:	ooxml	1.10	Paul Obermeier
70:	openjpeg	2.5.3	Paul Obermeier
71:	OpenSceneGraph	3.6.5	Paul Obermeier
72:	OpenSceneGraphData	3.4.0	Paul Obermeier
73:	oratcl	4.6.1	Alexander Schoepe
74:	osgcal	0.2.1	Paul Obermeier
75:	osgearth	2.10.1	Paul Obermeier
76:	pandoc	3.5	Paul Obermeier
77:	parse_args	0.5.1	Paul Obermeier
78:	pawt	1.2.0	Paul Obermeier

79:	pdf4tcl	0.9.4	Paul Obermeier
80:	pgintcl	3.5.2	Paul Obermeier
81:	photoresize	0.2.1	Paul Obermeier
82:	pkgconfig	0.29.2	Paul Obermeier
83:	PNG	1.6.48	Paul Obermeier
84:	poApps	3.2.0	Paul Obermeier
85:	poClipboardViewer	3.2.0	Paul Obermeier
86:	poImg	3.0.1	Paul Obermeier
87:	poLibs	3.2.0	Paul Obermeier
88:	poMemory	1.0.0	Paul Obermeier
89:	printer	0.9.6.16	Paul Obermeier
90:	publisher	2.0	Paul Obermeier
91:	puppyicons	0.1	Paul Obermeier
92:	Python	3.7.7	Paul Obermeier
93:	rbc	0.2.0	Alexander Schoepe
94:	Redistributables		Paul Obermeier
95:	rl_json	0.16.0	Paul Obermeier
96:	rtxt	0.1.1	Paul Obermeier
97:	ruff	2.6.0	Paul Obermeier
98:	scrollutil	2.6	Paul Obermeier
99:	SDL	2.26.2	Paul Obermeier
100:	SetupPython		Paul Obermeier
101:	SetupTcl		Paul Obermeier
102:	shellicon	0.2.0	Paul Obermeier
103:	Snack	2.2.12	Paul Obermeier
104:	sqlite3	3.47.1	Paul Obermeier
105:	SWIG	4.3.1	Paul Obermeier
106:	tablelist	7.7	Paul Obermeier
107:	tbclload	2.0a0	Alexander Schoepe
108:	Tcl	8.6.17	Paul Obermeier
109:	tcl3dExtended	1.0.1	Paul Obermeier
110:	tcl3dFull	1.0.1	Paul Obermeier
111:	tcl9migrate	1.0	Paul Obermeier
112:	Tcladdressbook	1.2.4	Alexander Schoepe
113:	tclAE	2.0.7	Alexander Schoepe
114:	Tclapplescript	2.2	Alexander Schoepe
115:	tclargp	0.2	Paul Obermeier
116:	tclcompiler	2.0a0	Alexander Schoepe
117:	tclcsv	2.4.3	Paul Obermeier
118:	tcldebugger	2.0.1	Paul Obermeier
119:	tclfpdf	1.7.2	Paul Obermeier
120:	tclgd	1.4.1	Alexander Schoepe
121:	Tclkit		Paul Obermeier
122:	tcllib	2.1	Paul Obermeier
123:	tclMuPdf	2.5.1	Paul Obermeier
124:	tclparser	1.9	Alexander Schoepe
125:	tclpy	0.4.1	Paul Obermeier
126:	tclssg	3.0.1	Paul Obermeier
127:	TclStubs	8.6.17	Paul Obermeier
128:	TclTkManual		Paul Obermeier
129:	tcltls	1.7.23	Alexander Schoepe
130:	tcluvc	0.1	Paul Obermeier
131:	tclvfs	1.5.0	Paul Obermeier
132:	tclws	3.5.0	Paul Obermeier
133:	tclx	9.0.0	Paul Obermeier
134:	tdom	0.9.6	Paul Obermeier
135:	thtmlview	2.0.0	Paul Obermeier
136:	TIFF	4.7.0	Paul Obermeier
137:	tinyxml2	9.0.0	Paul Obermeier
138:	Tix	8.4.4	Paul Obermeier
139:	Tk	8.6.17	Paul Obermeier
140:	tkchat	1.482	Paul Obermeier
141:	tkcon	2.8	Paul Obermeier
142:	tkdnd	2.9.5	Paul Obermeier
143:	Tkhtml	3.0.2	Paul Obermeier
144:	tklib	0.9	Paul Obermeier
145:	tko	0.4	Paul Obermeier
146:	tkpath	0.4.2	Paul Obermeier
147:	tkribbon	1.2	Paul Obermeier
148:	tksqlite	0.5.14	Paul Obermeier
149:	TkStubs	8.6.17	Paul Obermeier

