**Embedded Tool Set (ETS)**

User Manual  
Software Engineering Manual

David T. Ashley ([dashley@gmail.com](mailto:dashley@gmail.com))

Table of Contents

[1 Introduction and Overview 3](#_Toc519969146)

[1.1 Overview of This Document 3](#_Toc519969147)

[1.2 Overview of *ETS* 3](#_Toc519969148)

[1.3 Information Contained in This Document 4](#_Toc519969149)

[2 Licensing 4](#_Toc519969150)

[2.1 Licensing for Source Code, Binaries, and Installation Packages 4](#_Toc519969151)

[2.2 Licensing for Use of Web-Based Tools 4](#_Toc519969152)

[2.3 Licensing for Software Used Within the Embedded Tool Set 4](#_Toc519969153)

[2.3.1 GMP (GNU Multiple Precision Arithmetic Library) 4](#_Toc519969154)

[3 Tool Set Architecture 5](#_Toc519969155)

[3.1 General Organization 5](#_Toc519969156)

[4 Tool Set Requirements and Goals 7](#_Toc519969157)

[4.1 Overview of Requirements and Goals 7](#_Toc519969158)

[4.2 Explanation of Individual Requirements and Goals 7](#_Toc519969159)

[5 Supported Platforms and Build Variants 7](#_Toc519969160)

[6 Build Instructions 10](#_Toc519969161)

[7 Coding Standards 10](#_Toc519969162)

[8 Design Standards 10](#_Toc519969163)

[9 Testing Standards, and Testing 10](#_Toc519969164)

[10 Procedures and Checklists 10](#_Toc519969165)

[10.1 Creating Dave Ashley’s Home Server from Stock Linux Distribution 10](#_Toc519969166)

[10.2 Moving Directories Under SVN on \*nix Server 11](#_Toc519969167)

[10.3 Setting Attributes for Keyword Expansion and EOL on \*nix Server 11](#_Toc519969168)

[11 Topics to be Filed 11](#_Toc519969169)

[11.1 Use of Code Signing Key 11](#_Toc519969170)

[12 Index 12](#_Toc519969171)

# Introduction and Overview

## Overview of This Document

The *Embedded Tool Set* (or *ETS*) is a collection of open-source software tools that are designed 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 is designed to accommodate both *users* of the tool set (those that usually do not use the source code, and only desire to know how to use the tool set), and *developers* (those that desire to extend or modify the tool set, do use the source code, and desire to know how the tool set works internally).

The user manual (which would normally be intended for users) and the software engineering manual (which would normally be intended for developers) are combined; because if they were separate documents, there would be a great amount of duplicated information.

## Overview of *ETS*

*ETS* is designed for high integrity (with specific features described in §3). (Because a defect in ETS can introduce a defect in a product, high integrity is a requirement.)

ETS functionality is arranged hierarchically in 4 levels:

* Tool Set (TS). (The entire ETS tool set.)
* Tool Collection (TC). (A set of logical tools that are packaged together in one executable.)
* Tool (TL). (A set of functions that have related functionality: for example, analyzing a first-order filter and designing a first-order filter.)
* Tool Function (TF). (An individual function within a tool.)

Conceptually,[[1]](#footnote-1) each tool function accepts a regular language as input and produces a regular language as output. Tools and tool functions are packaged within the tool set in 4 ways:

* As built-in functions for a script interpreter.[[2]](#footnote-2) (The script interpreter can process built-in functions both interactively and as part of a script.
* As GUI interfaces within executables.
* As stand-alone console-mode executables that implement specific sets of tools or tool functions.
* As functionality implemented by a web page.

Every effort has been made to keep naming and functionality consistent among the 4 packaging methods. Consistency helps to:

* Facilitate ease of use. (The same names and input and output formats are shared among all packaging styles.)
* Minimize source code bulk. (The same source code is used to build all 4 packaging styles.)

## Information Contained in This Document

TBD.

# Licensing

## Licensing for Source Code, Binaries, and Installation Packages

The *Embedded 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.

## Licensing for Use of Web-Based Tools

The licensing needs to include a strong browse-wrap disclaimer system.

## Licensing for Software Used Within the Embedded Tool Set

### GMP (GNU Multiple Precision Arithmetic Library)

The GMP (provided under the LGPL) is used:

* Modified in PC applications.
  + Must provide modified source.
  + Must not prevent users from modifying that portion of the source code further.
* Unmodified in server applications.
  + No obligations.
  + May provide the source as a courtesy, in case user has difficulty obtaining it any other way.

