

# Extended Basic Tools (EBT)

## User Manual

Version 1.0a 24 Aug 2020  
(Record of Revisions located in Appendix 1)

Copyright 2020 TOD A. WULFF

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.

## Introduction to EBT

Extended Basic Tools (EBT) is a set of extensions for the [Coridium BASICtools IDE](#) which Coridium Corporation provides for free when used with any of the [Coridium SBC Boards](#). EBT is provided for free and relies on the existence of BASICtools. Without BASICtools, EBT is, by design, materially non-functional. The scope of this document is related to EBT and it is not intended to be a User Manual for Coridium's BASICtools IDE (BT). This work assumes that one has a basic to good understanding of BT, ARMBasic, and how to use each on SBCs that have ARMBasic Firmware thereon.

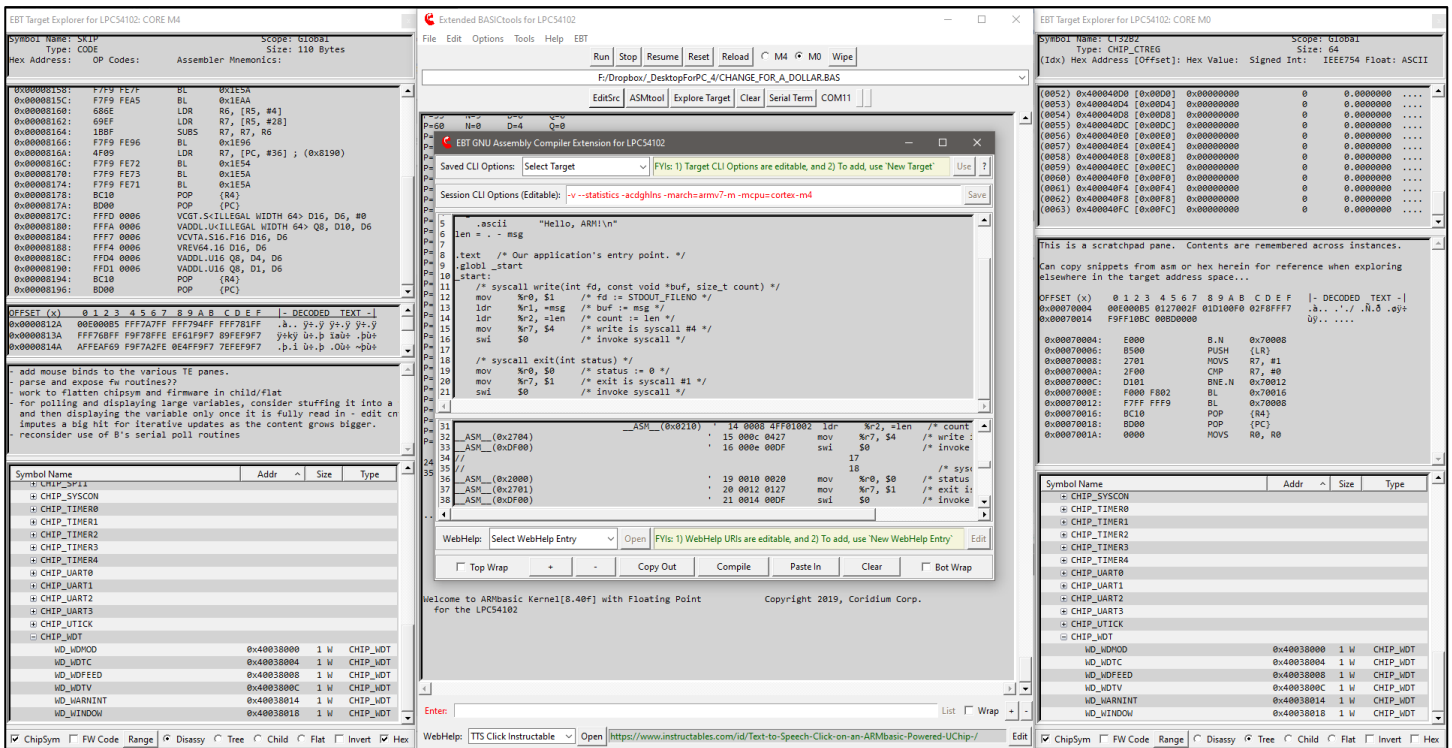


Figure 1: Extended BASICtools with Dual Target Explorers (one instance per core) and ASMtool Modules activated

Good day. My name is Tod Wulff, an aerospace & defense professional that is also a bit of a geek at heart.

Hailing from the era of dial-up BBS, 8-bit Microcontrollers, Kaypro/Commodore/Tandy/TI-994A personal computers, when Radio Shack stores were plentiful (the good ole days), one of my first hobby embedded projects was working with a MEK6800D2 Motorola Microprocessor Training Kit, which I had purchased while working as a co-op at the MSU EE labs (after completing my High School Electronics VoTech training in Southern Lower MI). That project involved my prototyping the Radio Shack SP0256 NARRATOR™ SPEECH PROCESSOR onto the MEK6800D2, wiring it up and programming the 6800 to get it to emit pseudo-speech (those who've worked with the SP0256 based HW know exactly what I am alluding to). It worked wonderfully and I progressed down the path of cutting my teeth in embedded microcontrollers and Assembly. After High School, life got in the way, Military, War, Spouses, Children, entering the civil sector, starting a career, etc. all added up to my shelving my hobby in favor of pursuing the endeavors of life in a Western culture (here in the US).

Skip forward 20 years, coming to the inevitable time where the kids are maturing to the point that the Bride and I are severely distracting to their social status (and life in general), the mortgage, vehicles, college bills are slowly being paid down, earnings getting better with advancements, and my having enough spare time to the point where I could start to refocus on some selfish interests, I picked back up on the hobby electronics gig. Anyways, given my lineage and history, I sought out and found a dev environ that I quickly bonded with - ARMBasic - BASIC was my first love and this fit the bill of not only reacclimating myself to programming, but working with hardware that was vastly more powerful than what I had started with decades earlier, and thus the journey began. Having done some windows scripting for a couple years prior, using AutoHotKey (AHK), I decided to try to integrate a home-brewed AHK-based IDE, intended for use with early Coridium goods, into Notepad++, my editor of choice back then (remaining so thus far). That endeavor was only partially successful.

This was circa 2006-2009. Then, for reasons well beyond our control, life changed (as it had for many during those years – housing crisis). Hobbies shelved - focus on a new career, recovering from financial struggles (was heavily vested in the real-estate domain and we took it in the shorts and the youngins were just getting to the point where College funding was an imperative). Basically, Life and First-World problems (we are really blessed, considering the challenges and toils that folks in other parts of the world struggle with on a daily basis) manifested themselves and ... the hobby got shelved. I picked back up briefly on it in 2011-2012-ish then was met with another career change - hobby shelved yet again.

Fast forward another decade and ... I am back and, Good Lord Willing, hopefully for the duration (until I take that proverbial dirt nap and start pushing daisies up from below). So, here we are. Wow - Arduino (what is that weird word?) had stormed the market. Makers?? What the heck are they?! ... :) My friends at Coridium Corp (proprietors of ARMBasic and ARM-based microcontroller dev boards) had remained steadfast and true. Now, instead of the LPC2xxx series of controllers, there is this new (to me) entity of ARM, and Cortex M0/M3/M4, and Arduino, and ... WOW! The culture has morphed quite a bit, and in many a great way. Peeps are collaborating remotely and, indeed, globally. Hardware is getting amazingly fast and powerful, and ARMBasic, having matured and steadfastly hardened with employment across many different families of silicon, is a thing of beauty for me and many others.

EBT is, comparatively, somewhat recently birthed, with initial code being drafted in November 2018, and a mature alpha coming into existence by February 2019. Further development to EBT was sporadic until July 2019 when I was tasked professionally on a temporary duty assignment that ended up being 9 months long (so much for 'temporary'...). Returning home in April/May 2020, EBT dev was resumed slowly and continues into today, and the foreseeable future.

One initial intent for EBT (not being called such when conceived, but rather 'Tweaked BT' [sigh]) was to provide a means to have a workflow that somewhat automated the inclusion of ARM Assembly constructs into an ARMBasic user app. The next extension was to change the default color scheme from white to gray, to match how Notepad++ was set up, to provide visual continuity, if you will.

Very quickly, it took on a life of its own and has been updated over the course of the last year and a half, pseudo-continuously. The initial roots of EBT can be found in the ASMtool module within EBT. Being an Android junkie and somewhat spoiled with customization/theming, coupled with being picky about how the tools used on a regular basis are customizable to one's liking, to include workflow aspects that conform to what is perceived as one's needs or desires, caused EBT to morph into what is perceived as a substantive set of extensions to BT.

Note: Overt care to not confuse 'Extensions' with 'Enhancements' is employed. The former is an appropriate descriptor, while using the latter to describe EBT would be presumptively inappropriate as it is, admittedly, a quite subjective term...

Ok, 'nuf with the diatribe - the purpose of this document is to explain EBT and its use - let's get to it.

The best way to see and begin to understand EBT is to 'install' it and use it. Given it is an extension to an existing toolset, there are prerequisites needing to be fulfilled in order to make use of same.

# EBT Prerequisites

The first requirement is to have BASICtools (BT) for ARM installed on an x86 32-/64-bit Windows dev box. BT can be download from [here](#). An archive of the current (at time of drafting) complement of EBT files can be downloaded from [here](#). This User Manual ([online .pdf version](#)) is also linked to in EBT's WebHelp MiniTool, detailed later in this document.