150:	tksvg	0.14	Paul Obermeier
151:	Tktable	2.12	Paul Obermeier
152:	tkwintrack	2.1.1	Paul Obermeier
153:	treectrl	2.5.2	Paul Obermeier
154:	Trf	2.1.4	Paul Obermeier
155:	trofs	0.4.9	Paul Obermeier
156:	tserialport	1.1.1	Alexander Schoepe
157:	tsw	1.2	Paul Obermeier
158:	twapi	5.2.0	Paul Obermeier
159:	tzint	1.1.1	Alexander Schoepe
160:	udp	1.0.12	Paul Obermeier
161:	ukaz	2.1	Paul Obermeier
162:	vectcl	0.2.1	Paul Obermeier
163:	Vim	9.0.0	Paul Obermeier
164:	vlerq	4.1	Paul Obermeier
165:	wcb	4.2	Paul Obermeier
166:	windetect	2.0.1	Paul Obermeier
167:	winhelp	1.1.1	Paul Obermeier
168:	Xerces	3.2.4	Paul Obermeier
169:	xz	5.4.1	Paul Obermeier
170:	yasm	1.3.0	Paul Obermeier
171:	ZLib	1.3.1	Paul Obermeier

List of all libraries (using command line option [--homepages](#))

#:	Name	Version	Homepage

1:	apave	4.4.10	https://aplsimple.github.io/en/tcl/pave/index.html
2:	argparse	0.61	https://github.com/georgtree/argparse
3:	awthemes	10.4.0	https://sourceforge.net/projects/tcl-awthemes/
4:	BawtLogViewer	3.2.0	https://www.tcl3d.org/bawt/
5:	Blender	3.0.0	https://www.blender.org/
6:	Boost	1.78.0	https://www.boost.org/
7:	BWidget	1.10.1	https://core.tcl-lang.org/bwidget/
8:	Cal3D	0.120	https://github.com/mp3butcher/Cal3D
9:	Canvas3d	1.2.3	http://3dcanvas.tcl-lang.org/
10:	cawt	3.1.1	https://www.tcl3d.org/cawt/
11:	cawtapp	3.1.1	https://www.tcl3d.org/cawt/
12:	ccl	4.0.6	https://sourceforge.net/projects/cigi/
13:	cffi	2.0.3	https://github.com/apnadkarni/tcl-cffi
14:	cfitsio	4.1.0	https://heasarc.gsfc.nasa.gov/fitsio/
15:	CMake	3.25.2	https://www.cmake.org/
16:	critcl	3.3	https://andreas-kupries.github.io/critcl/
17:	curl	7.70.0	https://curl.haxx.se/libcurl/
18:	DiffUtil	0.4.3	https://github.com/pspjuth/DiffUtilTcl/
19:	DirectXTex	2021_11	https://github.com/microsoft/DirectXTex/
20:	Doxygen	1.8.15	http://www.doxygen.org/
21:	Eigen	3.3.9	http://eigen.tuxfamily.org/
22:	expect	5.45.4.1	https://sourceforge.net/projects/expect/
23:	Ffidl	0.9.1	https://github.com/prs-de/ffidl
24:	ffmpeg	7.1.2	https://www.ffmpeg.org/
25:	fftw	3.3.9	http://www.fftw.org/
26:	fitsTcl	2.5.1	https://heasarc.gsfc.nasa.gov/docs/software/ftools/fv/fitsTcl_home.html
27:	freeglut	3.2.2	https://sourceforge.net/projects/freeglut/
28:	FreeType	2.13.3	http://www.freetype.org/
29:	FTGL	2.1.3	https://sourceforge.net/projects/ftgl/
30:	gdal	2.4.4	https://www.gdal.org/
31:	gdi	0.9.9.15	http://www.schwartzcomputer.com/tcl-tk/tcl-tk.html
32:	GeographicLib	1.52	https://geographiclib.sourceforge.io/
33:	GeographicLibData		https://geographiclib.sourceforge.io/
34:	geos	3.7.2	http://trac.osgeo.org/geos/
35:	giflib	5.2.1	http://giflib.sourceforge.net/
36:	GL2ps	1.4.2	http://www.geuz.org/gl2ps/
37:	GLEW	2.2.0	https://github.com/nigels-com/glew/
38:	glfw	3.3.8	https://www.glfw.org/
39:	gorilla	1.6.1	https://github.com/zdia/gorilla/wiki
40:	hdc	0.2.0.1	http://www.schwartzcomputer.com/tcl-tk/tcl-tk.html
41:	Img	2.1.0	https://tkimg.sourceforge.net/
42:	imgjp2	0.1.1	https://www.androwish.org/home/dir?name=jni/imgjp2
43:	imgtools	0.3.1	https://tkimgtools.sourceforge.net/
44:	InnoSetup	6.2.2	http://www.jrsoftware.org/isinfo.php
45:	iocp	2.0.2	https://github.com/apnadkarni/iocp/