# Tool Set Architecture

The general architecture of the tool set is described before the requirements (§4). This presentation order allows the architecture to be traced to the requirements in §4.

## 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 §4. 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 : C/C++ Build Parameterization

|  |  |
| --- | --- |
| **PREPROCESSOR CONSTANT** | **DESCRIPTION** |
| **Platform (K\_ETS\_PF) (All #define’s below mutually exclusive)** | |
| K\_ETS\_PF\_WINAPI | Windows API (also sometimes called Win32, although this a misnomer because 64-bit programs can also use the Win32 API). |
| K\_ETS\_PF\_MFC | Program uses the Windows API with the MFC. |
| K\_ETS\_PF\_WIN\_NET | Windows .NET. |
| K\_ETS\_PF\_UNIX | Unix. |
| K\_ETS\_PF\_LINUX | Linix. |
| K\_ETS\_PF\_FREE\_BSD | Free BSD. |
| K\_ETS\_PF\_CYGWIN | Cygwin. |
| K\_ETS\_PF\_ANDROID | Android. |
| K\_ETS\_PF\_FIRE\_OS | Fire OS. |
| K\_ETS\_PF\_IOS | iOS. |
| **Operating System Word Size (K\_ETS\_WSZ\_OS)** | |
| K\_ETS\_WSZ\_OS | Should be set to the number of bits that characterize the operating system. In the case of Windows/x86, this differentiates 32-bit vs. 64-bit Windows. |
| **Processor for Assembly-Language (K\_ETS\_PROC\_AL) (All #define’s below mutually exclusive)** | |
| K\_ETS\_PROC\_AL\_X86 | Intel 8086 |
| **Machine Word Size (K\_ETS\_WSZ\_M)**  **(Note: Machine word size does not imply C or C++ default integer size.)** | |
| K\_ETS\_WSZ\_M | Set to the number of bits in a native machine word. |
| **Integer Size (K\_ETS\_SZ\_INT\_LL)** | |
| K\_ETS\_SZ\_INT\_LL | Set to the number of bits in a C long long integer. |
| **Machine Word Size (K\_ETS\_SZ\_INT\_L)** | |
| K\_ETS\_SZ\_INT\_L | Set to the number of bits in a C long integer. |
| **Machine Word Size (K\_ETS\_SZ\_INT)** | |
| K\_ETS\_SZ\_INT | Set to the number of bits in a C integer. |
| **Machine Word Size (K\_ETS\_SZ\_INT\_S)** | |
| K\_ETS\_SZ\_INT\_S | Set to the number of bits in a C short integer. |
| **Machine Word Size (K\_ETS\_MWS)**  **(Note: machine word size does not imply C or C++ default integer size.)** | |
| K\_ETS\_MWS | Should be set to the number of bits in a native machine word. |
| K\_ETS\_MWS | Should be set to the number of bits in a native machine word. |
| **Machine Integer Representation (K\_ETS\_MIR) (All #define’s below mutually exclusive)** | |
| K\_ETS\_MIR\_2SCOMP | Integers have traditional 2’s complement representation. (This allows many programming optimizations.) |
| K\_ETS\_MIR\_SIGNMAG | Integers have sign-magnitude representation. |
| K\_ETS\_MIR\_OTHER | Integers have another representation. |
| **Machine Floating Point Unit (K\_ETS\_MFPU) (All #define’s below mutually exclusive)** | |
| K\_ETS\_MFPU\_NO | Hardware does not have a floating-point processor, and floating-point operations are done in software (relatively slow). |
| K\_ETS\_MFPU\_YES | Hardware does have a floating-point processor, and floating-point operations are done in hardware (very quick). |
| **Project Type (K\_ETS\_PROJTYPE) (All #define’s below mutually exclusive)** | |
| K\_ETS\_PROJTYPE\_APP | Project is an application. |
| K\_ETS\_PROJTYPE\_LIB | Project is a library. |
| K\_ETS\_PROJTYPE\_OBJ | Project is an object file or set of object files. |
| **Project Link Type (K\_ETS\_LINKTYPE) (All #define’s below mutually exclusive)** | |
| K\_ETS\_LINKTYPE\_STATIC | Project will be linked statically. |
| K\_ETS\_PROGTYPE\_DYNAMIC | Project will be linked dynamically. |
| **Project Debug Level Type (K\_ETS\_DEBUGLVL) (All #define’s below mutually exclusive)** | |
| K\_ETS\_DEBUGLVL\_DEBUG | Project is intended for debugging or analysis. |
| K\_ETS\_DEBUGLVL\_RELEASE | Project is intended for release. |
| **Program Type (K\_ETS\_PROGTYPE) (All #define’s below mutually exclusive)** | |
| K\_ETS\_PROGTYPE\_CONSOLE | Program is a console-mode utility (text input, text output). |
