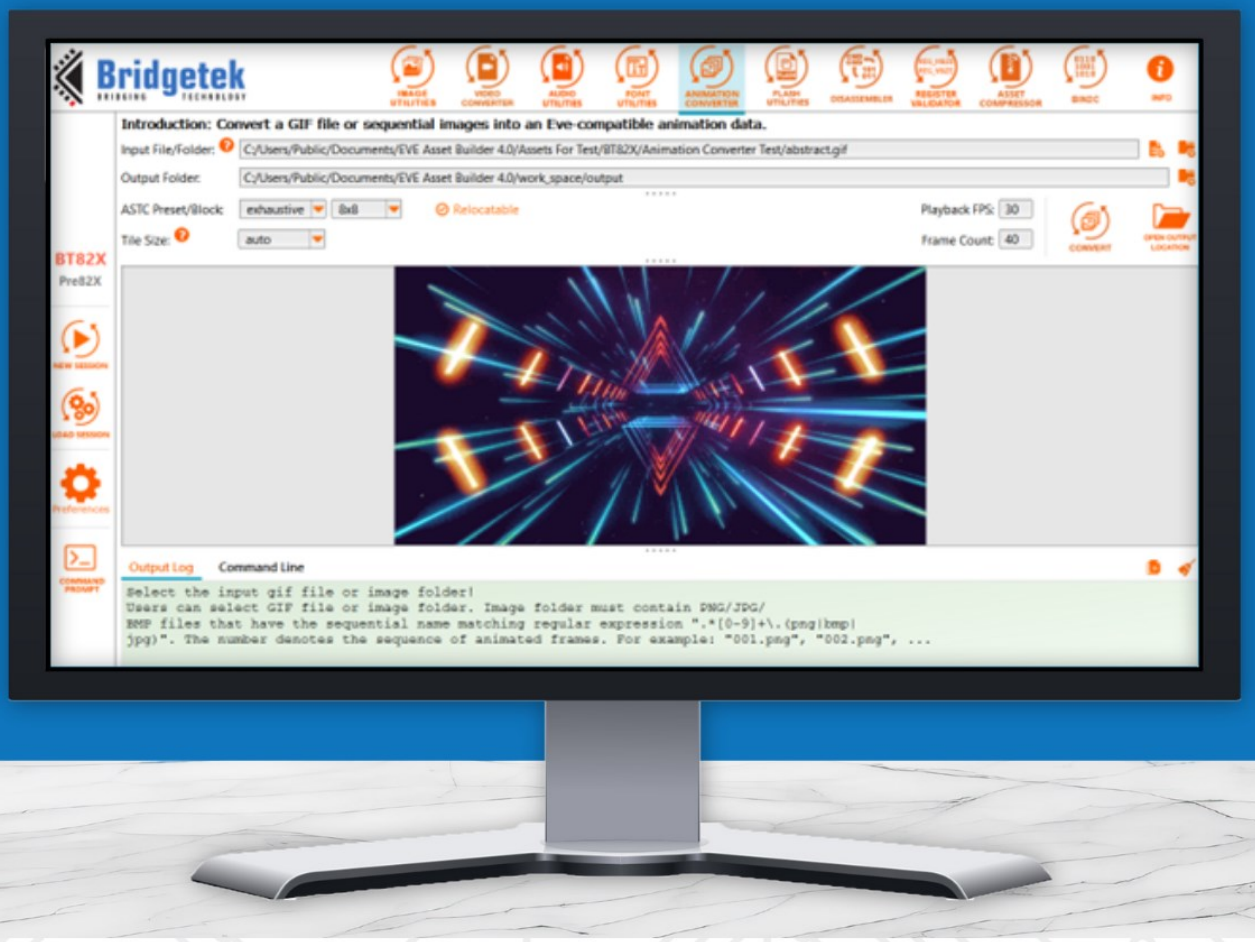
Bridgetek
BRIDGING TECHNOLOGY

EVE ASSET BUILDER 4.0

USER GUIDE



DOC. VER. 1,0
ISSUE DATE: 9 JULY 2025

Contents

I. Preface

A. Purpose

This user guide outlines the features, setup, and usage procedures for the **E**VE **A**sset **B**uilder (EAB).

B. Related Documents

- BT82X Series Programming Guide
- BT82X Datasheet
- BT81X Series Programming Guide
- FT81x_BT88x Series Programming Guide
- FT80x Series Programming Guide
- BT817/8 Datasheet
- BT817AQ For Automotive Datasheet
- BT815/6 Datasheet
- BT88X Datasheet
- FT81X Datasheet
- FT801 Datasheet
- FT800 Datasheet

C. Acronyms and Abbreviations

Terms	Description
ASCII	American Standard Code for Information Interchange
ASTC	Adaptive Scalable Texture Compression
BLOB	Binary Large Object
BMP	Bitmap Image File
EAB	EVE Asset Builder
EDF	EVE Flash Description File
ESD	EVE Screen Designer
ESE	EVE Screen Editor
EVE	Embedded Video Engine
HAL	Hardware Abstraction Layer
MPSSE	Multi-Protocol Synchronous Serial Engine
PNG	Portable Network Graphics
RDID	Read Device Identification
Pre82X	EVE Chip Generation Prior to BT82X
SPI	Serial Peripheral Interface
UI	User Interface
UTF	Unicode Transformation Format
WAV	Waveform Audio File Format

II. Overview

A. Introduction

The EVE Asset Builder (EAB) is a software application specifically designed for Windows operating systems. It provides a range of utilities to assist in generating resources for EVE chips. This document aims to provide a comprehensive guide on effectively using the tools within EAB.

B. Key Features

The following are the key features of EVE Asset Builder:

- Image Utilities: Convert bitmaps, validate PNG/JPEG, optimize JPEG, view RAW format.
- Video Converter: Convert a video file to MJPEG based AVI file.
- Audio Utilities: Convert an audio to EVE-compatible format, validate WAV file.
- Font Utilities: Convert a font to Legacy/Extended1/Extended2 format.
- Animation Converter: Convert a GIF file or a sequence of images to EVE-compatible format.
- Flash Utilities: Generate flash images; detect, program, update, write, erase, and read flash chips connected to EVE devices.
- Asset Compressor: Compress/Validate an asset to EVE-compatible format.
- Bin2C: Convert a binary file into a text file with C array defined.
- Disassembler: Translate EVE instructions into human-readable text.
- Register Validator: Validate BT82X display configuration registers.
- Custom Touch Firmware Compiler: Compile a C-like source file into loadable firmware.

C. What's New in EAB 4.0?

The EAB 4.0 unifies support for all EVE series chips, including legacy Pre82X (FT80X, FT81X/BT88X, BT81X) and the BT82X series. It integrates functionalities previously found in EAB 2.x and 3.x into a single application. Future versions will deprecate EAB 2.x and 3.x.

Feature Updates:

General

- Add a toggle button to switch between BT82X and Pre82X modes.
- Add a button to explain why BT82X mode of EAB only supports high quality bitmap formats.
- Add a button to explain the difference between Zlib and Zopfli.
- Users can change or restore the ASTC encoder through the Preferences dialog.
- Logging service can now be enabled or disabled.
- Sample Application generation can now be toggled on or off.
- The eab_tools.exe command-line tool now supports both BT82X and Pre82X.
- Command Prompt sets the current working folder to the session's output directory.
- Add build and debug support for BT82X example applications in VSCode.

Audio Utilities

- Check the validity of WAV audio files.
- Support audio channel selection.

Custom Touch Compiler

- Add a link in each sample tab to open the binary output folder for quick access.

Flash Utilities

- Add 4096-byte padding at the end of flash image for compatibility.
- Improve pseudo-code for better readability and logic clarity.

D. Known Issues & Limitations

None

E. Credits

Open-Source Software

- Qt: *https://www.qt.io/*
- Python: *https://www.python.org/*
- ASTC Encoder: *https://github.com/ARM-software/astc-encoder*
- Nuitka: https://nuitka.net/
- FFMPEG: *https://www.ffmpeg.org/*
- OptiPNG: *http://optipng.sourceforge.net/*
- JPEGTran: *http://jpegclub.org/jpegtran/*
- QtProgressCircle: *https://github.com/mofr/QtProgressCircle*
- FriBidi: *https://github.com/fribidi/fribidi*

Icons Copyright

Icons made by *Freepik*, *Smashicons*, *Pixel perfect*, *Dave Gandy*, *Royyan Wijaya*, *Icons8*

Audio Copyright

Audio for testing by *Lesfm* from *Pixabay*, and *mixkit*

Video Copyright

Video for testing by *vecteezy*

III. Setup and Installation

A. System Requirements

To install the application, ensure that your PC meets the requirements recommended below:

- ✓ Recommended Windows 10 or above
- ✓ 64-bit platform
- ✓ 1.6GHz or faster processor
- ✓ 1GB of RAM (1.5GB if running on a virtual machine)
- ✓ A multi-core CPU is highly recommended
- ✓ At least 512MB of hard disk space
- ✓ Display resolution 1300 x 900 pixels or higher
- ✓ "Write" permission for the installation folder

B. Installing EVE Asset Builder

The following steps will guide you through the EVE Asset Builder *Setup/Installation* process.

i. Download the package from *https://brtchip.com/toolchains/#EVEAssetBuilder*.

ii. Extract the zip file contents. Double click on the file **EVE-Asset-Builder-setup.exe**.

iii. The EVE Asset Builder Setup Wizard is displayed along with a Welcome message.

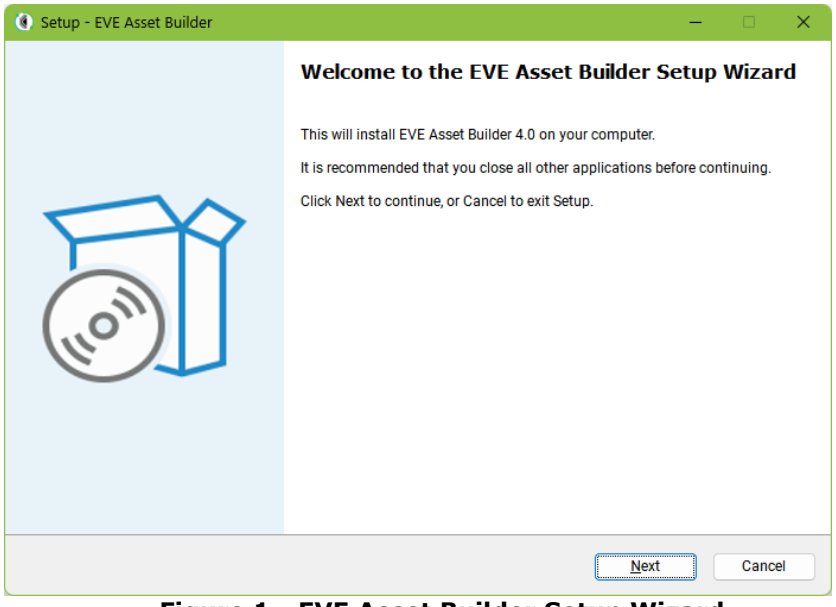


Figure 1 - EVE Asset Builder Setup Wizard

iv. Click **Next** to select a "Destination Folder" for installing the files. Accept the default folder or click **Browse** to specify a different location. Click **Next** to confirm the destination folder and continue.