46:	itk	4.2.5	https://core.tcl-lang.org/itk/
47:	iwidgets	4.1.2	https://sourceforge.net/projects/incrtcl/
48:	jasper	2.0.25	https://github.com/jasper-software/jasper/
49:	jigsaw	2.0	http://www.easton.me.uk/tcl/jigsaw/index.html
50:	JPEG	9.f	http://www.ijg.org/
51:	KDIS	2.9.0	https://sourceforge.net/projects/kdis/
52:	libffi	3.4.8	https://github.com/libffi/libffi
53:	libgd	2.3.2	https://libgd.github.io
54:	libressl	2.9.2	https://www.libressl.org/
55:	libwebp	1.2.4	https://developers.google.com/speed/webp/
56:	libxml2	2.10.3	https://gitlab.gnome.org/GNOME/libxml2
57:	materialicons	0.2	https://www.androwish.org/home/dir?ci=tip&name=assets/materialicons0.2
58:	mawt	0.5.0	https://www.tcl3d.org/mawt/
59:	memchan	2.3.1	https://memchan.sourceforge.net/
60:	mentry	4.5	http://www.nemethi.de/
61:	Mpexpr	1.2.1	https://sourceforge.net/projects/mpexpr/
62:	mqtt	4.0	https://chiselapp.com/user/schelte/repository/mqtt/home
63:	mupdf	1.26.0	https://mupdf.com/
64:	MuPDFWidget	2.4	https://sourceforge.net/projects/irrational-numbers/
65:	nacl	1.1.1	https://tcl.sowaswie.de/repos/fossil/nacl/home
66:	nasm	2.16.3	https://www.nasm.us/
67:	nsf	2.4.0	https://next-scripting.org
68:	OglInfo	1.0.1	https://www.tcl3d.org/
69:	ooxml	1.10	https://fossil.sowaswie.de/ooxml/home
70:	openjpeg	2.5.3	http://www.openjpeg.org/
71:	OpenSceneGraph	3.6.5	http://www.openscenegraph.org/
72:	OpenSceneGraphData	3.4.0	http://www.openscenegraph.org/
73:	oratcl	4.6.1	https://github.com/sm-shaw/Oratcl
74:	osgcal	0.2.1	https://sourceforge.net/projects/osgcal/
75:	osgearth	2.10.1	http://osgearth.org/
76:	pandoc	3.5	https://pandoc.org/
77:	parse_args	0.5.1	https://github.com/RubyLane/parse_args
78:	pawt	1.2.0	https://www.tcl3d.org/pawt/
79:	pdf4tcl	0.9.4	https://sourceforge.net/projects/pdf4tcl/
80:	pgintcl	3.5.2	https://sourceforge.net/projects/pgintcl/
81:	photoresize	0.2.1	https://github.com/auriocus/PhotoResize
82:	pkgconfig	0.29.2	https://www.freedesktop.org/wiki/Software/pkg-config/
83:	PNG	1.6.48	http://www.libpng.org/pub/png/
84:	poApps	3.2.0	https://www.tcl3d.org/poApps/
85:	poClipboardViewer	3.2.0	http://www.poSoft.de/
86:	poImg	3.0.1	https://www.tcl3d.org/poPkgs/poImg.html
87:	poLibs	3.2.0	http://www.poSoft.de/
88:	poMemory	1.0.0	https://www.tcl3d.org/poPkgs/poMemory.html
89:	printer	0.9.6.16	http://www.schwartzcomputer.com/tcl-tk/tcl-tk.html
90:	publisher	2.0	https://sourceforge.net/projects/irrational-numbers/
91:	puppyicons	0.1	https://www.androwish.org/home/dir?ci=tip&name=android/puppyicons0.1
92:	Python	3.7.7	http://www.python.org/
93:	rbc	0.2.0	https://www.sourceforge.net/projects/rbctoolkit/
94:	Redistributables		https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-
	redist		
95:	rl_json	0.16.0	https://github.com/RubyLane/rl_json
96:	rtext	0.1.1	https://chiselapp.com/user/fvogel/repository/rtext
97:	ruff	2.6.0	https://ruff.magicsplat.com/
98:	scrollutil	2.6	http://www.nemethi.de/
99:	SDL	2.26.2	https://www.libsdl.org/
100:	SetupPython		https://www.tcl3d.org/bawt/
101:	SetupTcl		https://www.tcl3d.org/bawt/
102:	shellicon	0.2.0	https://sourceforge.net/projects/shellicon/
103:	Snack	2.2.12	https://github.com/scottypitcher/tcl-snack
104:	sqlite3	3.47.1	https://www.sqlite.org/
105:	SWIG	4.3.1	http://www.swig.org/
106:	tablelist	7.7	http://www.nemethi.de/
107:	tbclload	2.0a0	https://github.com/tcltk-depot/tbclload
108:	Tcl	8.6.17	https://core.tcl-lang.org/tcl/
109:	tcl3dExtended	1.0.1	https://www.tcl3d.org/
110:	tcl3dFull	1.0.1	https://www.tcl3d.org/
111:	tcl9migrate	1.0	https://github.com/apnadkarni/tcl9-migrate
112:	Tcladdressbook	1.2.4	https://sourceforge.net/projects/tcladdressbook/
113:	tclAE	2.0.7	https://sourceforge.net/projects/tclae/
114:	Tclapplescript	2.2	https://sourceforge.net/projects/tclapplescript/
115:	tclargp	0.2	https://wiki.tcl-lang.org/page/argp
116:	tclcompiler	2.0a0	https://github.com/tcltk-depot/tclcompiler
117:	tclcsv	2.4.3	https://sourceforge.net/projects/tclcsv
118:	tcldebugger	2.0.1	https://github.com/tcltk-depot/tcl-debugger
119:	tclfpdf	1.7.2	https://github.com/lamuzzachiodi/tclfpdf
120:	tclgd	1.4.1	https://github.com/flightaware/tcl.gd
121:	Tclkit		https://sourceforge.net/projects/kbskit/
122:	tcllib	2.1	https://core.tcl-lang.org/tcllib
123:	tclMuPdf	2.5.1	https://sourceforge.net/projects/irrational-numbers/