**NOTE:** At its root BT is, and by inclusion EBT is also, a set of TCL scripts that provide a UI framework and, when coupled with compilers, disassemblers, etc., comprise a complete and extended ARMBasic Integrated Development Environment. TCL is a powerful cross-platform interpreted scripting language. Sadly, not being a Linux or iOS type, no effort or assertions are made regarding EBT's functionality when installed on a machine that is not a Windows box. For those users who are on Windows boxes, assertions that it *may* work are made (see license/warranty on first page). For \*nix-based OSes, EBT's author is not personally interested in progressing through the curve to ensure that EBT works with a BT installation on those machines. However, there is no opposition to someone taking the time to check and advise if EBT works thereon or not. If code changes/additions are needs to enable EBT to work on same, and if you, the reader of this document, are able and willing to tackle modifying EBT's set of scripts to work on Linux or iOS dev boxes, it is endorsed and will be supported, as long as the effort is reasonable and not too disruptive to other endeavors (Selfish? Maybe. | Practical? Absolutely.).

Once BT-proper is installed, the next step is to get the current build of EBT downloaded (link above) and extract same (preserving directory structure) into the Coridium BASICtools directory (often located at %ProgramFilesx86%/Coridium/ (the default offered by the installer)). Administrator approval will likely be queried for by the OS.

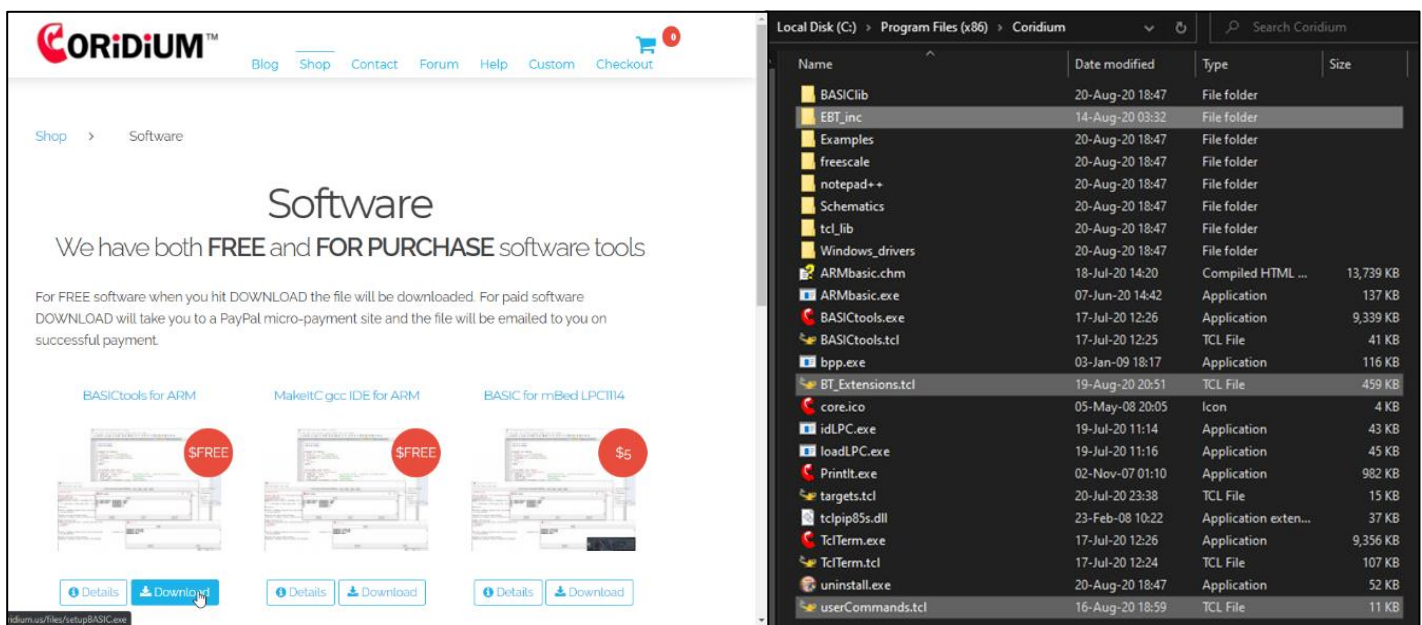


Figure 2: BASICtools Download Page - EBT files extracted into Coridium folder

EBT consists of two .tcl scripts in the root directory of the Coridium Install – BT\_Extensions.tcl and userCommands.tcl. Additionally, there are a number of utility binaries, support scripts, and other support files of varying types included in the EBT archive. These other files, when extracted to the Coridium folder, should end up in the EBT\_inc subdirectories.

The userCommands.tcl script is organically 'sourced' in by BT, when it is started. The original intent was to enable BT's user devs to have custom commands be intercepted from the embedded target and be able to have the IDE/host dev box programmatically react to same. An example is either clearing the BT console or responding to a BELL ASCII character being sent by the target to cause the host system to emit sounds through the PC speaker.

This capability of being able to run customized user drafted .tcl scripts at BT's startup was latched onto and heavily employed to Extend BT's core capabilities and UI. In a nutshell, userCommands.tcl checks if an BT .ini saved setting reflect that EBT is turned on and, if so, launches BT with the EBT extensions enabled. If the EBT .ini is missing (i.e. first run), or the enablement of EBT is set to 'off', then BT-proper is launched and EBT is not started, enabling a stock Coridium BASICtools IDE experience.

Speaking of the .ini settings file. BT and EBT each have their own .ini. This was done to help ensure that if EBT is disabled, that there is the smallest possibility of EBT impeding the operation of BT. This philosophy was carried forward in EBT in that EBT doesn't materially alter the BT-proper menus/functionality, but rather has its own menu entry/hierarchy and functionality. The intent is to have EBT extend BT's capabilities, never impeding and keeping material alteration of same to a minimum. Extension of, not enhancement to, is the theme here.

And while we're talking about other directories, as additional information, BT's configuration .ini file is stored in %AppData%/Roaming/Coridium/ (EBT's is in the EBTini subfolder therein). Temp files, used during various phases of the tool chain's endeavors, possibly being touched on later herein, are stored in %AppData%/Local/Temp/Coridium/.

Once BT is installed, and EBT's archive is extracted into the Coridium program folder, BT should be started. Once BT is started, to start EBT, one would click on the **Tools/Debug/Extended Debug** menu option that Coridium was kind enough to code into BT. If things are as they should be, BT will source EBT's scripts and one will be met with EBT's default UI.