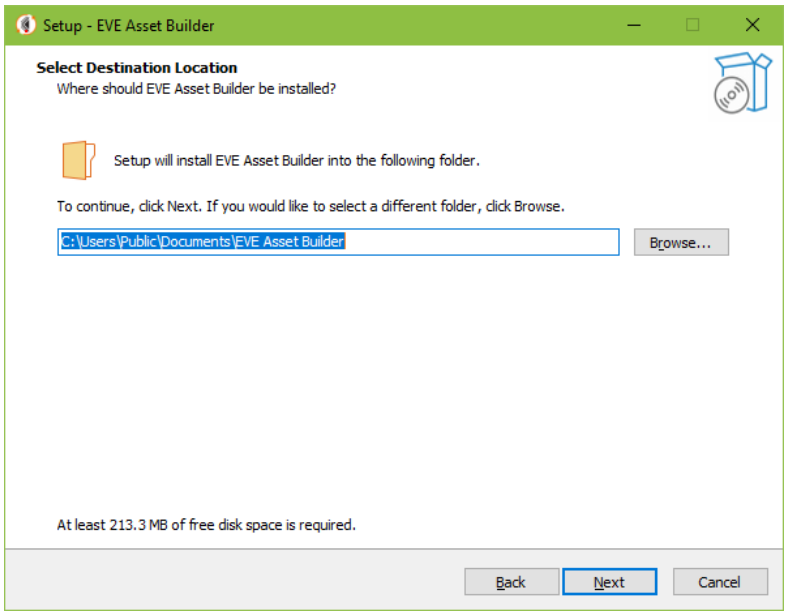


Figure 2 - EVE Asset Builder Setup - Select the Destination Folder

v. In the **Select Additional Tasks** window, check the **"Create a desktop shortcut"** box, to have the EVE Asset Builder icon displayed on the desktop if required. Click **Next** to prepare for the installation.

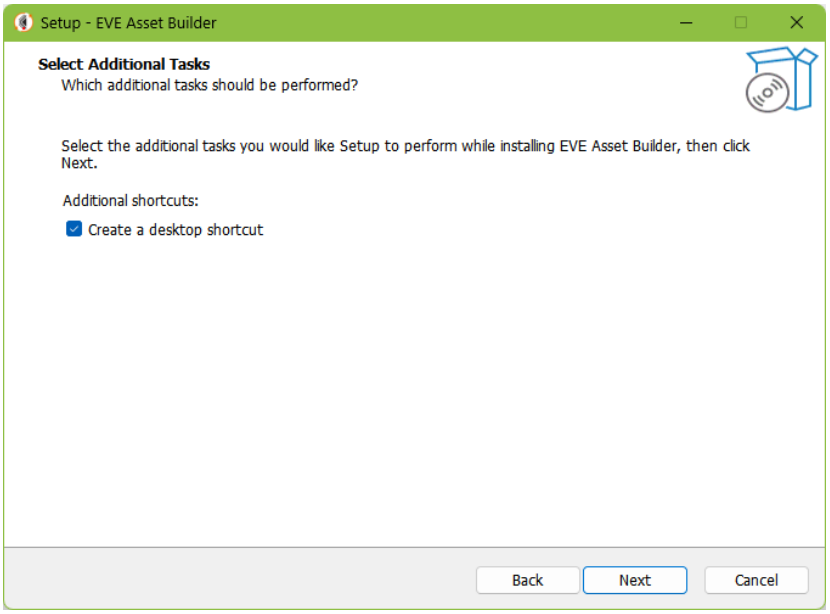


Figure 3 - EVE Asset Builder Setup - Ready for Installation

vi. Click **Install** to start the installation.

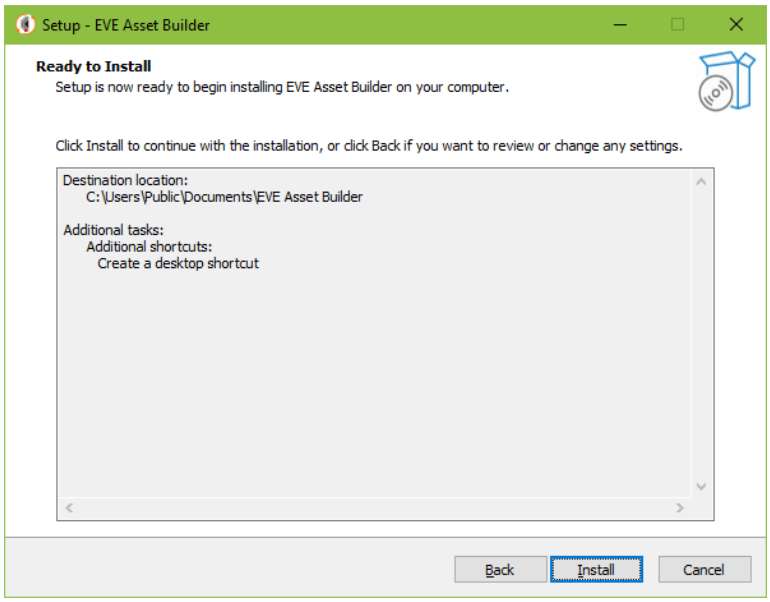


Figure 4 - EVE Asset Builder Setup - Start Installation

vii. A progress bar indicates that the installation is in progress.

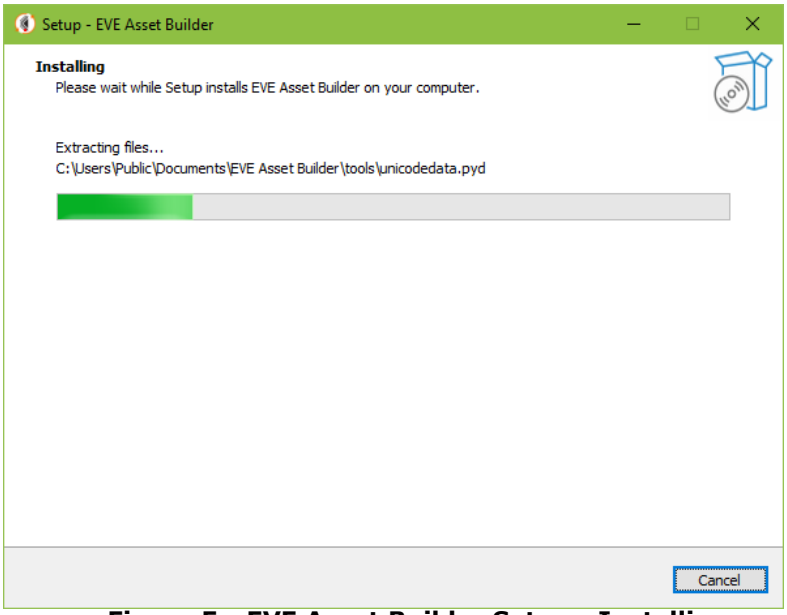


Figure 5 - EVE Asset Builder Setup - Installing

viii. Upon successful installation, click **Finish**. The EVE Asset Builder application UI is displayed.

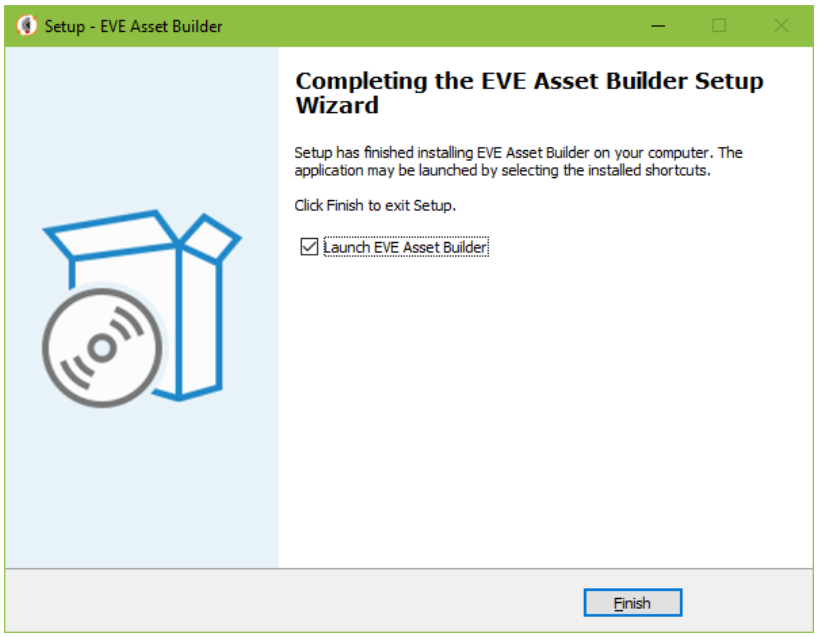


Figure 6 - EVE Asset Builder Setup - Finish

IV. Core Technologies

A. EVE Chip Series

Starting from version 4.0, EAB supports the following EVE chip generations:

- BT82X refers to: BT820
- BT81X refers to: BT815/6, BT817/8
- BT88X refers to: BT880/1/2/3
- FT81X refers to: FT810/1/2/3
- FT80X refers to: FT800/1

Users can use the toggle button in the left panel to switch between utilities for BT82X and Pre82X chip generations.

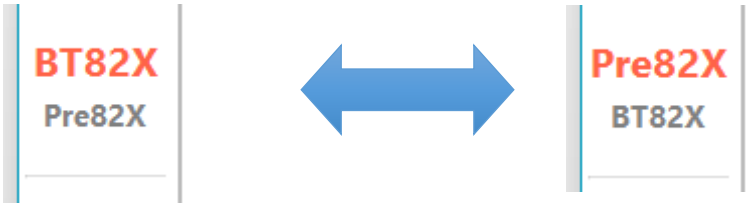


Figure 7 - Chip Generation Switch

B. ASTC Encoder

Within EAB, users can choose the ASTC preset and ASTC block as follows:

ASTC Preset: fastest, fast, medium, thorough, and exhaustive. Encoding speed is inversely proportional to output quality: the "fastest" preset offers the quickest performance with the lowest quality, while "exhaustive" ensures the highest quality at the cost of slower processing.

ASTC Block: 4x4, 5x4, 5x5, 6x5, 6x6, 8x5, 8x6, 8x8, 10x5, 10x6, 10x8, 10x10, 12x10, 12x12. Smaller blocks give better quality with lower compression; larger blocks give higher compression with lower quality.

Users can download various versions of ASTC encoders and choose the most suitable one for use with EAB.

