**David T. Ashley’s Tool Set**

Software Engineering Manual

Table of Contents

[1 Introduction and Overview 3](#_Toc469780327)

[2 Licensing 3](#_Toc469780328)

[2.1 Licensing History 3](#_Toc469780329)

[2.2 License 4](#_Toc469780330)

[3 Use of Code Signing Key 4](#_Toc469780331)

[4 Tool Set Design 4](#_Toc469780332)

[5 Supported Platforms and Build Variants 5](#_Toc469780333)

[6 Build Instructions 7](#_Toc469780334)

[7 Coding Standards 7](#_Toc469780335)

[8 Design Standards 7](#_Toc469780336)

[9 Testing Standards, and Testing 7](#_Toc469780337)

[10 Index 8](#_Toc469780338)

# Introduction and Overview

*David T. Ashley’s Tool Set*[[1]](#footnote-1) (or *DTATS*) is the name given to the collection of all of Dave Ashley’s open-source software endeavors that are intended to run on a personal computer or server (rather than on an embedded system). Much of the tool set is geared towards embedded software development, but it is an eclectic collection.

This document contains the software engineering description of the tool set, and covers issues that are not generally of interest to casual users of the tool set. These issues include:

* Licensing.
* Procedures for using a code signing key.
* A description of supported platforms for which the tool set can be built.
* Build instructions for supported platforms.
* Design of the tool set.
* Coding standards.
* Design standards.
* Testing standards, and how the tool set is tested.

# Licensing

## Licensing History

I do embedded software for a living, and I’m aware that:

* Most consumers do not want to know anything about the origin of the software that operates their consumer electronic device. Making such information available to consumers simply annoys them.
* Open-source licenses impose a burden on manufacturers or embedded products because:
  + They require the manufacturer to make information available to consumers, either in printed form or on the Internet, about which open-source software components are used in the products, AND
  + They require modified software components to be made available as well.

My preference would be for a *do what you want but you can’t sue me* license for all software I make publicly available, but this is not considered an open-source license. I did have an e-mail exchange with Richard Stallman around 2000 about what constitutes an open-source license. Mr. Stallman is of the opinion that an open-source license must require notice to consumers of the inclusion of open-source software in an embedded product, and must require that modified source code be made public. (I consider both elements to be burdensome for manufacturers of embedded products.)

I’ve decided to take the following approach:

* Software intended for personal computers and servers is released under a minimally-restrictive open-source license (*The MIT License*, see §2.2).
* Software intended to be used in an embedded system is released under a minimally-restrictive license that does not meet the criteria to be an open-source license. (This license is yet to be authored.) This license does not require manufacturers of embedded products to advise consumers that the embedded product contains this software, and does not require users to make changes public.

## License

David T. Ashley’s Tool Set (source code, binaries, ancillary documents / files / images) is released under The MIT License. The license text is reproduced below.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The intent of this choice of a license is to be as permissive as possible while still meeting the minimum requirements for an open-source license that limits liability.

# Use of Code Signing Key

TBD.

I anticipate in the future that I will begin the practice of signing executables. In this section, I need to include the procedure.

# Tool Set Design

The tool set consists of:

* A number of individual projects (i.e. programs):
  + Each project consists of:
    - The project files (Visual Studio project files, makefiles, etc.).
    - Source and graphics files that are unique to the program (the *main()* function, icons, etc.).
  + Each project may make reference to files in the shared source code (described below).
  + Each project parameterizes the build (by setting preprocessor directives) for the target platform.
* Shared source code:
  + Does not stand alone—it is included in a project.
  + Parameterized for the build platforms and variants.

# Supported Platforms and Build Variants

The C/C++ code of the tool set is build is parameterized in a number of nearly orthogonal directions, as described in Table 1.

Within a build, every C/C++ source file is parameterized identically. In a product like Microsoft Visual Studio, the parameterization would be done via GUI options that affect the options provided to the C/C++ compiler. In a more traditional command-line build, the parameterization would typically be done via the “-D” compiler option.

Within each category, constants are mutually exclusive, and only one constant can be applied, for example, “-D DTATS\_PF=DTATS\_PF\_K\_MFC”. In the future, bit-masked constants (not mutually exclusive) may be added.