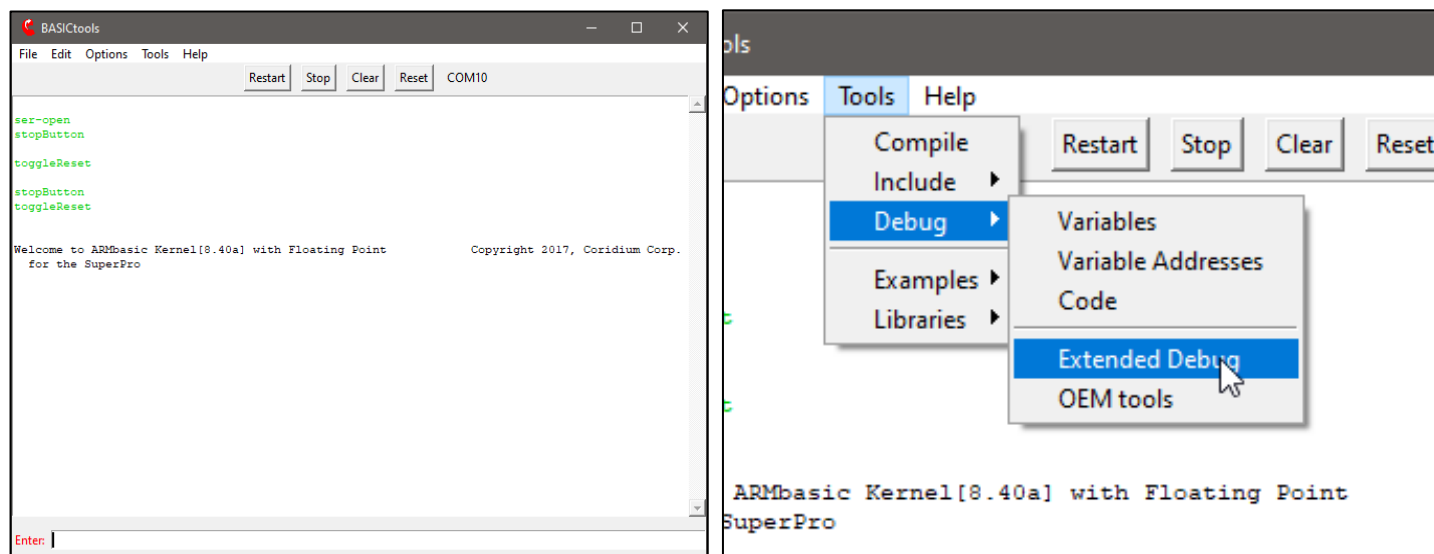


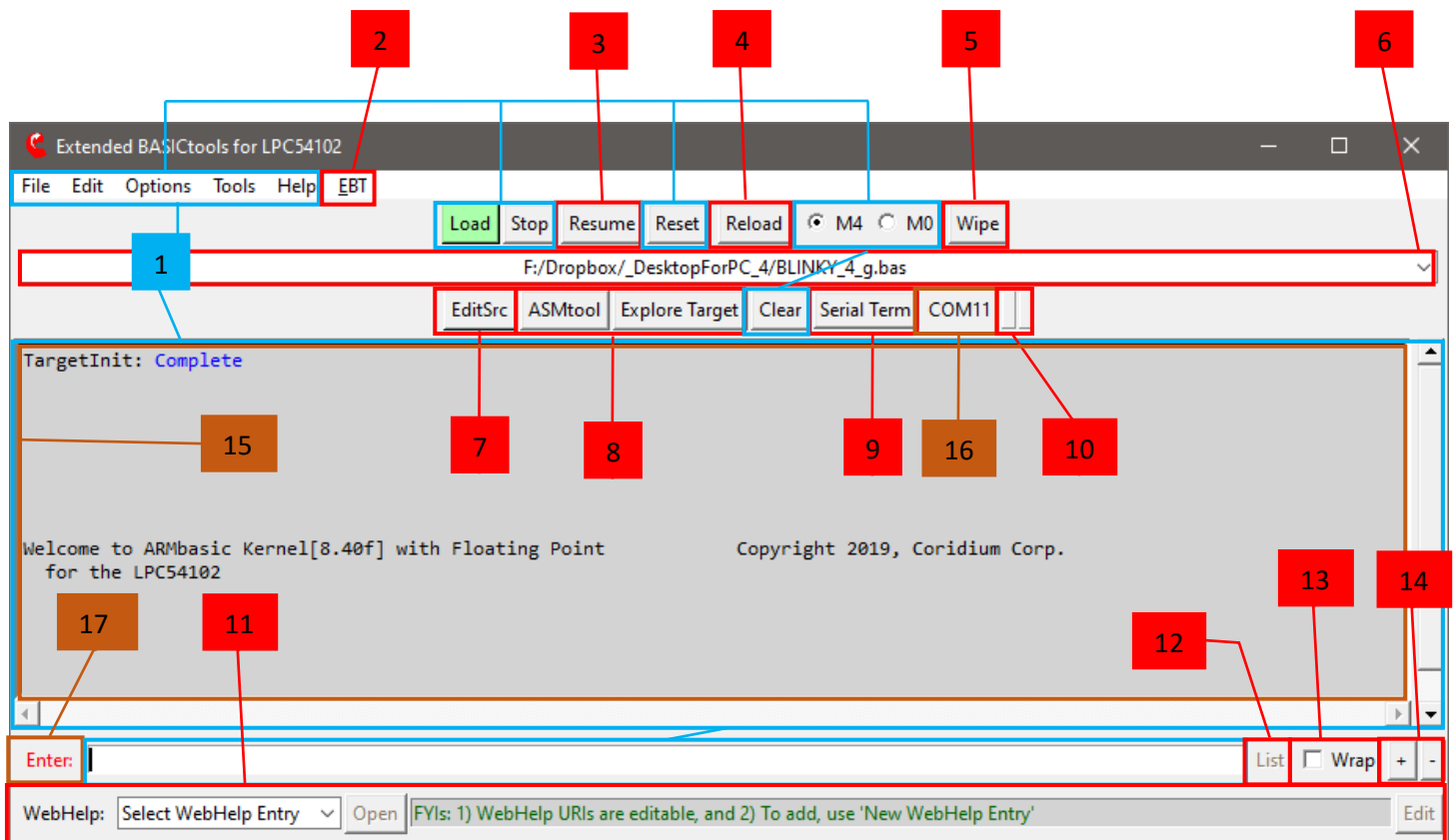
Figure 3: BASICtools Default UI – Starting EBT from BT proper

To further explain EBT functionality herein, screen shots and descriptive text will be used to illustrate the various aspects of using EBT. Also, be advised that there are two 'submodules' that EBT offers for dev's use – ASMtool and TargetExplorer (TE). Both of those tools offer substantive functionality which warrant their own addendums to this EBT User Manual. Those will be drafted and released as supplements hereto as quick as practical.

With that, let's get down to capturing tool images and explaining as much as practical about the feature sets and functionality of EBT's extensions to BASICtools.

## EBT UI ELEMENTS – Existing, New, and Extended

The following image/table depict the elements that comprise the main EBT UI and provide details regarding the controls, their functions, and if functional differences which may exist between themselves and the BT-proper counterparts (if any).



**Figure 4: Extended BASICtools Default UI**

EXISTING BT UI W/ MINOR OR NO CHANGES		EBT ELEMENT W/ NEW FUNCTIONALITY	BT ELEMENT W/ EXTENDED FEATURES
ITEM	DESCRIPTION		
1	Existing <b>Stock BT UI elements</b> with functions that are largely unfettered (color/font/label/content excepted :-)		
2	<b>EBT Menu</b> – will be detailed on dedicated pages herein		
3	<b>Resume</b> Button – simulates the user typing a carrot (^) – useful after a ‘STOP’ debug command		
4	<b>Reload</b> Button – mirrors the BT Reload menu function, with the User App in the DDL (#6) being used		
5	Core <b>Wipe</b> Button – to overtly erase ARMbasic user app from the MCU – on Core M0, codes an endless loop		
6	Historized <b>Recent</b> User App Drop Down List – similar to Recent Files in BT File menu		
7	<b>Edit</b> Source File Button – opens the DDL (#6) selection in the editor configured during BT setup		
8	<b>ASMTtool</b> and <b>TargetExplorer</b> Modules – each of which to be detailed in dedicated manual addendums hereto		
9	<b>Serial Terminal</b> Button – opens a TclTerm stand-alone serial terminal (for debug/monitor)		
10	Serial Traffic ‘LEDs’ – <b>Red</b> = Host Tx (lt), <b>Green</b> = Host Rx (rt), are also buttons to unlock UI elements if needed		
11	<b>WebHelp</b> MiniTool – will be detailed on dedicated pages herein		
12	<b>List</b> Button – becomes enabled with iterative compilation activities, used to list the WIP User App code		
13	<b>Wrap</b> Toggle – to turn on and off wrapping of text in the console		
14	Increase(+)/Decrease(-) Font Size – for real-time adjustment of BT/EBT UI element text		
15	BT/EBT Target <b>Console</b> – this is the drag and drop target for user AB apps ( <i>see wrapped scripts dialog @ end</i> )		
16	BT/EBT <b>COMxx</b> Port Indicator – in EBT is also a button that will re-enumerate the current serial instance		
17	BT/EBT <b>Enter</b> Label – in EBT this is also a button to transmit a carriage return out the serial port		

**Table 1: Extended BASICtools UI Elements Index**

# EBT Menu

EBT's menu allow the user to access and/or control various aspects of EBT's functionality:

- Preprocessor Selection and various Preprocessor Settings for the selected EBT Preprocessor - FilePP.
- Menu selectable short-cuts to relevant explorer folders on the host dev box.
- Menu selectable short-cuts to various intermediate files generated by the tool chain during compilation.
- Control of various options that control EBT's UI, 'Windowing' on host OS, and functional behavior.
- Cessation of EBT with Restoration of stock BT functionality (imputes an automatic restart of the toolchain).
- On-demand Instantiation of TclTerm as an additional serial console (useful at times for debug/monitoring).
- Access to a composite pre-processed ARMBasic (AB) Source File – the toolchain's input to the AB compiler.

Each of these will be touched on, some in granular detail where appropriate, in the following dialog and images.

## EBT Menu | EBT Source Code Preprocessors

BT-proper makes use of a customized version of CPP, which functions in a manner similar to CPP, and is called BPP, presumably an acronym for BASICpreprocessor (a reasonable assumption never thought to have affirmed). BPP is, by design and lineage, pretty focused at one thing, preprocessing source-code files with a rigid structure of functionality and capability. It is not intended to be expandable and is pretty limited in a myriad of other ways, as it was designed to do just one thing. Don't mistake these comments as non-appreciation. It does perform its designed functions pretty darn well. However, when one starts to request/expect CPP/BPP to do more, the wheels can depart the bus pretty quickly.

These limitations (was and still is perceive as such) was the initiating factor that caused the hunt for other preprocessors to begin. It didn't take too long to find FilePP. After testing FilePP, and eventually realizing just how extremely powerful such an extendible preprocessor solution was, it became evident that FilePP needed to become an EBT team member, and was integrated as one of the key extensions that EBT brought to the table for a dev to leverage in any AB dev effort.

It is acknowledged that, for the vast majority of AB dev work, BT's Stock BPP is more than adequate at preprocessing AB source file, doing so really well. However, it would be appropriate to acknowledge that there are some specific feature sets that FilePP empowers an AB dev with that simply cannot be accomplished with the BPP tool as it exists today.