The "*Change ASTC Encoder*" button is found in the Preferences dialog. When clicked, it opens a folder selection dialog where users can choose a directory containing ASTC encoder binaries with suffixes like *sse2*, *sse4.1*, and *avx2*. EAB then reads the CPU specifications and automatically picks the most suitable encoder. A separate button is available to restore the default encoder, which is version **5.2.0**.

Note:

- ASTC format assets are not compatible between BT82X and Pre82X because of differences in bitmap layout. Therefore, ASTC-based bitmaps, fonts, and animation assets cannot be shared or used interchangeably across the two platforms.

- Official link to ASTC encoder releases: https://github.com/ARM-software/astc-encoder/releases

C. Relocatable Asset (BT82X Onward)

The BT82X generation introduces a structure that allows fixed-address assets to become relocatable. With the ***cmd_loadasset*** command, assets can be loaded at any 64-byte aligned offset within the RAM_G address space, removing the need for users to manage asset addresses manually. This feature currently supports relocation of Animation and Font assets.

Note: Relocatable assets must be loaded at an offset aligned to a 64-byte boundary.

D. Audio Format and Sample Rate

Sample Rate: Users can retain the input sample rate or selecting any value from the provided list: 48000, 44100, 32000, 22500, 16000, and 8000 Hz.

Audio Format:

- S16S_SAMPLES (stereo)
- S16_SAMPLES (mono)
- 8-bit signed PCM
- 8-bit u-Law
- 4-bit IMA ADPCM

Note: S16S_SAMPLES and S16_SAMPLES are introduced in BT82X and are recommended for this generation due to their high audio quality. Lower formats should be avoided on BT82X, as they provide inferior sound quality.

V. Getting Started

A. Session

This functionality allows for the recording of user activities for future reference. On the left panel, two buttons help create or load sessions.

Figure 8 - Session Buttons

NEW SESSION: Start recording user actions, all data is stored in a session file.

LOAD SESSION: Read a session file and restore all previous inputs.

Note: When EAB is first started, it creates a default session. On subsequent starts, it will load the most recent session.

1. New Session

Input and output folders will be set as specified in the session dialog below for all utilities of EAB. The output folder must be empty. A session should be named to find easily at the time of loading. Once a session has started, all inputs from users will be recorded and then saved to a file called *<session name>.eab*

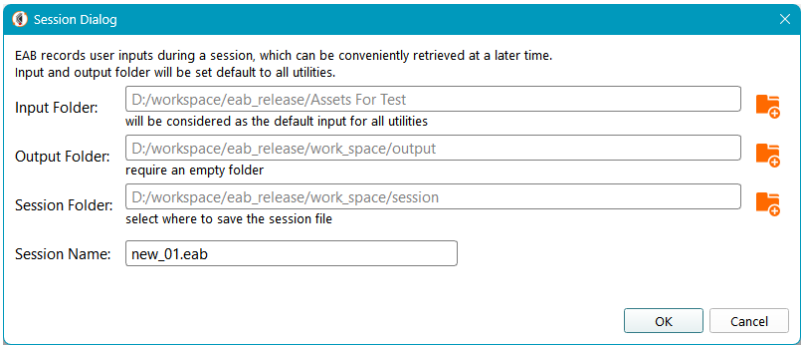


Figure 9 - Session Dialog

2. Load Session

Load a previously saved session, restoring all user inputs.

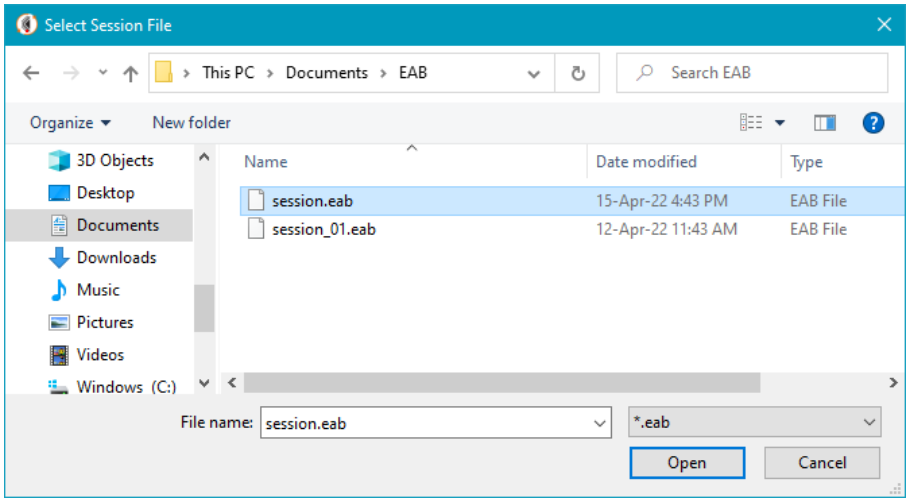


Figure 10 - Select Session File

B. Preferences Dialog

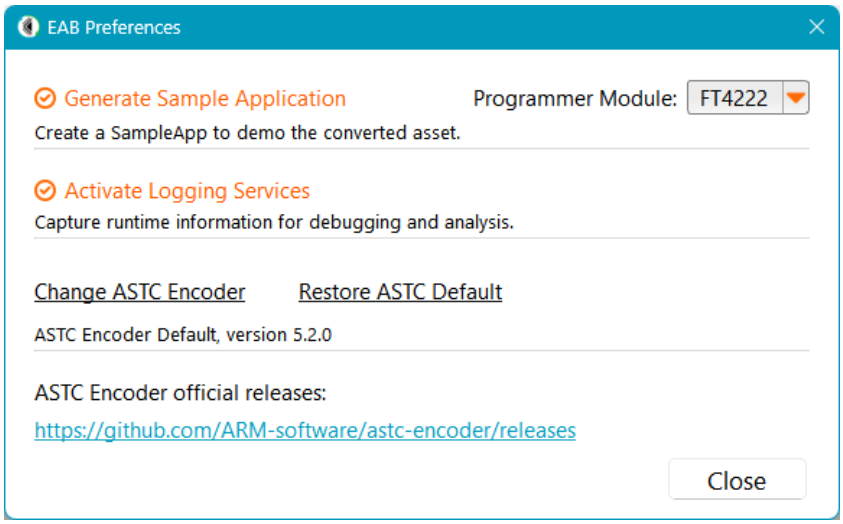


Figure 11 - Preferences Dialog

When users press the Preferences button on the left panel, a dialog opens with settings for the entire application.

Generate Sample Application: When checked, EAB will create a sample project after conversion to showcase the usage of the converted assets. The sample project supports FT4222 and MPSSE platform.

Activate Logging Services: When enabled, EAB writes the output or errors of each conversion to a log file.

Change ASTC Encoder, Restore ASTC Default: see ASTC Encoder

C. Output Folder

Generally, all utilities include an Output Folder section with a read-only text box and a button to open a folder selection dialog. This folder is used to save all output files, such as converted assets, metadata JSON files, and more.

D. Command Prompt

Located at the left edge, the *COMMAND PROMPT* button will open a command window. The system temporarily adds the paths of *eab_tools.exe* and *ffmpeg.exe* to the PATH environment variable so that users can execute these tools without concern for where they are located.

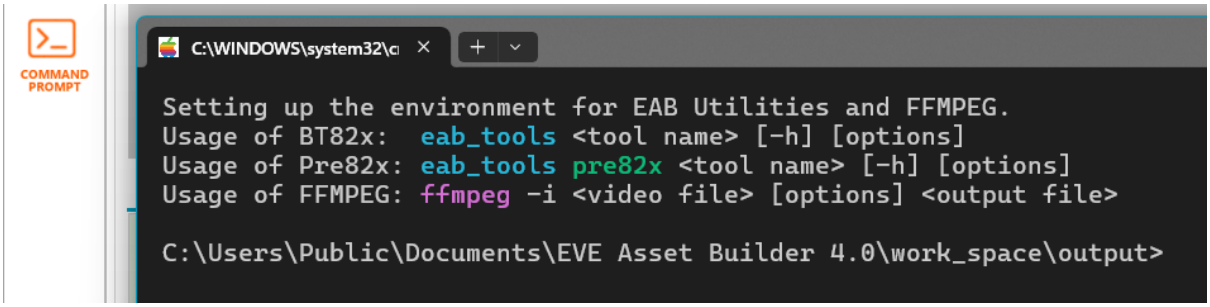


Figure 12 - Command Prompt Window

Upon execution, every tool in the EVE Asset Builder generates a command-line text in the Log Window. This text can be copied and pasted by users into the command window, allowing them to execute it from there.

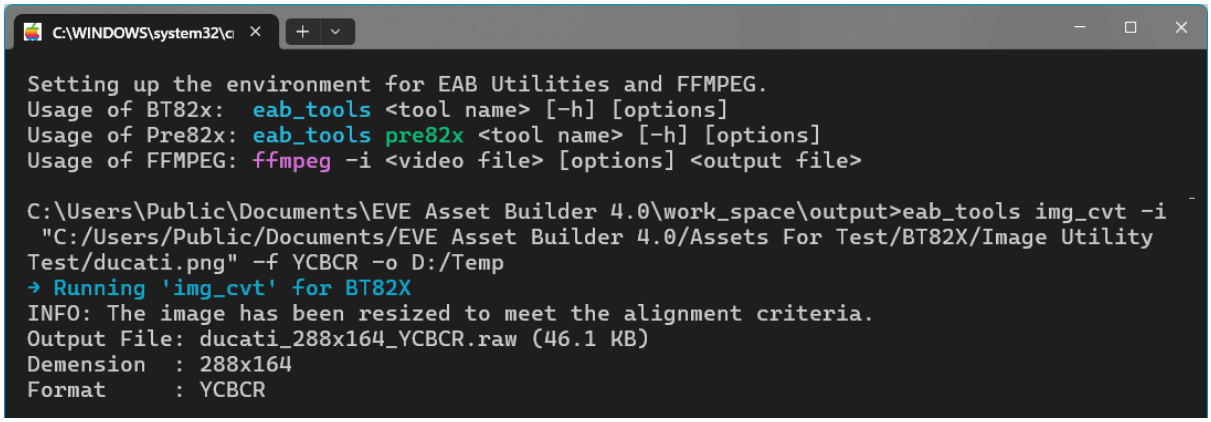


Figure 13 - Command Prompt - Example

E. Log Window

The *Log Window* displays information related to inputs, outputs, warnings, or errors to assist users in effectively utilizing EAB tools.

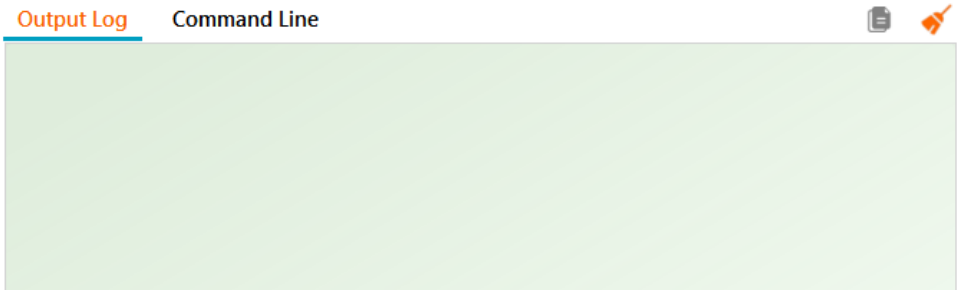


Figure 14 - Log Window

CLEAR: Erase all content in Log Window.