124:	tclparser	1.9	https://github.com/tcltk-depot/tcl-parser
125:	tclpy	0.4.1	https://github.com/aidanhs/libtclpy
126:	tclssg	3.0.1	https://github.com/tclssg/tclssg
127:	TclStubs	8.6.17	https://core.tcl-lang.org/tcl/
128:	TclTkManual		http://www.tcl-lang.org
129:	tcltls	1.7.23	http://core.tcl-lang.org/tcltls/
130:	tcluvc	0.1	https://www.androwish.org/home/dir?ci=tip&name=jni/tcluvc
131:	tclvfs	1.5.0	https://core.tcl-lang.org/tclvfs/
132:	tclws	3.5.0	https://core.tcl-lang.org/tclws/
133:	tclx	9.0.0	https://github.com/tcltk-depot/tclx
134:	tdom	0.9.6	http://tdom.org/
135:	thtmlview	2.0.0	https://github.com/mittelmark/thtmlview/
136:	TIFF	4.7.0	http://www.simplesystems.org/libtiff/
137:	tinyxml2	9.0.0	https://github.com/leethomason/tinyxml2
138:	Tix	8.4.4	https://tix.sourceforge.net/
139:	Tk	8.6.17	https://core.tcl-lang.org/tk/
140:	tkchat	1.482	http://tkchat.tcl-lang.org/
141:	tkcon	2.8	https://github.com/bohagan1/TkCon/
142:	tkdnd	2.9.5	https://github.com/petasis/tkdnd
143:	Tkhtml	3.0.2	https://github.com/olebole/tkhtml3
144:	tklib	0.9	https://core.tcl-lang.org/tklib
145:	tko	0.4	https://chiselapp.com/user/rene/repository/tko/index
146:	tkpath	0.4.2	https://github.com/tcltk-depot/tkpath
147:	tkribbon	1.2	https://github.com/petasis/tkribbon
148:	tksqlite	0.5.14	http://reddog.s35.xrea.com/wiki/TkSQLite.html
149:	TkStubs	8.6.17	https://core.tcl-lang.org/tk/
150:	tksvg	0.14	https://github.com/oehhar/tksvg/
151:	Tktable	2.12	https://github.com/bohagan1/TkTable/
152:	tkwintrack	2.1.1	https://sourceforge.net/projects/tkwintrack/
153:	treectrl	2.5.2	https://github.com/tcltk-depot/tktreectrl
154:	Trf	2.1.4	https://tcltrf.sourceforge.net/
155:	trofs	0.4.9	https://math.nist.gov/~DPorter/tcltk/trofs/
156:	tserialport	1.1.1	https://fossil.sowaswie.de/tserialport
157:	tsw	1.2	http://www.nemethi.de/
158:	twapi	5.2.0	https://github.com/apnadkarni/twapi
159:	tzint	1.1.1	https://fossil.sowaswie.de/tzint/
160:	udp	1.0.12	https://core.tcl-lang.org/tcludp/
161:	ukaz	2.1	https://github.com/auriocus/ukaz
162:	vectcl	0.2.1	https://github.com/auriocus/VecTcl/
163:	Vim	9.0.0	https://www.vim.org/
164:	vlerq	4.1	https://www.equi4.com/vlerq-org/
165:	wcb	4.2	http://www.nemethi.de/
166:	windetect	2.0.1	https://sourceforge.net/projects/tkwintrack/
167:	winhelp	1.1.1	https://www.androwish.org/index.html/dir?name=undroid/winhelp
168:	Xerces	3.2.4	http://xerces.apache.org/
169:	xz	5.4.1	https://sourceforge.net/projects/lzmautils/
170:	yasm	1.3.0	https://github.com/yasm/yasm/
171:	ZLib	1.3.1	http://www.zlib.net/