While in-depth details about what FilePP does, how it does it, and how to get it to do it, is well beyond the scope of this manual, the following listed items are some of FilePP's features that see regular employment by EBT's dev:

- |                                   |  |
|-----------------------------------|--|
| • Compile-time Maths              | • Multi-Line Macros                                |
| • Multi-Level Preprocessing Debug | • Selectable Boundary Macro Expansion              |
| • C comment Removal               | • Inner Literal Macro Expansion                    |
| • AB Comment Removal              | • Regex-based Macro Definition & Expansion         |
| • For-Next Loop Preprocessing     | • Multi-file Macro Definition, Build-up, Expansion |

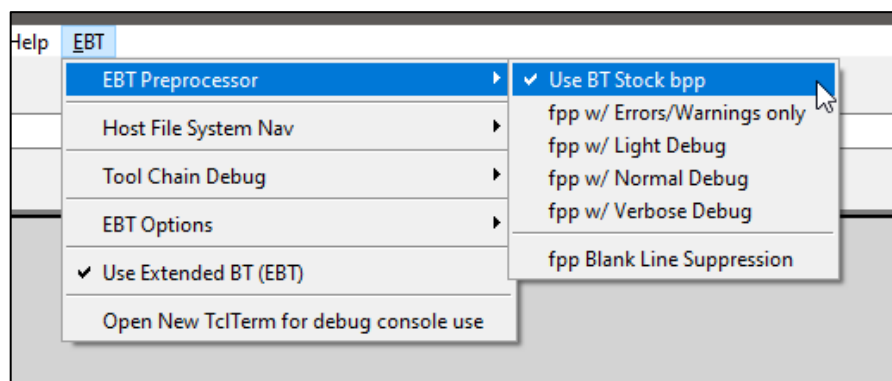


Figure 5: EBT Preprocessor Selection and Control



## EBT Menu | Selectable Windows Explorer folders short-cuts

It should be pretty self-evident what these various menu selections do. If one doesn't grasp this, one may be well served to consider taking up knitting or some other hobby (kidding – if you get wrapped around the axle on these, just ask in the forums and someone should be able to help).

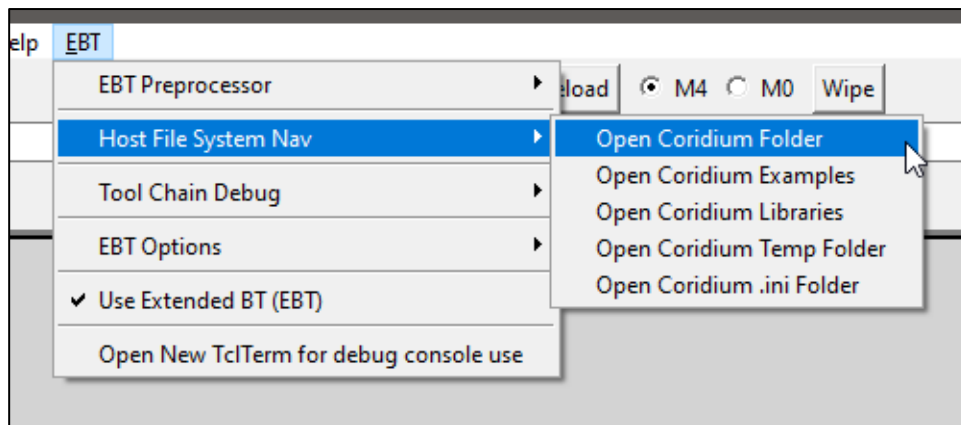


Figure 6: EBT Host File System Folder Short-cuts.

## EBT Menu | Tool-chain Intermediate File short-cuts

As with the previous, and for anyone familiar with a typical tool-chain process, extrapolating what these menu-selectable options do should be pretty self-evident. Again, if one gets wrapped around the axle on these, just ask in the forums and someone should be able to help.

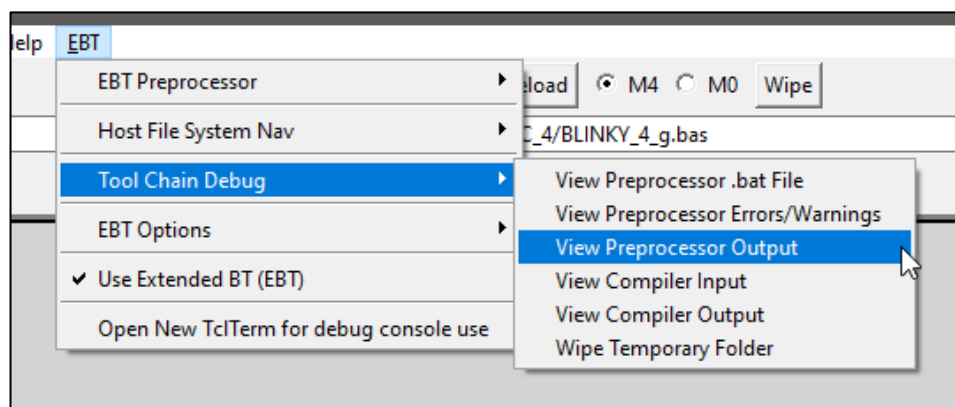


Figure 7: EBT Tool-chain Intermediate File Short-cuts.

## EBT Menu | EBT's UI Options, Host-OS 'Windowing', and Optional Functionality

BT and EBT have 'normal' windows on a Windows OS Host box. By normal, one is inferring that the window size is adjustable, has a title, has an icon on the task bar, and can be manipulated to be on top or not, as the case may be.

Windows supports having Modal windows or Tool Windows, where the windows have title bars, but lack some of the dressing and control that 'normal' windows have. One of these differences are that tool windows don't normally have icons on the Windows Task Bar (WTB). Turning on tool mode for the Target Explorer (TE) removes the WTB TE Icons. If one elects to have their WTB configured to always combine the icons thereon, if TE windows were normal windows, and one of them were minimized, when one clicked on the WTB icon, one would then be offered thumbnails of the windows to click on to select which window is desired to be brought to the top and focused. It can be frustrating to have to do so.

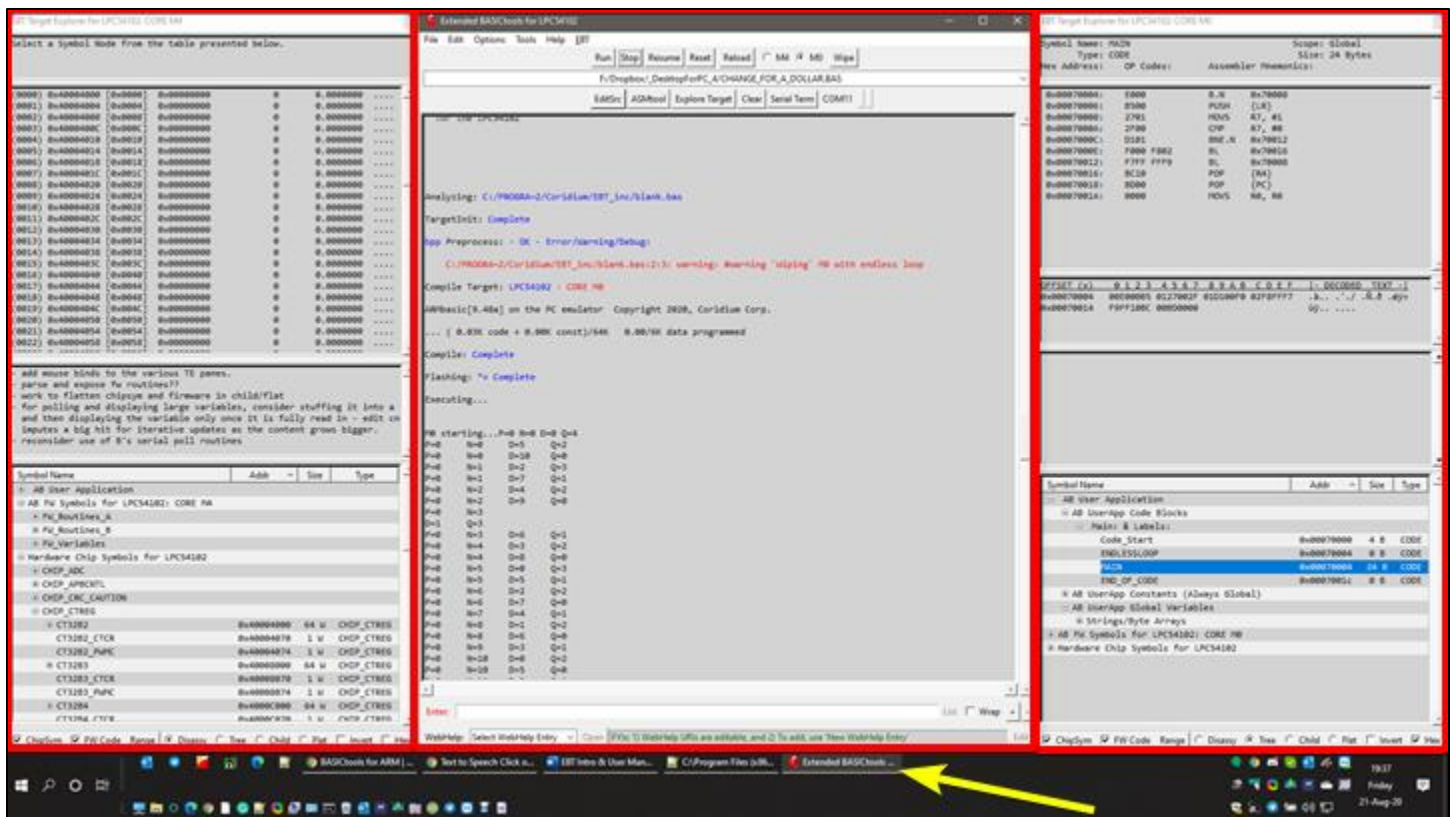


Figure 8: EBT Main UI Window with two TE 'Tool Windows' (note the single Task Bar Icon)

Thus, the option to make TE windows 'Tool' windows was coded into EBT. Additionally, it was determined that in some use cases, it might be desirable to have an option to force windows to keep either EBT's main window on top, or to be able to do so with the TE windows, regardless if they are 'normal' windows or tools windows. The options for indication and control of these window modes exists on the EBT Options menu.

With tool windows, typical observed behavior is to have tool windows minimize with the host-app's main window. The 'Minimize TE when EBT Minimized' option enables this behavior.