COPY: If there is any text selected, copy it to the clipboard. Otherwise, copy all the content.

Output Log Tab: Show the log output.

Command Line Tab: Show the command-line texts which have been executed. User can paste them into Command Prompt to run.

F. Naming Convention and Usage of Output Files

The output files generated by each tool in the EVE Asset Builder adhere to a specific naming convention, as outlined in the following table. The naming conventions and their intended usage are detailed in the table below. It should be noted that <input> refers to the base name of an input file, such as "lena" for an input file named "lena.png", while <output> is defined by the user.

Tool	Output Naming Convention	Example of input and output file	Usage
Image Converter	*<input>*_*<width>*x*<height>*_ *<output format>.raw*	lena.png -> lena_185x150_L8.raw	- Load the .raw file to RAM_G - Decode it using cmd_setbitmap
Video Converter	*<input>*.avi	big_buck.avi -> big_buck.avi	- Load the output file to RAM_G - Decode it using cmd_playvideo
Audio Converter	**Pre82X** *<input>*.raw	happy_summer.mp3 -> happy_summer.raw	- Load the .raw file to RAM_G - Play it by controlling the appropriate registers
	BT82X *<input>*.<raw|wav>	happy_summer.mp3 -> happy_summer.wav	- Load the .wav file to RAM_G - Use cmd_loadwav or cmd_play wav to playback the audio
Font Converter	**Pre82X** *<input>*_<font size>_ <bitmap format>.<raw|xfont|glyph>	arial.ttf -> arial_39_ASTC.raw, arial_39_ASTC.xfont, arial_39_ASTC.glyph	**Legacy:** - Write the raw file to a predefined offset in RAM_G **Extended:** - Write the xfont file to RAM_G - Write the glyph file to FLASH or RAM_G **Render:** - Decode it using cmd_setfont
	BT82X *<input>*_<font size>_ <bitmap format>.<raw|reloc> Notes: Assets use **.reloc** as the suffix when relocatable, and **.raw** otherwise.	arial.ttf -> arial_39_ASTC.raw, arial_39_ASTC.reloc	**Relocatable:** - Use cmd_loadasset to load the .reloc file into RAM_G **Raw:** - Write the raw file to a predefined offset in RAM_G **Render:** - Decode it using cmd_setfont
Animation Converter	**Pre82X** *<input>*.anim.<flash|ram_g>	abstract.gif -> abstract.anim.flash, abstract.anim.ram_g	- Write the .anim.flash to FLASH, or .anim.ram_g to RAM_G - Use cmd_animstart, or cmd_animframe to play the animation
	BT82X *<input>*.anim.<raw|reloc>	abstract.gif -> abstract.anim.raw, abstract.anim.reloc	**Relocatable:** - Use cmd_loadasset to load the .reloc file into RAM_G **Raw:** - Write the .anim.raw file to a predefined offset in RAM_G **Render:** - Use cmd_animstart, or cmd_animframe to play the animation
Flash Image Generator	*<output>*.bin *<output>*.map *<output>*.edf	abstract.anim.reloc, lena_ASTC_5x5.raw -> flash_16MB.bin,	- Write the .bin file to flash chip - See the offset and length of each asset in the map or the edf file - Use cmd_flashread to copy an asset to RAM_G, then use the appropriate commands to render it - Or, use cmd_flashsource to specify the asset's offset and

			render it with commands that support OPT_FLASH
		flash_16MB.map	Each line contains the asset information in the format: *name:offset:length*
		flash_16MB.edf	Refer to section EDF Block
Asset Compressor	**Zlib** <*output*>.zlib	arial.xfont, big_buck.avi -> inflate.zlib	- Load compressed file to flash or RAM_G - Use cmd_inflate to decompress the assets
	Zopfli <*output*>.zopfli	arial.xfont, big_buck.avi -> inflate.zopfli	
Bin2C Converter	<*input*>.c	img_ycbcr.bin -> img_ycbcr.c	Include the C file to use the converted C-array
Disassembler	<*input*>.da.txt	bt82x_dump.bin -> bt82x_dump.da.txt	Open .da.txt file to see the human -readable EVE commands
All Utilities	<*input*>.json	abstract.gif -> abstract.anim.json	A .json file contains the meta information of converted asset. It can be opened by any text editor. ESE or ESD will use it to load the assets into their projects too.

Table 1 - Naming convention and usage of output files

VI. Utilities For BT82X

A. Image Utilities

This utility provides various image-related features. The first tab focuses on converting PNG, JPEG, and BMP images to a format compatible with EVE. The second tab allows you to visualize the resulting raw output.

1. Image Converter

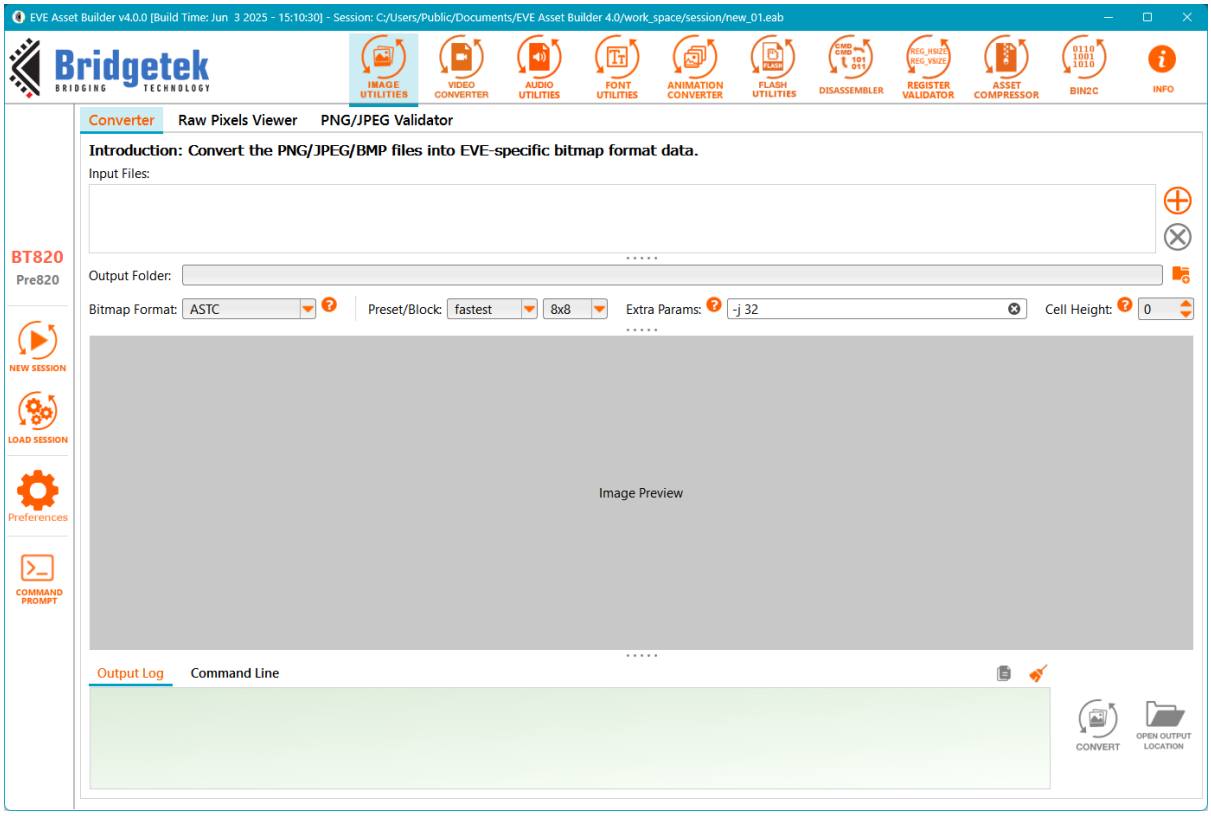


Figure 15 - Image Converter

Input Files: PNG/JPG/BMP images to be converted. Users can change the order of images by using *Ctrl+Up* or *Ctrl+Down*.

Bitmap Format: The output format of images, is one of the following:

- ARGB6, ARGB8, RGB6, RGB8
- LA1, LA2, LA4, LA8
- PALETTEDARGB8
- ASTC
- YCBCR

ASTC Preset/Block: see ASTC Encoder

ASTC Extra Params: Users can explore various options to determine the optimal balance for their specific needs, considering factors like memory availability, performance demands, and visual quality.

Cell Height: Include padding data to ensure that each bitmap cell aligns to 64 bytes in memory, enabling proper rendering in the ASTC format on the EVE chip. Specifying a value

greater than 0 will enable this feature. EAB will adjust the Cell Height value to align with the ASTC block size.

Dithering: Dithering Enabling dithering can help reduce noise levels, but it may negatively impact compression efficiency. The bitmap formats ASTC, YCBCR, and PALETTEDARGB8 do not support dithering.

Alpha (%): Ranges from 0 to 100. A value of 0 means fully transparent, while 100 means fully opaque. Users can draw a rectangle in the Image Preview by holding Ctrl and using the left mouse button to define the area where the alpha effect applies. This option is available in bitmap formats LA1, LA2, LA4, and LA8.

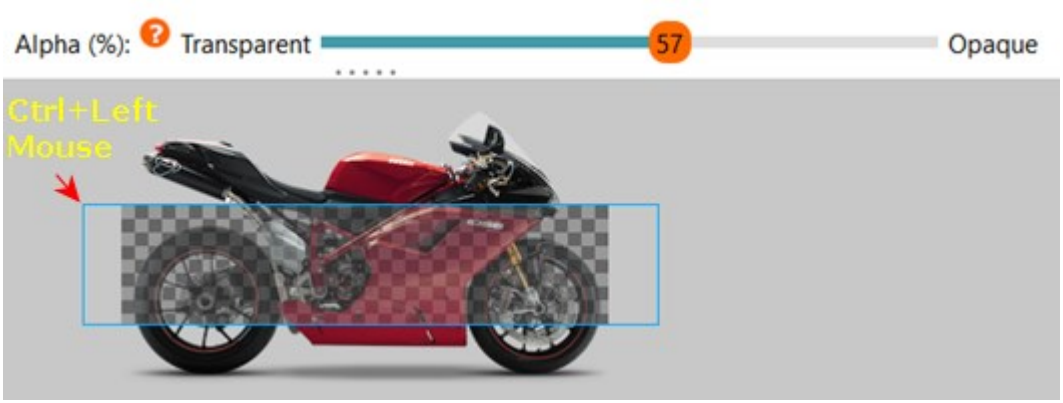


Figure 16 - Image Converter - Alpha

Note: Images are resized prior to conversion to ensure alignment with the requirements specified in the table below.