| K\_ETS\_PROGTYPE\_WINGUI | Program is a graphical program under Windows. |
| K\_ETS\_PROGTYPE\_TCL\_A\_CONSOLE | Program is a Tcl console-mode utility, using Tcl code ported by Dave Ashley around 2004. |
| K\_ETS\_PROGTYPE\_TCL\_A\_GUI | Program is a Tcl/Tk graphical utility, using Tcl/Tk code ported by Dave Ashley around 2004. |
| K\_ETS\_PROGTYPE\_TCL\_B\_CONSOLE | Placeholder for future console port of Tcl. |
| K\_ETS\_PROGTYPE\_TCL\_B\_GUI | Placeholder for future graphical port of Tcl/Tk. |
| K\_ETS\_PROGTYPE\_CLIKE\_A\_CONSOLE | Placeholder for future console application involving “Clike” (a yet-to-be-developed C-like scripting language). |
| K\_ETS\_PROGTYPE\_CLIKE\_A\_GUI | Placeholder for future graphical application involving “Clike” (a yet-to-be-developed C-like scripting language). |
| K\_ETS\_PROGTYPE\_UNIX\_SWING | Program developed using Unix Swing. |
| K\_ETS\_PROGTYPE\_UNIX\_AWT | Program developed using Unix AWT. |
| K\_ETS\_PROGTYPE\_CGIBIN\_HELPER | Program is invoked by CGI-BIN PHP, Python, or Perl scripts to implement functionality awkward under the scripting language. |
| K\_ETS\_PROGTYPE\_CGIBIN\_HTTPD | Program is a CGI-BIN program invoked directly by httpd to answer HTTP[S] requests. |
| K\_ETS\_PROGTYPE\_CGIBIN\_SERVER | Program listens on TCP ports and is an actual HTTP[S] server. |
| K\_ETS\_PROGTYPE\_UNITTEST\_MODULE | Program is a unit test program compiled to test an individual software module. |
| **Screen Size (K\_ETS\_SCREENSIZE) (All #define’s below mutually exclusive)** | |
| K\_ETS\_SCREENSIZE\_SMALL | The target screen size is small (such as a cellphone). |
| K\_ETS\_SCREENSIZE\_LARGE | The target screen size is large (such as a tablet computer or computer). |
| K\_ETS\_SCREENSIZE\_ADAPTIVE | The program adapts to the screen size. |
| **Threadedness (K\_ETS\_THREADS) (All #define’s below mutually exclusive)** | |
| K\_ETS\_THREADS\_1R | The program runs with one thread, a greatly reduced priority (essentially, a background program). |
| K\_ETS\_THREADS\_1 | The program runs with one thread, at unmodified priority. |
| K\_ETS\_THREADS\_2 | The program runs with two threads, at unmodified priority. |
| K\_ETS\_THREADS\_3 | The program runs with three threads, at unmodified priority. |
| K\_ETS\_THREADS\_4 | The program runs with four threads, at unmodified priority. |
| K\_ETS\_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. |
| K\_ETS\_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. |
| K\_ETS\_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.

# Procedures and Checklists

## Creating Dave Ashley’s Home Server from Stock Linux Distribution

Need to include the step of modifying http.conf so that .svn contents not served.

[dashley\_wm@spock ~]$ diff /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf~

225,231d224

< # The following lines prevent access to .svn directory internals. This is relevant

< # because some web content is served directly out of an SVN sandbox.

< <DirectoryMatch .\*\.svn/.\*>

< Deny From All

< </DirectoryMatch>

<

< #

## Moving Directories Under SVN on \*nix Server

## Setting Attributes for Keyword Expansion and EOL on \*nix Server

# Topics to be Filed

## 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.

# Index

David T. Ashley’s Tool Set, 3

DTATS, 3

testing, 3

1. Within the CLIKE interpreter, for efficiency, conversions between string and other representations are performed only when necessary. For example, “*$y=2\*sqrt($x)*” will not result in a string representation of “*sqrt($y)*” being calculated before the multiplication by “*2*”. [↑](#footnote-ref-1)
2. The script interpreter interprets a language called CLIKE, described fully in §TBD. The interpreter can be used both interactively (in this mode it might be described as a *very* powerful calculator) and to run scripts (which give programmatic access to all ETS functionality). [↑](#footnote-ref-2)