The 0x prefix option enables TE to display hex numbers with the non-standard 0x prefix notated onto a hex number. BASIC standard is often either \$ or &H (ARMbasic is &H). For some users, the 0x prefix either is visually easier to employ, or is a habit of muscle memory borne from working with other dev languages. This is why the 0x option exists.

EBT Target Explorer for LPC54102: CORE M0									
Symbol Name: CT32B2		Scope: Global		Type: CHIP_CTREG		Size: 64			
(Idx)	Hex Address	[Offset]	Hex Value:	Signed Int:	IEEE754	Float:	ASCII		
(0000)	8H40004000	[&H0000]	&H00000000	0	0.0000000	....			
(0001)	8H40004004	[&H0004]	&H00000000	0	0.0000000	....			
(0002)	8H40004008	[&H0008]	&H00000000	0	0.0000000	....			
(0003)	8H4000400C	[&H000C]	&H00000000	0	0.0000000	....			
(0004)	8H40004010	[&H0010]	&H00000000	0	0.0000000	....			
(0005)	8H40004014	[&H0014]	&H00000000	0	0.0000000	....			
(0006)	8H40004018	[&H0018]	&H00000000	0	0.0000000	....			
(0007)	8H4000401C	[&H001C]	&H00000000	0	0.0000000	....			
(0008)	8H40004020	[&H0020]	&H00000000	0	0.0000000	....			
(0009)	8H40004024	[&H0024]	&H00000000	0	0.0000000	....			
(0010)	8H40004028	[&H0028]	&H00000000	0	0.0000000	....			
(0011)	8H4000402C	[&H002C]	&H00000000	0	0.0000000	....			
(0012)	8H40004030	[&H0030]	&H00000000	0	0.0000000	....			

EBT Target Explorer for LPC54102: CORE M0									
Symbol Name: CT32B2		Scope: Global		Type: CHIP_CTREG		Size: 64			
(Idx)	Hex Address	[Offset]	Hex Value:	Signed Int:	IEEE754	Float:	ASCII		
(0000)	0x40004000	[0x0000]	0x00000000	0	0.0000000	....			
(0001)	0x40004004	[0x0004]	0x00000000	0	0.0000000	....			
(0002)	0x40004008	[0x0008]	0x00000000	0	0.0000000	....			
(0003)	0x4000400C	[0x000C]	0x00000000	0	0.0000000	....			
(0004)	0x40004010	[0x0010]	0x00000000	0	0.0000000	....			
(0005)	0x40004014	[0x0014]	0x00000000	0	0.0000000	....			
(0006)	0x40004018	[0x0018]	0x00000000	0	0.0000000	....			
(0007)	0x4000401C	[0x001C]	0x00000000	0	0.0000000	....			
(0008)	0x40004020	[0x0020]	0x00000000	0	0.0000000	....			
(0009)	0x40004024	[0x0024]	0x00000000	0	0.0000000	....			
(0010)	0x40004028	[0x0028]	0x00000000	0	0.0000000	....			
(0011)	0x4000402C	[0x002C]	0x00000000	0	0.0000000	....			
(0012)	0x40004030	[0x0030]	0x00000000	0	0.0000000	....			

Figure 9: Stock AB &H hex prefix (left) vs. EBT's Optional 0x hex prefix (right)

Another BT UI default behavior is to wait until the first time a target is programmed before it updates the UI to reflect said target, not only in the title bar, where it depicts the controller with an expressive verbose name, but also on the UI, if the unit is a multi-core device (i.e. LPC4330, LPC54102, etc.), where the MCU Core selection radio buttons come into existence.



EBT’s approach can be made to be a bit more aggressive in where it actively queries the target controller at startup and updates the UI accordingly. The reasoning for having EBT’s optionally behave be this way is two-fold:

- 1) It enables the UI elements to be updated at startup
  - a. admitted eye candy, but for those who have a bit of CDO (OCD in alphabetical order, as it should be...) in them, this can make for a lower-stress user experience, at the price of a bit of startup latency to EBT.
- 2) It enables EBT to perform some background operations related to the firmware options in TE.
  - a. Note: These FW options within TE are still being worked on and are NOT mature yet (hence the ‘placebo’).

Another way in which to customize EBT is to modify the fonts used on the UIs. This is accessed by the Choose EBT Font option in the EBT Options Menu. Selecting same pops up a font-picker UI. It is relevant to note that the Monospaced fonts are highlighted in Red text on the font name selection control. Additionally, in the sample control, one can observe what is often salient to developers – Visual presentation of specific characters (Slashed Zero (0) vs. letter o/O, Upper-/Lower-case il vs IL vs 1, 2 vs Z, 5 vs S, etc.). The author prefers Consolas due to its commonality and these visual elements. The font selection is for all of the UI elements that are generated by EBT. Menus and control text is all stock BT font.

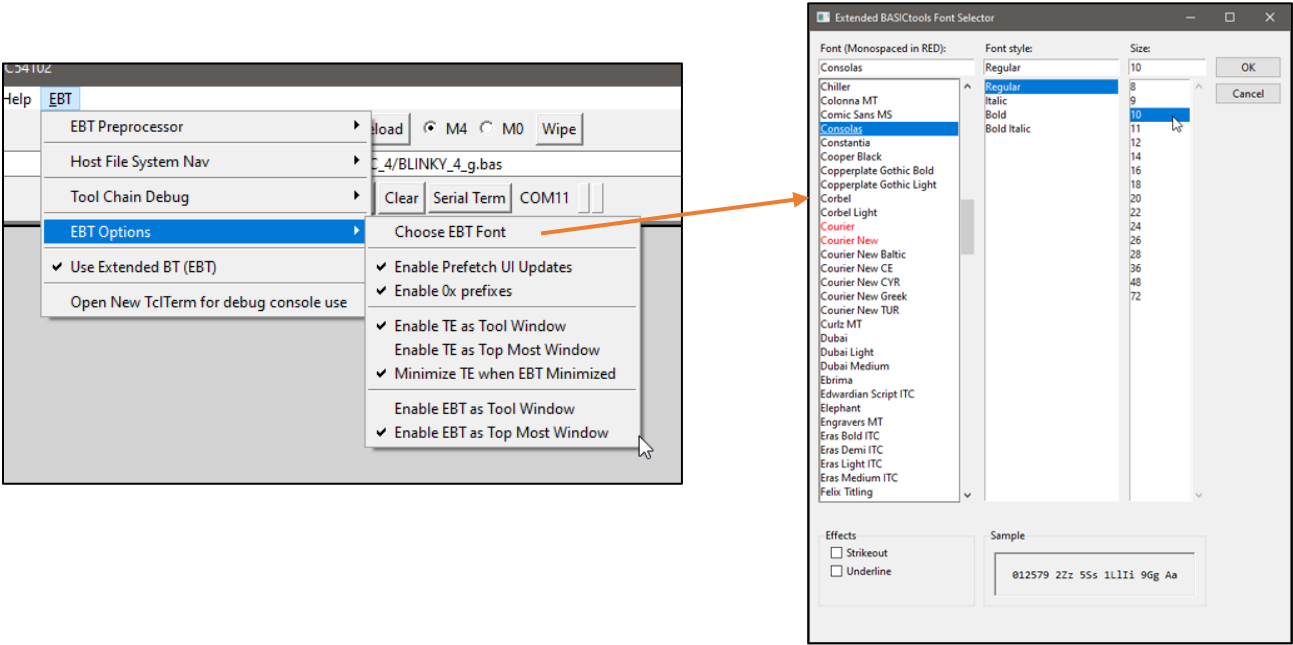


Figure 10: EBT Options - Window Modes, Prefetch, and Font Selection

### EBT Menu | Use Extended BT (EBT)

When EBT is active, this option is enabled. It is provided on the EBT menu to enable the dev user to deselect and restore the Stock BT functionality. Deselecting this menu option will cause a restart of the toolchain, in which the stock BT UI will be restored upon restart.

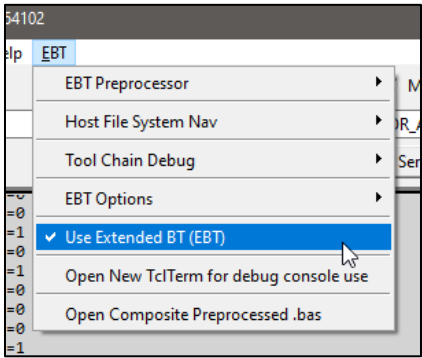


Figure 11: Use Extended BASICtools – Deselect to restore Stock BT-proper functionality

## EBT Menu | Open New TclTerm Instance

This menu option opens up a separate instance of BT's TclTerm Serial Terminal Program. A dev may find this useful to have the target stream debug out of a separate serial port (possibly at uber high rates – author has had 1.2mbps debug streaming from a Coridium SBC Target). Same could also be used to enable serial IO comms with a 2<sup>nd</sup> core on a multi-core target (4330/54102/...) via a separate serial port (peripheral or bitbanged via GPIO).