Bitmap Format	Bits Per Pixel	Image Width
L1	1	8-pixel aligned
L2	2	4-pixel aligned
L4	4	2-pixel aligned
L8	8	-
LA1	2	4-pixel aligned
LA2	4	2-pixel aligned
LA4	8	-
LA8	16	-
RGB332	8	-
RGB565	16	-
RGB6	18	8-pixel aligned
RGB8	24	2-pixel aligned
ARGB2	8	-
ARGB4	16	-
ARGB1555	16	-
ARGB6	24	2-pixel aligned
ARGB8	32	-
PALETTEDARGB8	8	-
YCBCR	8	2-pixel aligned

Table 2 - Image Width Alignment

2. Raw Pixels Viewer

This tool attempts to visualize the raw bitmap data using the provided bitmap format and image size.

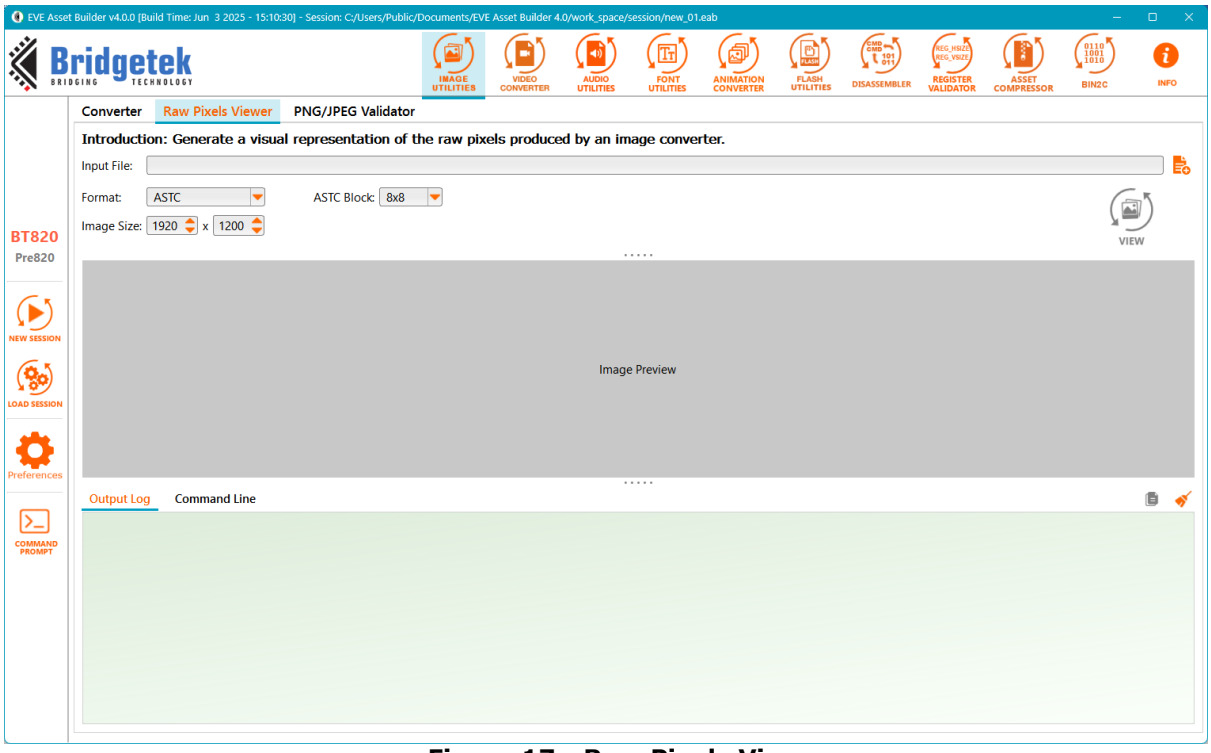


Figure 17 - Raw Pixels Viewer

Input File: Pick the .raw file to view. For PALETTEDARGB8, select the .index.raw.

Format: See *Bitmap Format*.

Image Size: Set the image width and height.

VIEW: Render the raw bitmap data. The resulting image appears in the Image Preview panel.

Notes: If EAB finds the corresponding json file, it will retrieve the bitmap format and image size, and update the UI accordingly.

3. PNG/JPEG Validator

This tool helps validate PNG or JPEG images to determine if they can be successfully loaded by ***cmd_loadimage***. It also provides an option to optimize incompatible images using third-party tools, *optipng* and *jpegtran*, to generate EVE-compatible images.

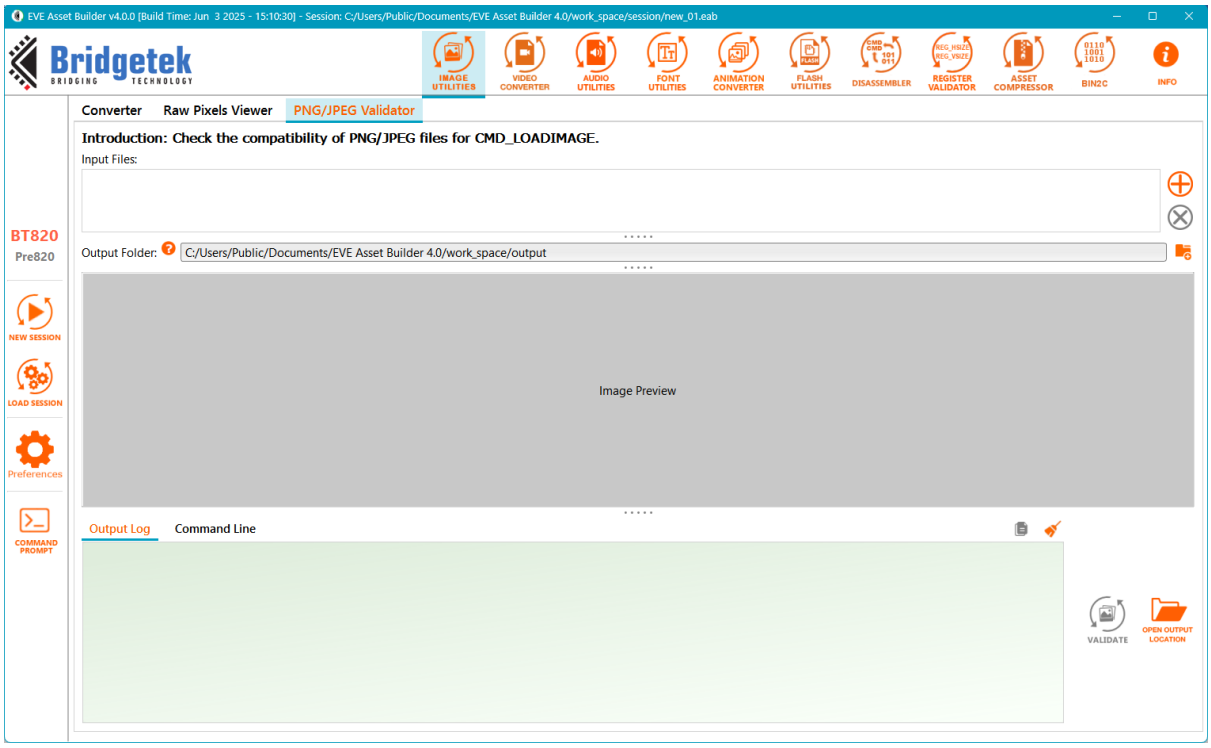


Figure 18 - PNG/JPEG Validator

Input Files: Select PNG or JPEG images for validation.

Image Preview: Display the selected image.

VALIDATE: Use the BT82X emulator is tto load images via the ***cmd_loadimage*** command. Consider the load successful if there is no coprocessor fault or if RAM_CMD is not stuck. This tool will prompt for optimizing incompatible images and then validate them again.

Notes:

- PNG: Adam-7 interlaced images are not supported. Only bit-depth 8 is allowed; bit-depths 1, 2, 4, and 16 are not supported.
- JPEG: Only baseline JPEGs with YCbCr formats 444, 422, or 420 are supported by ***cmd_loadimage***. Progressive JPEGs or other YCbCr formats are not compatible.
- This tool relies on the BT82X emulator to validate PNG/JPEG images. Users may see different results on the hardware.

B. Video Converter

This tool converts a video file into the MJPEG format and packages it within an AVI container. The resulting .avi file will be compatible with EVE devices. Users can also play it on personal computers to quickly verify if the converted file contains the desired video and audio. The audio stream format can be either mono or stereo.

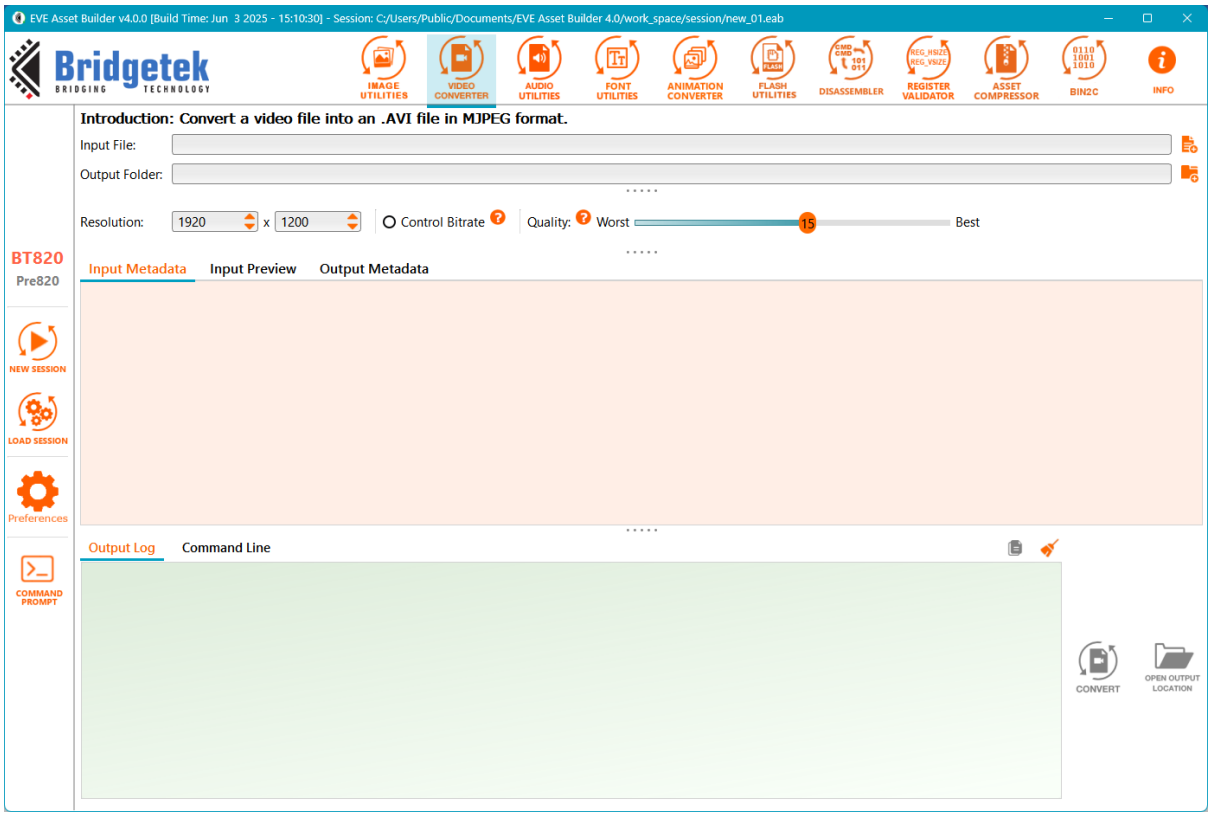


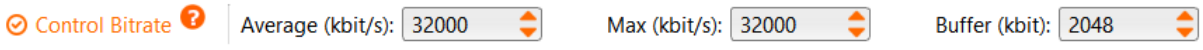
Figure 19 - Video Converter

Input File: Pick a video file to convert.