Table 1: C/C++ Build Parameterization

|  |  |
| --- | --- |
| **PREPROCESSOR CONSTANT** | **DESCRIPTION** |
| **Platform (DTATS\_PF)** | |
| DTATS\_PF\_K\_WINAPI | Windows API (also sometimes called Win32, although this a misnomer because 64-bit programs can also use the Win32 API). |
| DTATS\_PF\_K\_MFC | Program uses the Windows API with the MFC. |
| DTATS\_PF\_K\_WIN\_NET | Windows .NET. |
| DTATS\_PF\_K\_UNIX | Unix. |
| DTATS\_PF\_K\_LINUX | Linix. |
| DTATS\_PF\_K\_FREE\_BSD | Free BSD. |
| DTATS\_PF\_K\_ANDROID | Android. |
| DTATS\_PF\_K\_FIRE\_OS | Fire OS. |
| DTATS\_PF\_K\_IOS | iOS. |
| **Machine Word Size (DTATS\_MWS)**  **(Note: machine word size does not imply C or C++ default integer size.)** | |
| DTATS\_MWS\_K\_16 | The machine word size is 16 bits. |
| DTATS\_MWS\_K\_24 | The machine word size is 24 bits. |
| DTATS\_MWS\_K\_32 | The machine word size is 32 bits. |
| DTATS\_MWS\_K\_48 | The machine word size is 48 bits. |
| DTATS\_MWS\_K\_64 | The machine word size is 64 bits. |
| DTATS\_MWS\_K\_96 | The machine word size is 96 bits. |
| DTATS\_MWS\_K\_128 | The machine word size is 128 bits. |
| DTATS\_MWS\_K\_GT\_128 | The machine word size is greater than 128 bits. |
| **Machine Integer Representation (DTATS\_MIR)** | |
| DTATS\_MIR\_K\_2SCOMP | Integers have traditional 2’s complement representation. (This allows many programming optimizations.) |
| DTATS\_MIR\_K\_SIGNMAG | Integers have sign-magnitude representation. |
| DTATS\_MIR\_K\_OTHER | Integers have another representation. |
| **Machine Floating Point Unit (DTATS\_MFPU)** | |
| DTATS\_MFPU\_K\_NO | Hardware does not have a floating-point processor, and floating-point operations are done in software (relatively slow). |
| DTATS\_MFPU\_K\_YES | Hardware does have a floating-point processor, and floating-point operations are done in hardware (very quick). |
| **Program Type (DTATS\_PROGTYPE)** | |
| DTATS\_PROGTYPE\_K\_CONSOLE | Program is a console-mode utility (text input, text output). |
| DTATS\_PROGTYPE\_K\_WINGUI | Program is a graphical program under Windows. |
| DTATS\_PROGTYPE\_K\_TCL\_A\_CONSOLE | Program is a Tcl console-mode utility, using Tcl code ported by Dave Ashley around 2004. |
| DTATS\_PROGTYPE\_K\_TCL\_A\_GUI | Program is a Tcl/Tk graphical utility, using Tcl/Tk code ported by Dave Ashley around 2004. |
| DTATS\_PROGTYPE\_K\_TCL\_B\_CONSOLE | Placeholder for future console port of Tcl. |
| DTATS\_PROGTYPE\_K\_TCL\_B\_GUI | Placeholder for future graphical port of Tcl/Tk. |
| DTATS\_PROGTYPE\_K\_CLIKE\_A\_CONSOLE | Placeholder for future console application involving “Clike” (a yet-to-be-developed C-like scripting language). |
| DTATS\_PROGTYPE\_K\_CLIKE\_A\_GUI | Placeholder for future graphical application involving “Clike” (a yet-to-be-developed C-like scripting language). |
| DTATS\_PROGTYPE\_K\_UNIX\_SWING | Program developed using Unix Swing. |
| DTATS\_PROGTYPE\_K\_UNIX\_AWT | Program developed using Unix AWT. |
| DTATS\_PROGTYPE\_K\_CGIBIN\_HELPER | Program is invoked by CGI-BIN PHP, Python, or Perl scripts to implement functionality awkward under the scripting language. |
| DTATS\_PROGTYPE\_K\_CGIBIN\_HTTPD | Program is a CGI-BIN program invoked directly by httpd to answer HTTP[S] requests. |
| DTATS\_PROGTYPE\_K\_CGIBIN\_SERVER | Program listens on TCP ports and is an actual HTTP[S] server. |
| DTATS\_PROGTYPE\_K\_UNITTEST\_MODULE | Program is a unit test program compiled to test an individual software module. |
| **Screen Size (DTATS\_SCREENSIZE)** | |
| DTATS\_SCREENSIZE\_K\_SMALL | The target screen size is small (such as a cellphone). |
| DTATS\_SCREENSIZE\_K\_LARGE | The target screen size is large (such as a tablet computer or computer). |
| DTATS\_SCREENSIZE\_K\_ADAPTIVE | The program adapts to the screen size. |
| **Threadedness (DTATS\_THREADS)** | |
| DTATS\_THREADS\_K\_1 | The program runs with one thread, a greatly reduced priority (essentially, a background program). |
| DTATS\_THREADS\_K\_1 | The program runs with one thread, at unmodified priority. |
| DTATS\_THREADS\_K\_2 | The program runs with two threads, at unmodified priority. |
| DTATS\_THREADS\_K\_3 | The program runs with three threads, at unmodified priority. |
| DTATS\_THREADS\_K\_4 | The program runs with four threads, at unmodified priority. |
| DTATS\_THREADS\_ADAPT\_HALF\_CORES | The program adapts to the number of cores on the target system, attempting to use one half of the cores, at normal priority. |
| DTATS\_THREADS\_ADAPT\_ALL\_CORES | The program adapts to the number of cores on the target system, attempting to use all of the cores, at normal priority. |
| DTATS\_THREADS\_PROG\_SET | The number of threads and priority are set by the program (rather than at compile time). |

# Build Instructions

TBD.

# Coding Standards

TBD.

# Design Standards

TBD.

# Testing Standards, and Testing

TBD.

# Index

David T. Ashley’s Tool Set, 3

DTATS, 3

testing, 3

1. I opted not to use the word *toolset*, as it tends to have the narrower meaning of add-ins for a specific application. [↑](#footnote-ref-1)