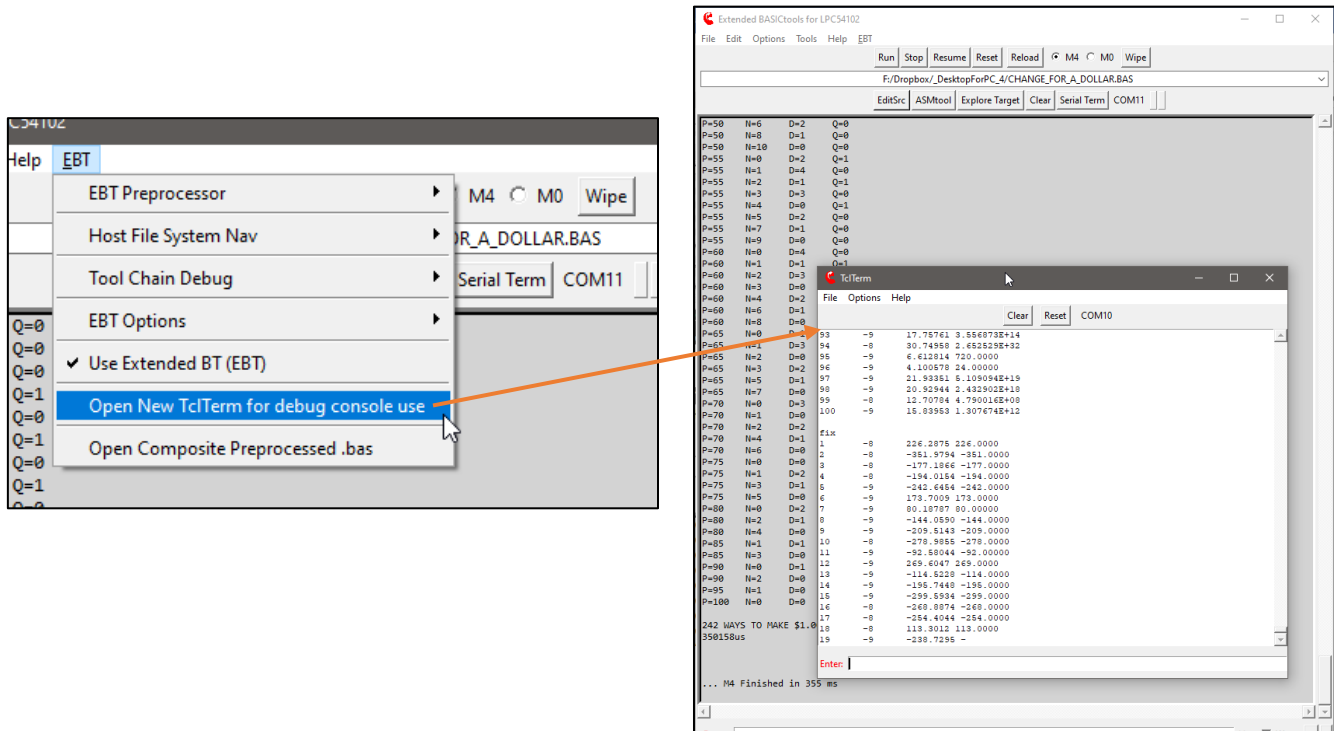


Figure 12: Open New TclTerm Instance (useful for Debug Console (or for 2<sup>nd</sup>-core comms) via other serial port)

## EBT Menu | Open Composite Preprocessed.bas

This menu option, when available (disabled prior to compilation), opens up an instance of the configured editor to allow the dev to review the composite .bas (after all BPP/FilePP preprocessing) to validate preprocessor impacts, etc.

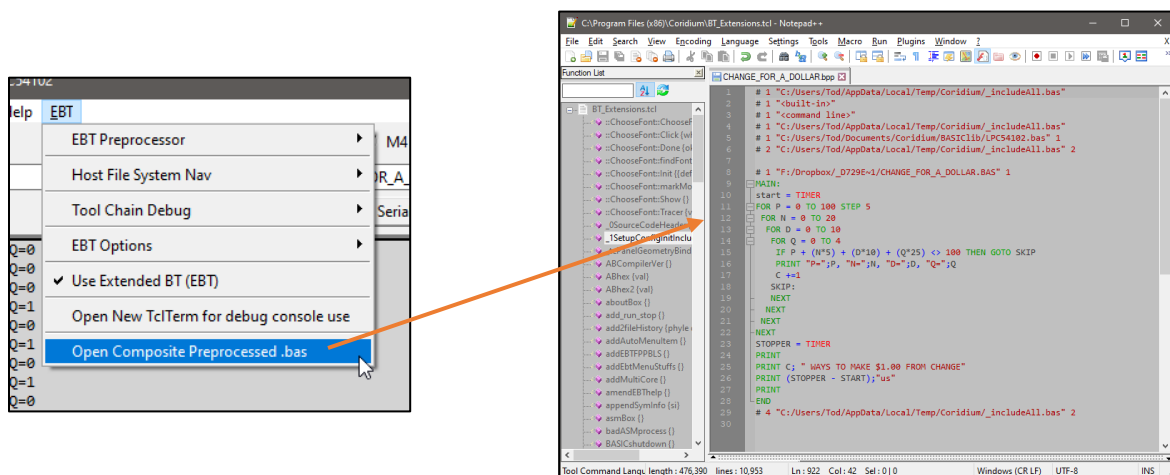
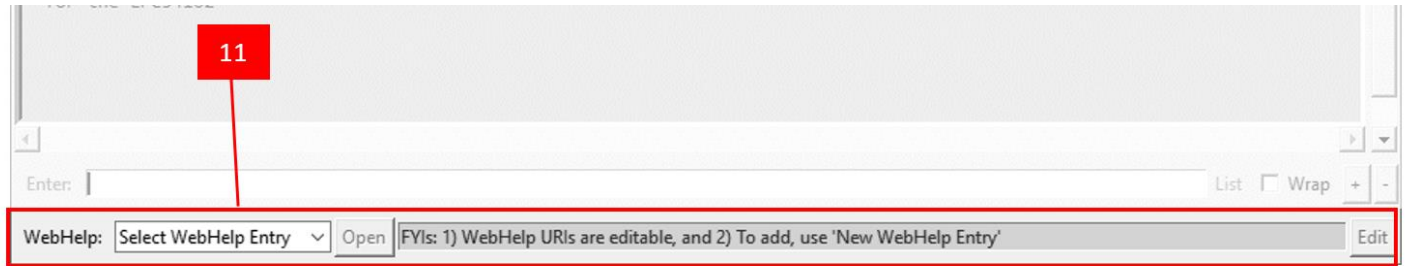


Figure 13: Open Composite Preprocessed.bas – available after sessions first compilation/flash load

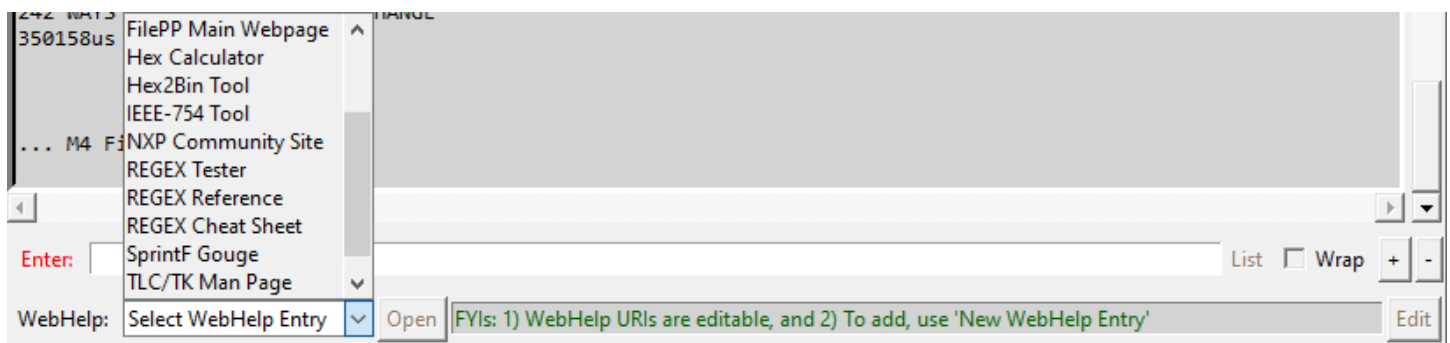
# EBT WebHelp MiniTool

EBT's main UI contains what has been termed as the WebHelp MiniTool at the very bottom of the window.



**Figure 13: Open Composite Preprocessed.bas – available after sessions first compilation/flash load**

This tool is intended to be a place in which links relevant to EBT, AB, MCUs, or whatever else a dev might find useful to have at their fingertips when working within EBT. As one can see in the following image, the author has a number of web resources that can be quickly opened by merely selecting same and then clicking the Open button alongside the DDL.

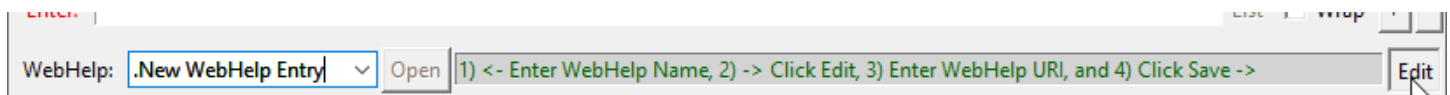


**Figure 14: WebHelp MiniTool Drop Down List depicting the current entries available for selection**

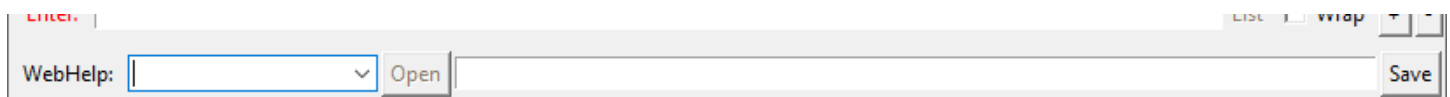


**Figure 15: REGEX Tester selected and Open being clicked – will open a browser to regex101.com, as depicted**

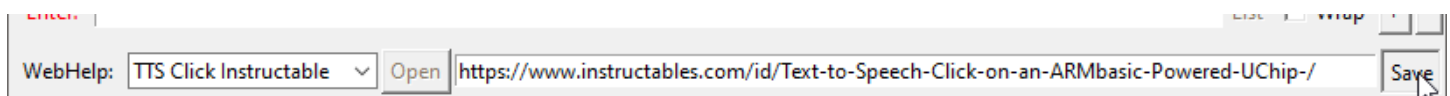
The author has programmed EBT's WebHelp MiniTool to allow new WebHelp entries to be added or edited in an easy and intuitive manner. The steps needing to be followed are detailed in the following images:



**Figure 16: Step 1: Select New WebHelp Entry**



**Figure 17: Step 2: Clear the entry name and click Edit (Edit button changes to a Save but)**



**Figure 18: Step 3: Enter WebHelp Entry Name, Enter URL for the resource, and then click Save button**

That is all there is to it. Anytime it is desired to visit that resource, just select it and then click the Open button alongside.