Resolution: Must be specified, the video will be converted to this resolution.

Quality: From 1 (best) to 31 (worst).

Control Bitrate: Allow users to control the bitrate. Enabling this mode will override the Quality setting, and vice versa.



- Average (kbit/s): Set the average bit rate for the output video
- Max (kbit/s): Set max bit rate for the output video
- Buffer (kbit): Set buffer size for the output video

Keep Audio Stream: Users can keep or remove the audio stream.

Audio Format and Sample Rate: see Audio Format and Sample Rate

C. Audio Utilities

1. Audio Converter

Convert a WAV or MP3 audio file into a format optimized for EVE devices, ensuring seamless processing and playback. The audio stream format can be either mono or stereo.

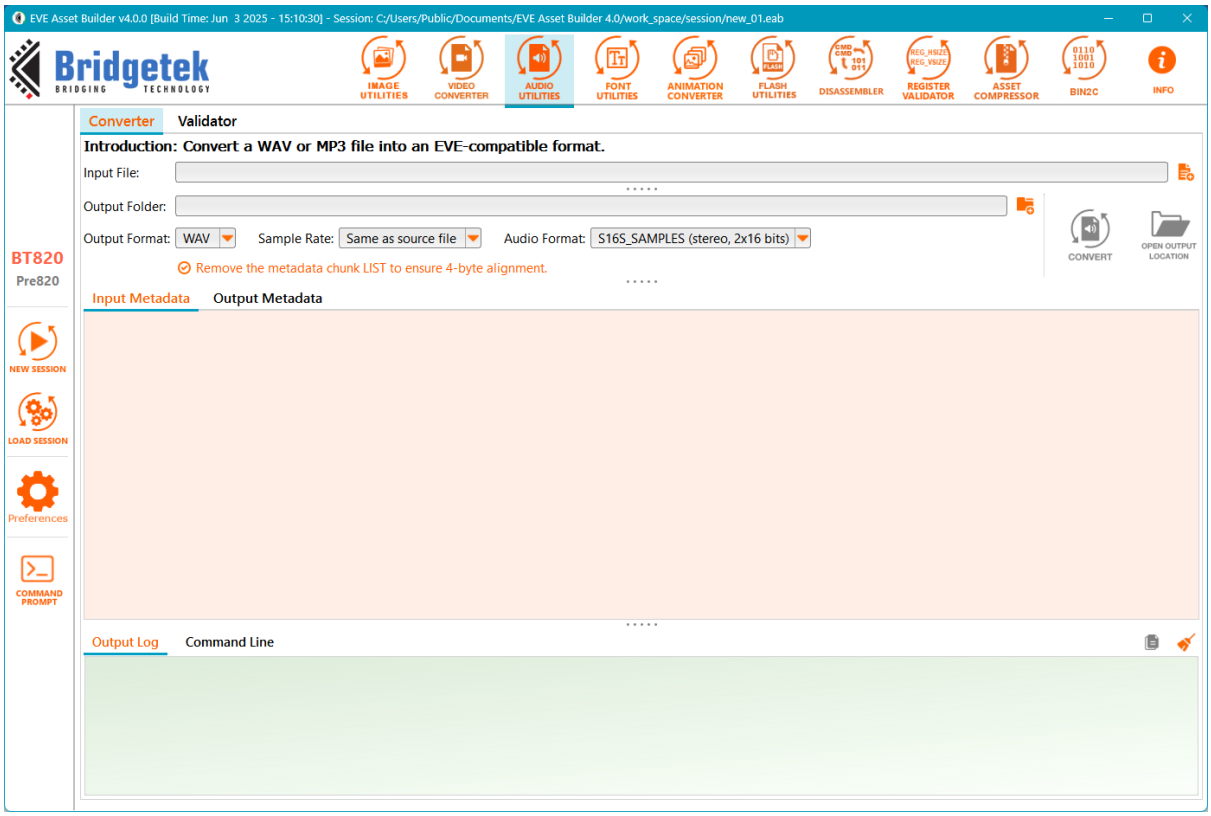


Figure 20 - Audio Converter

Input File: Pick an audio file to convert.

Audio Format, Sample Rate: see Audio Format and Sample Rate

Output Format: WAV or RAW.

Note: There is a checkbox to remove metadata from a WAV file, as it can cause the file length to be misaligned by 4 bytes and serves no purpose during playback.

2. WAV Validator

WAV Validator uses an emulator to load the WAV file via ***cmd_loadwav*** to determine whether the input file is compatible.

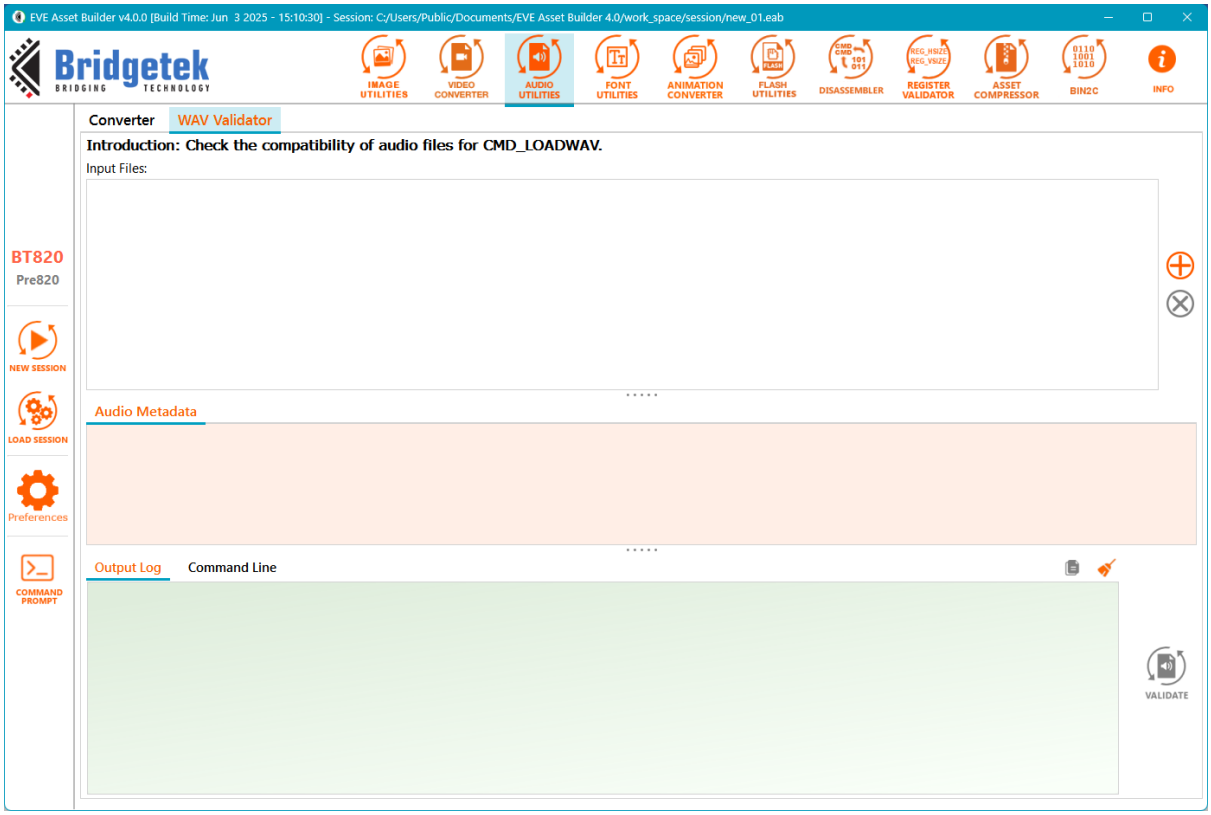


Figure 21 - WAV Validator

Input Files: Pick .wav files to verify.

Audio Metadata: Displays general information about the audio file using ffprobe.

EAB shows a PASS or FAIL result with the reason for each WAV file, along with a summary in the Output Log.

D. Font Utilities

1. Font Converter

Extract characters from the OpenType or TrueType font file into a specific font structure. Unprintable characters will be filtered out. Since each glyph is converted to a bitmap individually, combination characters are not supported.

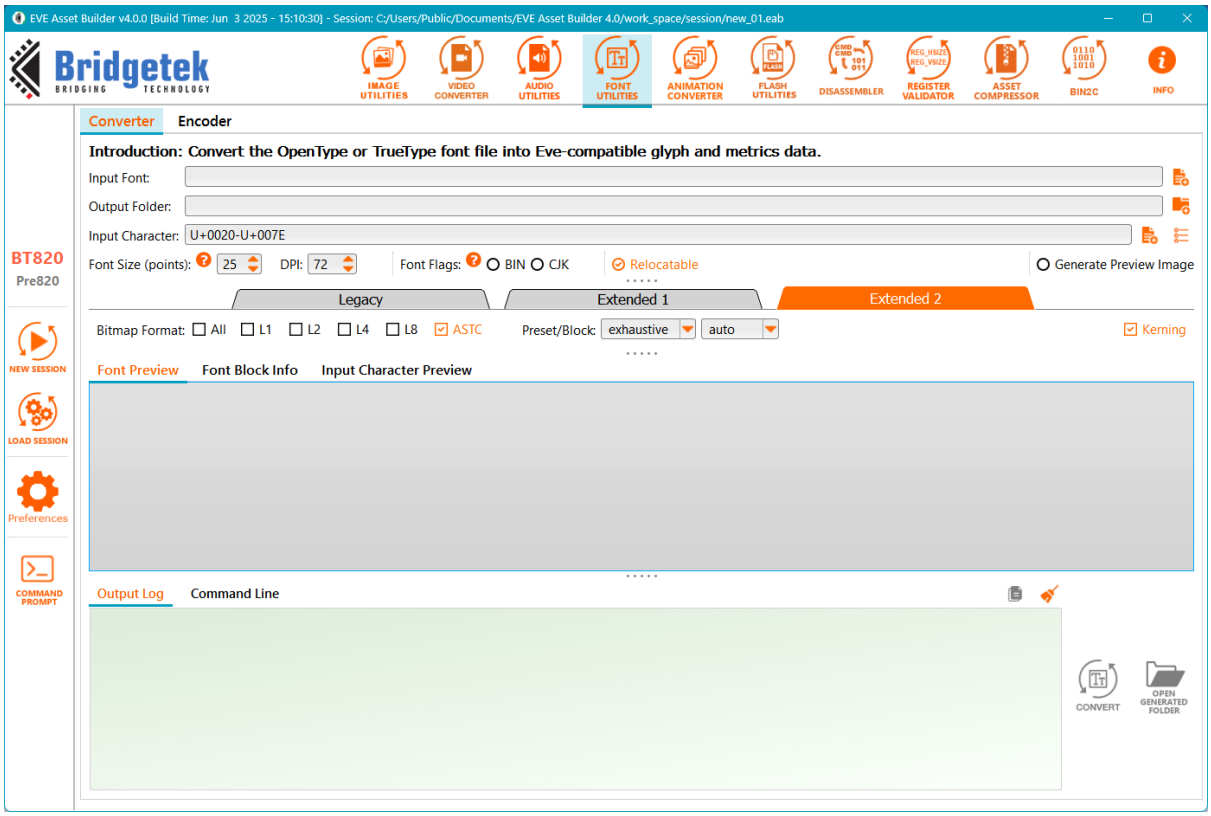


Figure 22 - Font Converter

Input Font: Pick a font file to convert.