9 MSYS / MinGW Information

This chapter describes the development environments `MSYS` and `MinGW`. These packages provide an environment using the GNU compiler collection (`gcc`) to build typical Open Source projects like **Tcl/Tk** under Windows.

9.1 Introduction

MSYS

MSYS, a contraction of "Minimal SYStem", is a Bourne Shell command line interpreter system. Offered as an alternative to Microsoft's `cmd.exe`, this provides a general purpose command line environment, which is particularly suited to use with MinGW, for porting of many Open Source applications to the MS-Windows platform.

MSYS is a collection of Unix tools for Windows. It contains all tools which are needed for the typical build process using the `configure / make` toolset.

Examples: `autogen`, `cp`, `rm`, `mv`, `mkdir`, `m4`, `make`

MSYS is available as 32-bit version only. This version can be used in conjunction with both the 32-bit and 64-bit version of `MinGW`.

MSYS2

MSYS2 is a newer version of MSYS.
It is available from <https://www.msys2.org/>.

MinGW

Short description from the homepage of MinGW-w64: <https://sourceforge.net/projects/mingw-w64/>

MinGW, a contraction of "Minimalist GNU for Windows", is a minimalist development environment for native Microsoft Windows applications.

MinGW provides a complete Open Source programming tool set which is suitable for the development of native MS-Windows applications, and which do not depend on any 3rd-party C-Runtime DLLs. (It does depend on a number of DLLs provided by Microsoft themselves, as components of the operating system; most notable among these is `MSVCRT.DLL`, the Microsoft C runtime library. Additionally, threaded applications must ship with a freely distributable thread support DLL, provided as part of MinGW itself).