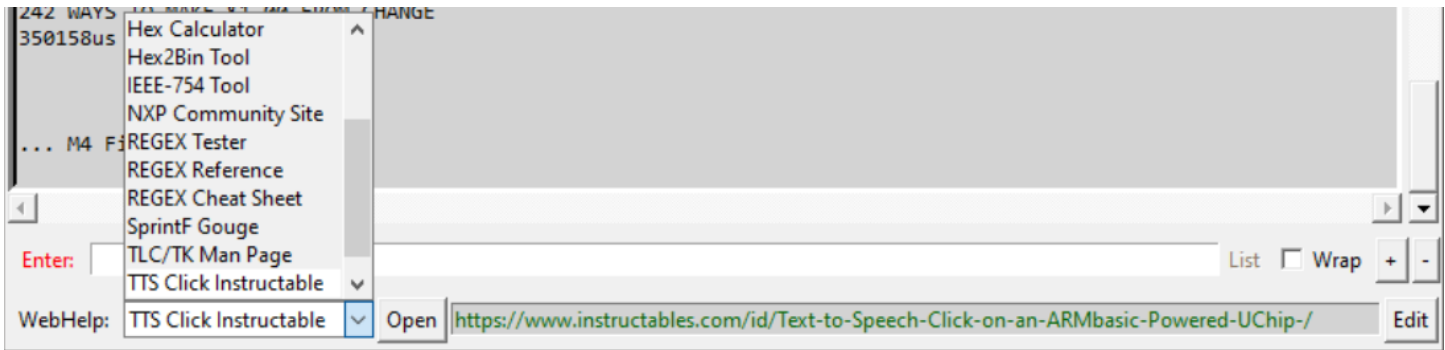


Figure 19: Selecting one of the entries in the WebHelp MiniTool

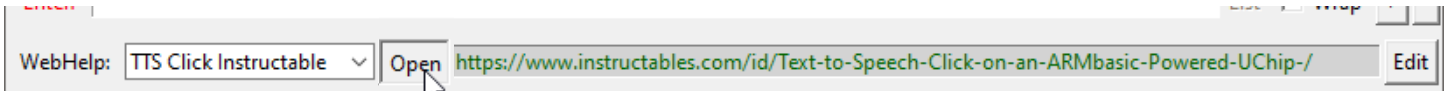


Figure 20: Clicking Open to launch a browser to open the selected resource

All EBT WebHelp MiniTool entries are backed up to the .ini file.

Also, the ASMtool, to be detailed in an EBT UM addendum (when completed), also has a WebHelp MiniTool. While the UI and controls are the same, the WebHelp entries therein are a separate set (also backed up in the EBT .ini) given the context of using the ASMtool to help compile ARM Assembly is likely [much] different than when developing ARMbasic.

## Conclusion

Thank you for taking the time to read about using the EBT toolset. If you have already tried EBT, or decide to do so, the author will be pleased. It is understood that it might not be to everyone's liking and that is OK. Trying to please everyone ensures a quick and painful trip to certain failure, which is not the intent here. If it does provide a few folks some usefulness, then it has achieved a goal.

If you experience trouble with EBT there are three options available to you to try to secure assistance:

- 1) Posting to thread on the Coridium Forums where EBT was announced,
- 2) Cracking open the tcl sources and see if you might glean a better understanding of what is transpiring and how it might be able to be resolved, or
- 3) Joining the Telegram ARMbasic Channel (<https://t.me/ARMbasic>) and seeing if anyone there might be someone who can offer input or assistance.

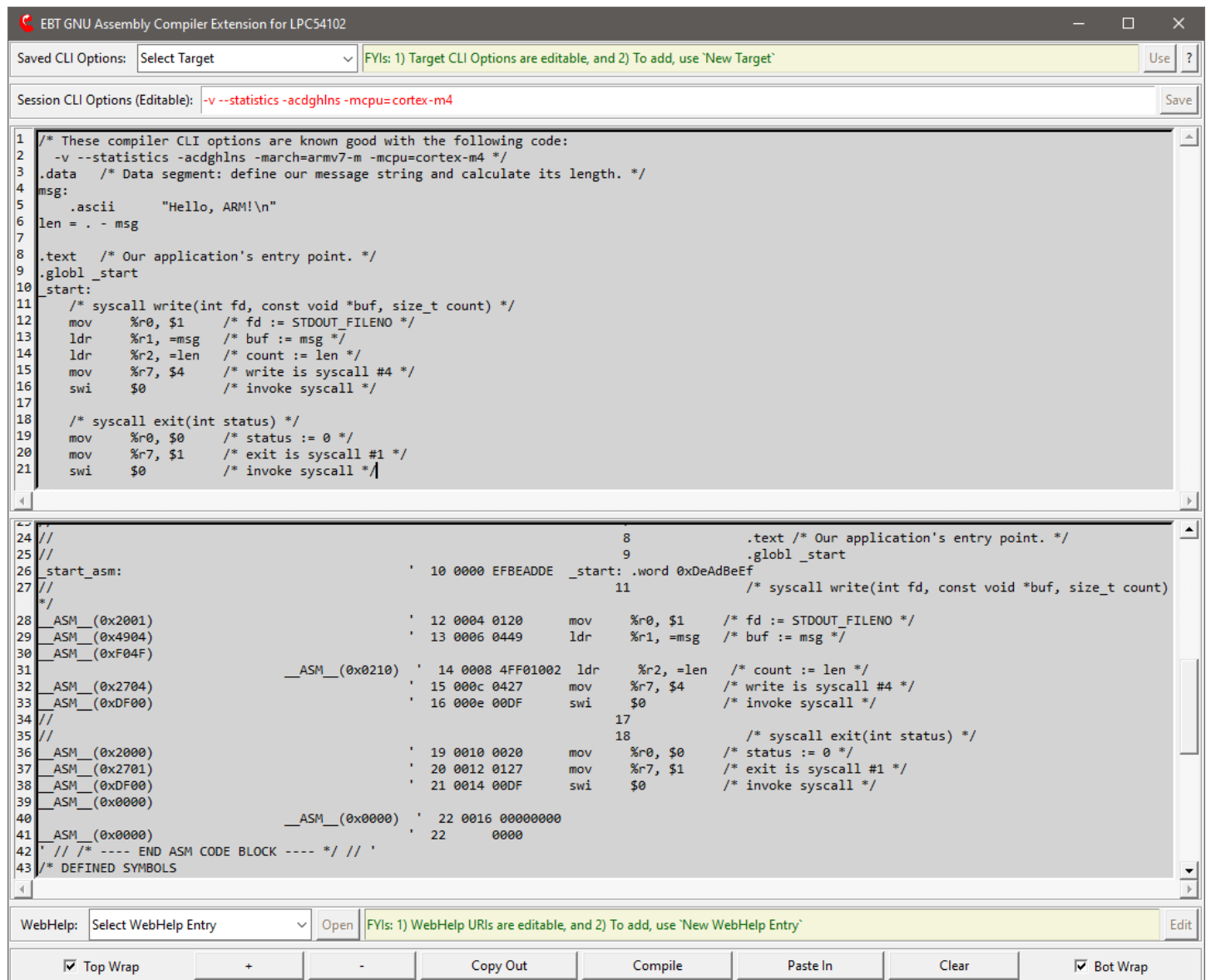
The author bids you well and hopes each and every one of you experience success. Take care and Happy Coding!

-t

## Wrapped .tcl scripts ... a note on borked Drag-N-Drop features of EBT

.tcl scripts can be ran directly, if a TCL/TK Wish Shell Interpreter is installed. Search for TCL Windows Binaries and you should come across something that will work. Author uses [Iron TCL](#). Coridium enables BT to be used sans an installed interpreter via [FreeWrap tcl script wrapper](#), providing an interpreted runtime wrapped around a .tcl script. BASICtools.exe is a 'wrapped' .tcl script. It has been noted that the more recent versions of FreeWrap creates a context that breaks the Drag-N-Drop EBT feature but is otherwise non-intrusive to EBT's ops. **Three options exist:** **1)** use EBT w/ BASICtools.exe accepting no DND support, **2)** Install a tcl interpreter, associate .tcl scripts w/ the tcl interpreter, & run BASICtools.tcl directly, or **3a)** Create a BASICtools.exe w/ an older version of FreeWrap or **3b)** request same via forums/telegram channel.

# Addendum 1: EBT ASMtool Use ... PLACE HOLDER for ASMtool Manual



The screenshot displays the EBT GNU Assembly Compiler Extension for LPC54102 interface. It features a main text area with assembly code, a disassembly view below it, and control panels at the top and bottom.

