**David T. Ashley’s Tool Set**

Software Engineering Manual

Table of Contents

[1 Introduction and Overview 3](#_Toc518236061)

[2 Licensing 3](#_Toc518236062)

[3 Use of Code Signing Key 3](#_Toc518236063)

[4 Tool Set Architecture 4](#_Toc518236064)

[5 Tool Set Requirements and Goals 4](#_Toc518236065)

[5.1 Overview of Requirements and Goals 4](#_Toc518236066)

[5.2 Explanation of Individual Requirements and Goals 5](#_Toc518236067)

[6 Supported Platforms and Build Variants 5](#_Toc518236068)

[7 Build Instructions 7](#_Toc518236069)

[8 Coding Standards 7](#_Toc518236070)

[9 Design Standards 7](#_Toc518236071)

[10 Testing Standards, and Testing 7](#_Toc518236072)

[11 Index 8](#_Toc518236073)

# 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

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.

# Use of Code Signing Key

The use of a code signing key would require the expenditure of a few hundred dollars a year, which seems like an unnecessary expenditure.

Software is released on a website with a cryptographic hash. The assumption is that no attacker would be able to modify the cryptographic hash published on the website.

Code signing may be considered in the future.

# Tool Set Architecture

The general architecture of the tool set is described before the requirements (§5). This presentation order allows the architecture to be traced to the requirements in §5.

## General Organization



Figure : Tool Set Architecture

* Shared source code (Figure 1).
  + The tool set uses a great deal of shared source code. This source code is potentially shared between all tools.
  + The shared source code may take the following forms (TBD):
    - DLL’s are not permitted, for the reasons explained in §5. Shared source code must be statically linked into each executable.
    - Libraries (or possibly a single library), that is compiled separately from any tool and linked in to each tool.
    - Both approaches would hopefully be used in such a way as to minimize the bulk introduced in each executable. (Libraries may in some cases be better in this regard, as they can be arranged so that the minimum unit incorporated into the executable is the function rather than the module.)
* The tool set consists of a number of tools (in Figure 1, Tool1, Tool2, ToolN).
  + Each tool is logically cohesive and performs a fairly narrow function. Examples:
    - Analysis and reduction of Boolean functions.
    - Generation of code to implement state machines.
    - Analysis and generation of code to perform specialized approximations.
  + Each tool potentially exists in some combination of the following three forms:
    - A standalone text-based console-mode executable.
    - A GUI-based tool.
      * The GUI-based tool is possibly bundled as part of a tool-collection. If not, it might be considered a tool collection with one tool.
    - Command[s] built into the scripting language.
* Each tool may have source code shared between the three forms (GUI, Cmdline, Scripted).
  + Because the three forms are so close in functionality, there would be shared source code unique to the tool.
* Data storage for each tool.
  + When tools need to store files, they should be:
    - A regular text language, friendly to humans (not XML!).
    - Designed so that small changes in whatever is manipulated by the tool result in small changes to the file (so that a human can figure out what has changed using a diff tool).
    - Should version control well, with minimal false positive changes.
      * Minimizes delta size with many version control tools.
      * Makes it easy for a human to figure out what has changed.
      * Indentation and trivial changes need to be controlled by the definition of the language to prevent false positive changes.
    - The regular language should have a formal parser that can determine:
      * Membership in the language.
      * Errors or warnings.
      * This will facilitate hand edits.
* Data interchange between tools.
  + GUI:
  + Cmdline:
  + Scripted:
* 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.

# Tool Set Requirements and Goals

## Overview of Requirements and Goals

All requirements and goals are listed immediately below, with further explanation and discussion provided in §4.2.

* License.
  + Availability of source code.
  + Lack of obligation to make source code changes publicly available.
* Source code.
  + Availability.
  + Documentation.
  + Lack of obligation to make source code changes publicly available.
* Verification of Computing Environment.
  + Ability to verify identity of all non-system components.
  + Ability to self-verify.
* Use of all available platform memory.
* Use of all available platform computing resources.
* Scriptability
  + Integration with a Turing-complete scripting language.
  + Similarity of scripting language to C.
* Extensibility.
  + Ability to add and integrate custom tools.
  + Ability to add and integrate built-in commands to scripting language.
* Cross-platform usability.

## Explanation of Individual Requirements and Goals

# 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)