Input Character: Select the character set to convert.

Font Size: Set the desired font size. Any character wider than 255 pixels in width will be removed.

Font Flags:

BIN: When enabled, disabling newline interpretation in the font string.
CJK: When enabled, OPT_FILL breaks lines at any character instead of at word breaks.

Relocatable: see **Relocatable Asset**

Address: Specifies the offset address in RAM_G to load the converted font. Must be a multiple of 4. Available only when Relocatable is unchecked.

Generate Preview Image: Produces images to help verify that characters are rendered as expected.

Font Preview Area: Display sample characters by using the selected font.

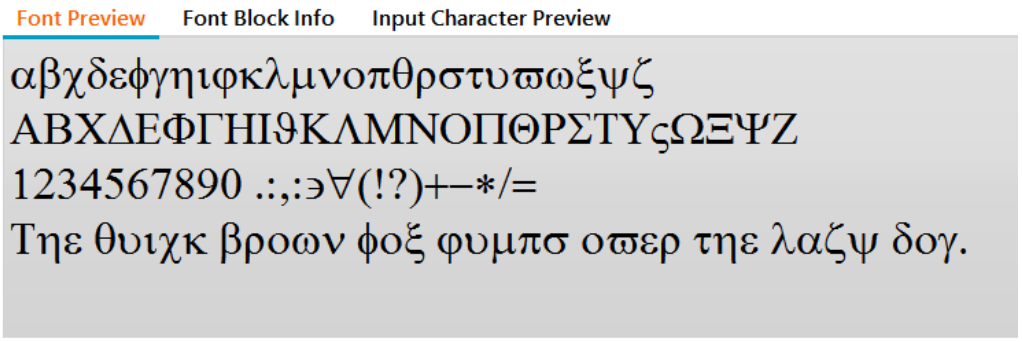


Figure 23 - Font Preview Area

Font Block Info:

- Show the number of glyphs that are valid for the selected font.
- Show a statistic of the selected font, including Block Name, Unicode Range, and Valid Glyphs/All Glyphs.

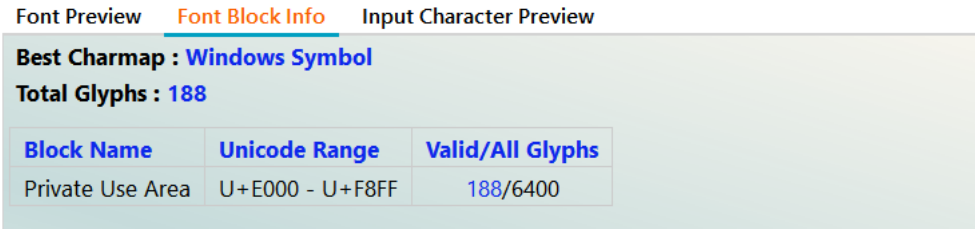


Figure 24 - Font Block Info

Input Character Preview:

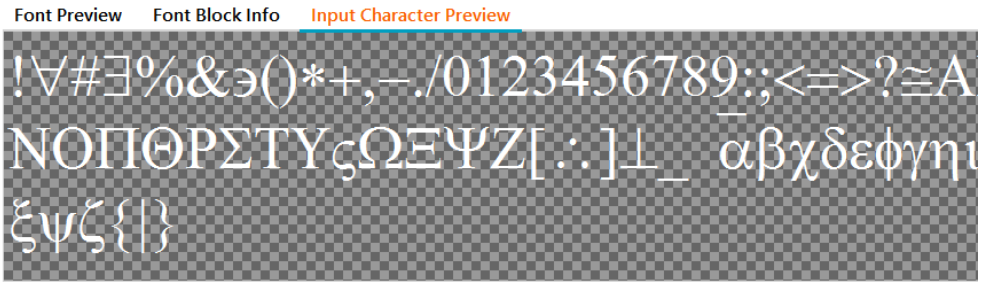


Figure 25 - Input Character Preview

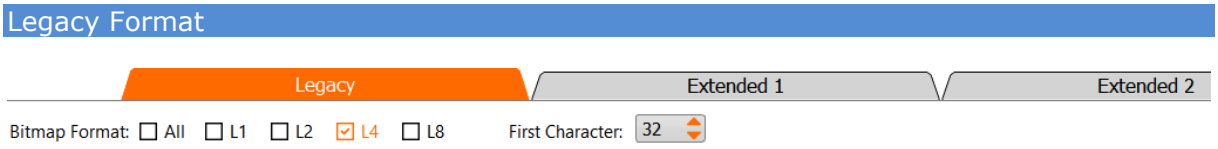


Figure 26 - Legacy Format

Bitmap Format: Supported bitmap formats are L1, L2, L4, and L8. Each selected format will be converted separately in a single operation.

First Characters: Set the first index for converted characters. The valid range is 1-127.

Output:

For each bitmap format, the tool generates:

- A .raw file includes the font metrics block and graphic data, or
- A .reloc file contains the converted font in a relocatable format.

Extended Format 1

Handle fonts with a full range of Unicode code points.

Figure 27 - Extended Format 1

Bitmap Format: Supported bitmap formats are ASTC, L1, L2, L4, and L8. Each selected format will be converted separately in a single operation.

ASTC Preset/Block: see ASTC Encoder

Note: For *auto* option, block footprint depends on font size:

- If font size > 52: block footprint = 10x8
- If font size < 19: block footprint = 8x5
- Otherwise: block footprint = 8x8

Output:

For each bitmap format, the tool generates:

- A .raw file includes the font metrics block and graphic data, or
- A .reloc file contains the converted font in a relocatable format.

Extended Format 2

This format supports all Unicode code points and includes kerning support. Although it requires the font file to contain a kerning table, users can enable or disable kerning freely. The font structure differs entirely from Extend Format 1, but its usage remains the same.

Figure 28 - Extended Format 2

Kerning: Enable kerning support for fonts with a GPOS table.

Refer to *Extended Format 1* for other settings.

2. Text Encoder

The Text Encoder serves as a companion utility to the font converter. It encodes user input text using either UTF-8 or ordinal code points, as determined by the .json file generated by the font converter. It aids programmers in seamlessly working with EVE coprocessor commands.

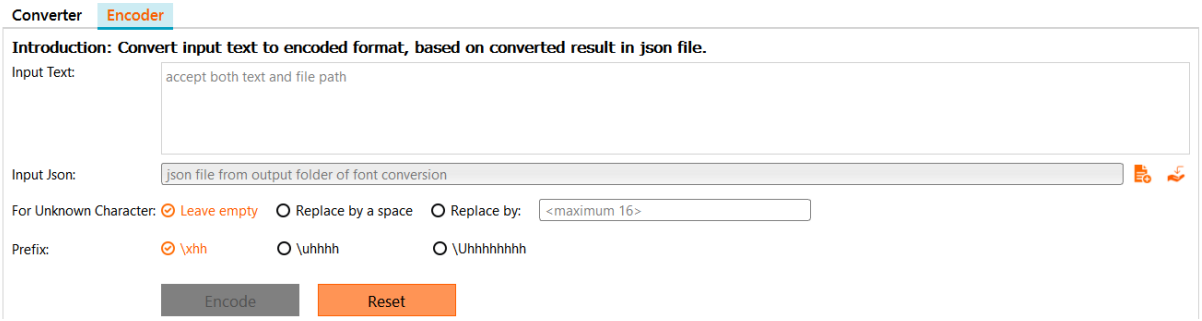


Figure 29 - Text Encoder

Input Text: The character set to encode.

Input Json: The file which determines what code points the input text maps to.

For Unknown Character: Options on how to process with unknown characters.

Prefix: The prefix of the code point.

Encoded Text: The resulting text is displayed here. Users can copy it and use it in another application.

One example is as below:

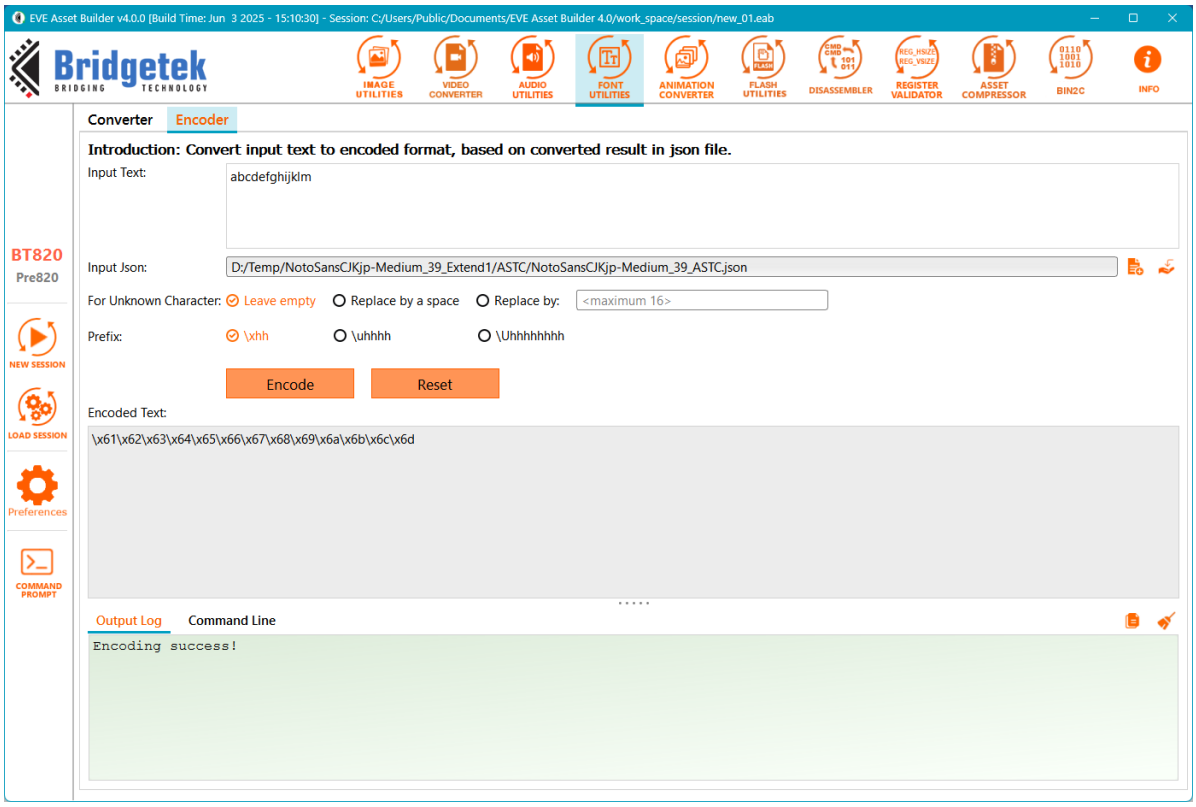


Figure 30 - Text Encoder Example

3. Built-in Font

Font Handles 16 to 25:

Non-antialiased, *Helvetica* typeface, from Apple x11, Adobe/DEC license.