**Top Panel:**

- Saved CLI Options:** Select Target (dropdown), FYIs: 1) Target CLI Options are editable, and 2) To add, use 'New Target' (text), Use (button), ? (button).
- Session CLI Options (Editable):** -v --statistics -acdghlms -mcpu=cortex-m4 (text), Save (button).

**Main Text Area (Assembly Code):**

```
1 /* These compiler CLI options are known good with the following code:
2 -v --statistics -acdghlms -march=armv7-m -mcpu=cortex-m4 */
3 .data /* Data segment: define our message string and calculate its length. */
4 msg:
5 .ascii "Hello, ARM!\n"
6 len = . - msg
7
8 .text /* Our application's entry point. */
9 .globl _start
10 _start:
11 /* syscall write(int fd, const void *buf, size_t count) */
12 mov %r0, $1 /* fd := STDOUT_FILENO */
13 ldr %r1, =msg /* buf := msg */
14 ldr %r2, =len /* count := len */
15 mov %r7, $4 /* write is syscall #4 */
16 swi $0 /* invoke syscall */
17
18 /* syscall exit(int status) */
19 mov %r0, $0 /* status := 0 */
20 mov %r7, $1 /* exit is syscall #1 */
21 swi $0 /* invoke syscall */
```

**Disassembly View:**

```
24 // 8 .text /* Our application's entry point. */
25 // 9 .globl _start
26 _start_asm: 10 0000 EFBEADDE _start: .word 0xDEADBEF
27 // 11 /* syscall write(int fd, const void *buf, size_t count)
28 /* 12 0004 0120 mov %r0, $1 /* fd := STDOUT_FILENO */
29 _ASM__(0x4904) 13 0006 0449 ldr %r1, =msg /* buf := msg */
30 _ASM__(0xF04F)
31 _ASM__(0x0210) 14 0008 4FF01002 ldr %r2, =len /* count := len */
32 _ASM__(0x2704) 15 000c 0427 mov %r7, $4 /* write is syscall #4 */
33 _ASM__(0xDF00) 16 000e 00DF swi $0 /* invoke syscall */
34 // 17
35 // 18 /* syscall exit(int status) */
36 _ASM__(0x2000) 19 0010 0020 mov %r0, $0 /* status := 0 */
37 _ASM__(0x2701) 20 0012 0127 mov %r7, $1 /* exit is syscall #1 */
38 _ASM__(0xDF00) 21 0014 00DF swi $0 /* invoke syscall */
39 _ASM__(0x0000)
40 _ASM__(0x0000) 22 0016 00000000
41 _ASM__(0x0000) 22 0000
42 /* // /* ---- END ASM CODE BLOCK ---- */ // */
43 /* DEFINED SYMBOLS
```

**Bottom Panel:**

- WebHelp:** Select WebHelp Entry (dropdown), Open (button), FYIs: 1) WebHelp URIs are editable, and 2) To add, use 'New WebHelp Entry' (text), Edit (button).
- Controls:** ☒ Top Wrap, + (button), - (button), Copy Out (button), Compile (button), Paste In (button), Clear (button), ☒ Bot Wrap.



# Addendum 2: EBT Target Explorer (TE) Use ... PLACE HOLDER for TE Manual

The screenshot displays two instances of the EBT Target Explorer (TE) application, each showing the internal state of a different microcontroller core.

**Left Instance: EBT Target Explorer for LPC54102: CORE M4**

- Symbol Name:** SKIP
- Type:** CODE
- Size:** 110 Bytes
- Hex Address:** OP Codes: Assembler Mnemonics:
- Assembly Code:** Shows instructions like `B.N 0x812E`, `PUSH {R}`, `BL 0x8080`, etc.
- Symbol Table:** Lists symbols like `AB User Application`, `AB UserApp Code Blocks`, `Code_Start`, `MAIN`, `END_OF_CODE`, `AB UserApp Constants (Always Global)`, `AB UserApp Global Variables`, `Hardware Chip Symbols for LPC54102`, `CHIP_ADC`, `CHIP_APBCTRL`, `CHIP_CRC_CAUTION`, `CHIP_CTREG`, `CHIP_DMA`, `CHIP_EXCEPT_VECT`, `CHIP_FTCN`.
- Scratchpad Pane:** Contains text about mouse binds and routines.

**Right Instance: EBT Target Explorer for LPC54102: CORE M0**

- Symbol Name:** CT32B2
- Type:** CHIP\_CTREG
- Size:** 64
- Hex Address [Offset]:** Hex Value: Signed Int: IEEE754 Float: ASCII
- Assembly Code:** Shows instructions like `B.N 0x7008`, `PUSH {R}`, `NOVS R7, #1`, etc.
- Symbol Table:** Lists symbols like `AB User Application`, `AB UserApp Code Blocks`, `Code_Start`, `ENDLESSLOOP`, `MAIN`, `END_OF_CODE`, `AB UserApp Constants (Always Global)`, `AB UserApp Global Variables`, `Strings/Byte Arrays`, `Hardware Chip Symbols for LPC54102`, `CHIP_ADC`, `CHIP_APBCTRL`, `CHIP_CRC_CAUTION`, `CHIP_CTREG`, `CT32B2`, `CT32B2_CTRC`.
- Scratchpad Pane:** Contains text about mouse binds and routines.

## Addendum 3: EBT FilePP Use ... PLACE HOLDER for FilePP Manual/Examples

```
163 // enable boundary-less macro expansion - needed for namespace manipulation in the following loop
164 #pragma filepp SetWordBoundaries 0
165 #for idx 1 <= _ABmt_TaskCount 1
166     #if idx eq 250
167         wait(500)
168     #endif
169     ABmt_TaskEntryAddress(idx) = addressof(ABT_idx_MainLoop)
170     print "Task ";idx," Indexed @ 0x";i2h(ABmt_TaskEntryAddress(idx))," T: ";timer
171 #endfor
172 #pragma filepp SetWordBoundaries 1
173 #define Lo stored_define
174 #undef stored_define
175
176 print "User Tasks Indexed: "; ABmt_TaskCount," T: ";timer
```

```
18 #ifndef ABmt_TaskInclusionBegin 'single load' construct
19 #define ABmt_TaskInclusionBegin
20
21 #if defined onEBT || defined FilePP 'danger will robinson, BT's BPP doesn't support non-boundary (intra-token) macro expansion
22 #error ABmt is written to be used with the FilePP Preprocessor (integrated via EBT), to facilitate more robust compile-time functionality than what BT's BPP can offer
23 #else 'with FilePP, intra-token macro expansion works, but path resolution behavior is different than BT's BPP
24 // #warning ***** ABmt_TaskInclusionBegin
25
26 #ifdef ABmt_SchedulerCompile
27 #warning *****
28 #warning STARTING TASK INCLUSIONS
29 #warning *****
30
31 // this starts the math construct that is used the the wrapper to do the math for each task that is being processed for inclusion
32 #define _ABmt_TaskCount_buildup add(0)
33
34 // This depends on use of FilePP with word boundaries being turned on
35 #pragma filepp SetWordBoundaries 0
36
37 // this is a regex construct that drives name unicity for each task
38 // And, as info, to turn off regex, each must be done individually
39 // this works in concert with the preprocessor math construct to ensure
40 // ABmt_TaskID will be expanded on each iteration of the taskwrapper
41
42 // this macro expands and replaces ABT_ to ABT_ABmt_TaskID_ - to do
43 #regex /(\s*ABT_)/$1_ABmt_TaskID_/
44
45 // this macro expands and replaces main to ABT_ABmt_TaskID_MainLoop
46 #regex /\s*(?!(main:))\s*/sub ABT_ABmt_TaskID_MainLoop/
47
48 // this and the next macro were processing function and sub for macro
49 // #regex /\s*(?!(function))\s*/b/function ABT_ABmt_TaskID_/
50 // #regex /\s*(?!(sub))\s*/b/sub ABT_ABmt_TaskID_/
51
52 // this macro expands and replaces end with endsb, for concluding
53 #regex /\s*(?!(end))\s*/b/endsb/
54
55 #else
56 #error This lib is for use with ABmt and is intended to be included
57
58 #endif
59
60 #endif
61
62 #endif
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

## Appendix 1: Document Control and Record of Revisions

Document Control	
Document Number	0235_1800_EBT.um.001
Document Title	Extended Basic Tools (EBT) User Manual
Authored By	Tod Wulff, Chief Hack @ The House of Wulff
Source Location	<a href="https://coridium.us/tod/EBT/EBT_User_Manual.docx">https://coridium.us/tod/EBT/EBT_User_Manual.docx</a>
Published Location	<a href="https://coridium.us/tod/EBT/CurrentRelease/EBT_User_Manual.pdf">https://coridium.us/tod/EBT/CurrentRelease/EBT_User_Manual.pdf</a>
Acknowledgements	Bruce Eisenhard, Founder/HMFIC @ Coridium Corp Gary Zwan, ARMbasic/Embedded Dev Compatriot

Record of Revisions			
Version #	Date issued	Revisor	Reason for Revision
0.1	21 Aug 2020	Tod Wulff	- Initial draft document
1.0	22 Aug 2020	Tod Wulff	- Initial Release Version
1.0a	24 Aug 2020	Tod Wulff	- Various minor fixes: typos, missing words, extra text box removed, layout fixes, etc. - Added placeholder for FilePP Manual & Examples - Added note re Wrapped .tcl Scripts & borked DND