MinGW compilers provide access to the functionality of the Microsoft C runtime and some language-specific runtimes. MinGW, being Minimalist, does not, and never will, attempt to provide a POSIX runtime environment for POSIX application deployment on MS-Windows.

MinGW provides the GNU Compiler Collection `gcc` for Windows. The SourceForge project `MinGW-w64` supplies 32-bit and 64-bit versions of `gcc`.

The `MinGW-w64` project also supplies an extended version of `MSYS` (see chapter [9.2 Installation](#) below for details).

9.2 Installation of MSYS

The following instructions were used to create the BAWT MSYS/MinGW distributions for gcc versions 4.9.2, 5.2.0, 7.2.0 and 8.1.0.

9.2.1 Download MSYS

Entry page:

<https://sourceforge.net/projects/mingwbuilds/files/external-binary-packages/>

File: *msys+7za+wget+svn+git+mercurial+cvs-rev13.7z*

Link:

<https://sourceforge.net/projects/mingwbuilds/files/external-binary-packages/msys%2B7za%2Bwget%2Bsvn%2Bgit%2Bmercurial%2Bcvs-rev13.7z/download>

9.2.2 Download MinGW

Entry page for 32-bit version:

<https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/>

Entry page for 64-bit version:

<https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/>

32-bit gcc 4.9.2

File: *i686-4.9.2-release-posix-dwarf-rt_v4-rev4.7z*

Link:

https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/4.9.2/threads-posix/dwarf/i686-4.9.2-release-posix-dwarf-rt_v4-rev4.7z/download

32-bit gcc 5.2.0

File: *i686-5.2.0-release-posix-dwarf-rt_v4-rev0.7z*

Link:

https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/5.2.0/threads-posix/dwarf/i686-5.2.0-release-posix-dwarf-rt_v4-rev0.7z/download

32-bit gcc 7.2.0

File: *i686-7.2.0-release-posix-dwarf-rt_v5-rev1.7z*

Link:

https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/7.2.0/threads-posix/dwarf/i686-7.2.0-release-posix-dwarf-rt_v5-rev1.7z/download

32-bit gcc 8.1.0

File: *i686-8.1.0-release-posix-dwarf-rt_v6-rev0.7z*

Link:

https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/8.1.0/threads-posix/dwarf/i686-8.1.0-release-posix-dwarf-rt_v6-rev0.7z/download

64-bit gcc 4.9.2

File: `x86_64-4.9.2-release-posix-seh-rt_v4-rev4.7z`

Link:

https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/4.9.2/threads-posix/seh/x86_64-4.9.2-release-posix-seh-rt_v4-rev4.7z/download

64-bit gcc 5.2.0

File: `x86_64-5.2.0-release-posix-seh-rt_v4-rev0.7z`

Link:

https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/5.2.0/threads-posix/seh/x86_64-5.2.0-release-posix-seh-rt_v4-rev0.7z/download

64-bit gcc 7.2.0

File: `x86_64-7.2.0-release-posix-seh-rt_v5-rev1.7z`

Link:

https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/7.2.0/threads-posix/seh/x86_64-7.2.0-release-posix-seh-rt_v5-rev1.7z/download

64-bit gcc 8.1.0

File: `x86_64-8.1.0-release-posix-seh-rt_v6-rev0.7z`

Link:

https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/8.1.0/threads-posix/seh/x86_64-8.1.0-release-posix-seh-rt_v6-rev0.7z/download

9.2.3 Configuration

The following instructions use the 32-bit version of gcc 4.9.2. Installation is done on drive C: Adapt file and directory names accordingly, when using other versions.

- Create directory `C:\MinGW-gcc4.9.2-32`
- Extract downloaded MinGW file in directory `C:\MinGW-gcc4.9.2-32`
- Extract downloaded MSYS file in directory `C:\MinGW-gcc4.9.2-32`

Your directory structure should now look as follows:

Dieser PC > OS (C:) > MinGW-gcc4.9.2-32 >		
Name	Änderungsdatum	Typ
mingw32	06.11.2015 16:49	Dateiordner
msys	06.11.2015 17:00	Dateiordner

Insert the next two lines into file `C:\MinGW-gcc4.9.2-32\msys\etc\fstab`

```
# Win32_Path          Mount_Point
C:/MinGW-gcc4.9.2-32/mingw32  /mingw
```

Start the MSYS Shell by double-clicking onto file `C:\MinGW-gcc4.9.2-32\msys\msys.bat`
You may create a shortcut of `msys.bat` on your desktop for easier access.

9.3 Installation of MSYS2

The following instructions were used to create the BAWT MSYS/MinGW distributions for gcc versions 11.2.0, 12.2.0, 13.2.0 and 14.2.0.

Download the newest MSYS2 32-bit installer.
Change the gcc versions as appropriate.

9.3.1 MSYS2/MinGW 64-bit

- Execute the installer program and install into `C:\gcc13.2.0_x86_64-w64-mingw32`
- After installation perform the following commands to update the packages and add additional packages needed for BAWT.

```
> pacman -Syu
```

- Run "MSYS2 MSYS" from Start menu.
- Update the rest of the base packages:

```
> pacman -Su
```

- Install additional tools:

```
> pacman -S --needed base-devel
> pacman -S zip
```

- Install compiler for 64-bit:

```
> pacman -S --needed mingw-w64-x86_64-toolchain
```

- Select the following packages:
`mingw-w64-x86_64-binutils`
`mingw-w64-x86_64-gcc`
`mingw-w64-x86_64-make`
`mingw-w64-x86_64-pkgconf`
`mingw-w64-x86_64-tools-git`
- Quit MSYS shell.

- Make adjustments for BAWT usage:
Remove file InstallationLog.txt
Remove directories mingw32 and installerResources.
Create directory msys32 and move all files and directories except mingw64 into msys32.
Remove subdirectory "User" from msys32/home.
- Create 7z file of `C:\gcc13.2.0_x86_64-w64-mingw32` using Ultra compression.

9.3.2 MSYS2/MinGW 32-bit

- Execute the installer program and install into `C:\gcc13.2.0_i686-w64-mingw32`
- After installation perform the following commands to update the packages and add additional packages needed for BAWT.

```
> pacman -Syu
```

- Run "MSYS2 MSYS" from Start menu.
- Update the rest of the base packages:

```
> pacman -Su
```

- Install additional tools:

```
> pacman -S --needed base-devel  
> pacman -S zip
```

- Install compiler for 32-bit:

```
> pacman -S --needed mingw-w64-i686-toolchain
```

- Select the following packages:
mingw-w64-i686-binutils
mingw-w64-i686-gcc
mingw-w64-i686-make
mingw-w64-i686-pkgconf
mingw-w64-i686-tools-git
- Quit MSYS shell.
- Make adjustments for BAWT usage:
Remove file InstallationLog.txt
Remove directories mingw64 and installerResources.
Create directory msys32 and move all files and directories except mingw32 into msys32.
Remove subdirectory "User" from msys32/home.
- Create 7z file of `C:\gcc13.2.0_i686-w64-mingw32` using Ultra compression.

9.4 Further Informations

Source: <https://sourceforge.net/p/mingw-w64/wiki2/MSYS/>

9.4.1 What is MSYS

MSYS is a Minimal SYStem, providing several crucial unix utilities under a compatibility layer (the msys-1.0.dll file). MSYS should provide everything to make compilation of common GNU software.

MSYS provided by the mingw-w64/w32 project

This package is not more than a collection of the 50+ packages provided by mingw.org. It was created as a (huge) convenience to our users, to let them be productive instead of downloading every part separately. The accompanying sources are also provided and can be found in the same download section as mentioned above.

This package is 32-bit, but will run flawlessly on x64 Windows. There will never be a 64-bit native MSYS (is there any need?) because the only compiler capable of building MSYS applications is the outdated gcc 3.4.4, which does not support x64 native Windows targets.

9.4.2 Where to get MSYS

There are three places you can get MSYS:

- The [MinGW project](#), with separate packages of all official MSYS packages. Takes a long time to download and install everything.
- The all-in-one package on the [MinGW-w64 download page](#). It is updated on request (see third option for very up to date collection)
- [MinGW-builds](#) provides an ultra-inclusive MSYS package with a bunch of additional useful stuff.

9.4.3 How to use MSYS

Installing MSYS is quite easy.

- You'll need to download the above package.
- Unzip it somewhere, for example C:\msys so that C:\msys\bin contains (among others) bash.exe.
- Doubleclick (or make a handy shortcut and run that) on C:\msys\msys.bat.
- Type sh /postinstall/pi.sh
- Answer the friendly questions and you're all set up.

Mingw-w64/w32 specifics

When running an autotools configure script, these options will come in handy:

- for a 64-bit build: `--host=x86_64-w64-mingw32`
- for a 32-bit build: `--host=i686-w64-mingw32`

If you are experiencing problems, you can also set `--build` to the same value. Some configure scripts also use `--target` instead of `--host`. Use `configure --help` to get all possible options.

--host, --target, and --build explained

`--host` specifies on what platform/architecture the compiled program is going to run. `--target` specifies the platform/architecture that the program should be configured for and will be compiled for. This should only have effect when building cross-compilers. `--build` specifies the platform/architecture the build process is going to be executed.

10 Release history

The following table gives an overview of the release history of **BAWT**. For detailed release information see the [BAWT homepage](#).

Version	Date	Release notes
0.1.0	2016-06-24	First version introduced at EuroTcl 2016 in Eindhoven.
0.2.0	2016-08-27	Improved build actions. New and updated libraries.
0.3.0	2016-10-23	Improved build actions. New and updated libraries.
0.4.0	2016-12-28	Improved build actions. New and updated libraries.
0.5.0	2017-03-19	Improved build actions. New and updated libraries.
0.6.0	2017-07-20	Improved build actions. New and updated libraries.
0.7.0	2017-08-26	Improved build actions. New and updated libraries.
0.7.1	2017-09-12	Support for Tcl/Tk 8.7.
0.7.2	2017-09-24	Support for Visual Studio 2017.
0.7.3	2018-01-04	Tcl/Tk 8.6.8. New and updated libraries.
0.8.0	2018-07-04	Support for nested Setup files. New and updated libraries.
0.9.0	2018-12-28	Tcl/Tk 8.6.9. New and updated libraries.
0.9.1	2019-03-09	Better support for Debug build mode. New and updated libraries.
1.0.0	2019-06-23	Several incompatible changes. Support for Visual Studio 2019.
1.1.0	2019-12-28	Tcl/Tk 8.6.10. Improved MinGW support for several libraries. New and updated libraries.
1.1.1	2020-01-12	Improved handling of C++ based Tcl extensions.
1.1.2	2020-02-16	Improved BawtLogViewer. New and updated libraries.
1.1.3	2020-03-15	Improved Linux build. Updated libraries.
1.1.4	2020-05-02	Improved MinGW support for several libraries. New and updated libraries.
1.2.0	2020-06-09	Additional MSYS2 support. New and updated libraries.
1.2.1	2020-09-05	Support for Tcl/Tk 8.7a4. New and updated libraries.
1.3.0	2021-01-08	Support for Tcl/Tk 8.6.11. Improved support for Tcl/Tk 8.7.a4. New and updated libraries.
2.0.0	2021-08-22	Support for primary and secondary compiler on Windows. Tcl/Tk 8.7.a5. New and updated libraries.
2.1.0	2021-12-28	Support for Tcl/Tk 8.6.12. New and updated libraries.
2.2.0	2022-04-15	Support for MinGW gcc 11. New and updated libraries.
2.2.1	2022-07-17	Maintenance release. New and updated libraries.
2.3.0	2022-12-18	Support for Tcl/Tk 8.6.13 and Apple Silicon (ARM). New and updated libraries.
2.3.1	2023-01-19	Maintenance release. New and updated libraries.
3.0.0	2024-12-28	Major release. Support for Tcl/Tk 9, Linux ARM and RISC-V. New and updated libraries.
3.0.1	2024-12-31	Patch release. Prefer BAWT supplied zip program on Windows.
3.1.0	2025-08-24	Maintenance release. Support for Tcl/Tk 8.6.17 and 9.0.2.
3.2.0	2025-11-08	Maintenance release. Support for Tcl/Tk 9.0.3.