Handle	Point Size	Bitmap Format	Bitmap Size (WxH in pixel)
16	NA	L1	8x8
17		L1	8x8
18		L1	8x16
19		L1	8x16
20		L1	10x13
21		L1	13x17
22		L1	14x20
23		L1	17x22
24		L1	24x29
25		L1	30x38

Table 3 - Font Handles 16 to 25

Font Handles 26 to 34:

Anti-aliased, *Roboto* typeface, from Google, Apache license.

Handle	Point Size	Bitmap format	Bitmap Size (W×H in pixel)
26	14	L4	14×16
27	17	L4	16×20
28	21	L4	18×25
29	24	L4	22×28
30	31	L4	28×36
31	42	ASTC 8×8	36×49
32	54	ASTC 8×8	46×63
33	71	ASTC 8×8	60×83
34	92	ASTC 10×8	78×108

Table 4 - Font Handles 26 to 34

E. Animation Converter

Convert a **GIF** file or a series of **PNG/JPEG/BMP** files into EVE-compatible animation files.

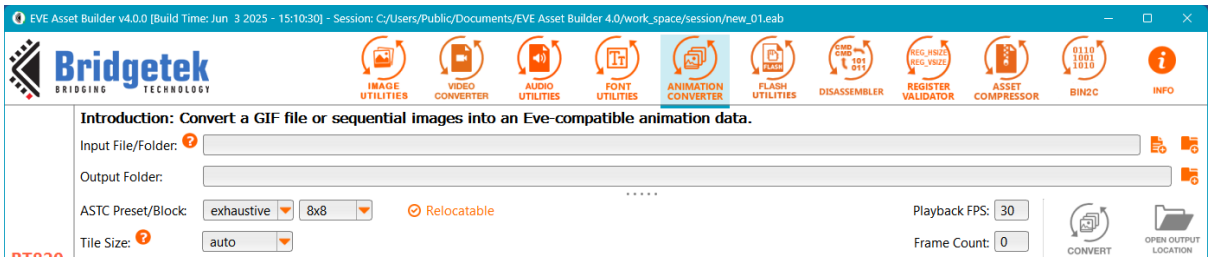


Figure 31 - Animation Converter

Input File/Folder: Users can select either a GIF file or an image folder. The folder must contain PNG, JPEG, or BMP files with sequential names matching the regular expression, e.g., ".*\[0-9]+.png", such as "001.png", "002.png", etc.

Relocatable: see Relocatable Asset

Address: Predefine an offset, which must be a multiple of 64, to load the animation. This option is available when **Relocatable** is unchecked.

ASTC Preset/Block: see ASTC Encoder

Frame Count: Display the number of available frames.

Playback FPS: For previewing the speed of animation.

Tile Size: Tile is the unit to determine the overlap area across frames. The valid options are *auto*, *manual*, or *not apply*.

- *auto*: The optimal tile size will be automatically chosen by EAB.
- *manual*: Users can manually set *Tile Width* and *Height*, which must multiply to a multiple of 4 to meet ASTC rendering requirements.
- *not apply*: Exclude tile application during conversion. Each frame will be converted separately.

Output

- A *.anim.raw* file that can be loaded to RAM_G and used to render animations, or
- A *.anim.reloc* file in relocatable format, generated when the *Relocatable* option is checked.

F. Flash Utilities

1. Hardware Configuration

Programmer Module: Select either FT4222 or MPSSE. Make sure the selected module is present on your device and all required drivers are installed. If your device has no module present, you may need to purchase these modules and connect them with your device properly. At any given time, only one Programmer Module can be utilized to execute a flash operation.

SPI Clock Rate (Mhz): set SPI clock rate that is used to communicate from programmer module to EVE chip.

Flash Chip Type: NOR or NAND. Only BT82X supports NAND flash.

2. Flash Image Generator

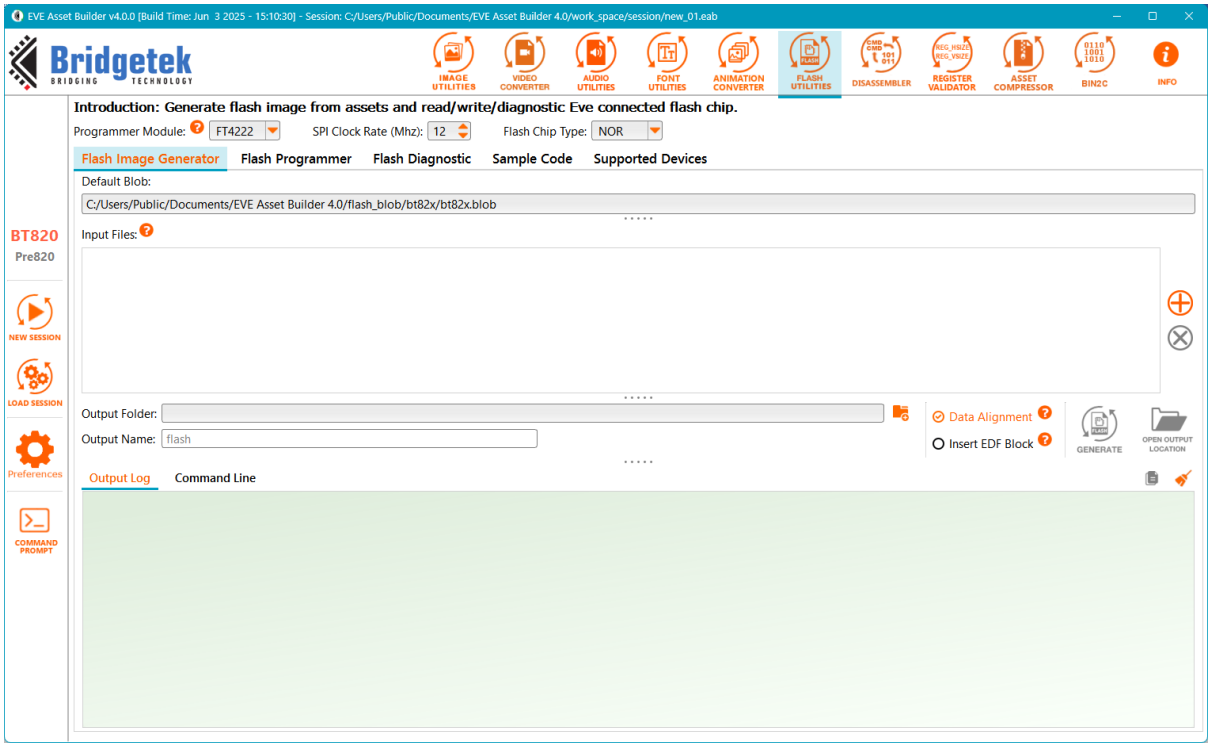


Figure 32 - Generate Flash Image

Default Blob: The 4096-byte flash driver used by the firmware, always located in the first block.

Input Files: Select the assets for flash content.

Data Alignment: By default, each asset is aligned to 64 bytes, and the entire flash file is aligned to 4096 bytes. If left unchecked, alignment is not applied, and users should have a strong reason for doing so.

Insert EDF Block: When enabled, EAB generates and inserts an EDF block after the blob driver. To modify the properties of an asset, users must place the corresponding JSON file in the same folder.

Output Name: Set the generated flash image name.

Output:

- A .bin file to write to the flash chip,
- A .map file keeps the offset and size of each asset as the format: ***<asset name> : <offset> : <size>***,
- An .edf file has the details of each asset.

EDF Block:

The "EVE Flash Description File" (EDF) block is a binary object containing asset metadata in sequential order, located immediately after the blob driver. The EDF structure is defined as follows:

```
struct EDF {
   uint16_t numAsset;      // total number of assets
   uint16_t edfSize;       // size of EVE_Flash_Asset_Info
   EVE_Flash_Asset_Info assets[0...numAsset-1]; // details of n assets
};

struct EVE_Flash_Asset_Info {
      uint16_t assetID;       // the sequential ID of the asset
      uint32_t startAddress;  // the offset of the asset
      uint32_t size;          // the asset size, in byte
      uint8_t  compression;   // 0 = raw, 1 = compress, 2 = relocatable
      uint8_t  type;          // the type of the asset, see below
      uint16_t subType;       // the subtype of the asset, see below
      uint16_t width;         // the width of bitmap/video/animation
      uint16_t height;        // the height of bitmap/video/animation
};
```

assetID denotes the order of an asset in flash image.

width and ***height*** are only meaningful for bitmap, video, and animation. For other types of assets, they will be set to zero.

type and ***subType*** are defined in the following tables.

type	subType	Description
0x01	-	Flash Driver (Blob)
0x02	*Table 6*	Bitmap
0x03	-	Bitmap Paletted Index
0x04	-	Bitmap Paletted Lut
0x05	*Table 7*	DXT1
0x07	-	PNG/JPEG File
0x20	*Table 8*	Animation Pre82X
0x21	-	Animation BT82X
0x30	*Table 9*	Audio Pre82X
0x31	*Table 9*	Audio BT82X
0x50	-	Video Pre82X
0x51	-	Video BT82X

0x70	-	Legacy Font (Pre-BT82X)
0x71	-	Extended Font (Pre-BT82X)
0x80	-	Legacy Font (BT82X)
0x81	-	Extended Font 1 (BT82X)
0x82	-	Extended Font 2 (BT82X)
0xFC	-	Padding Data
0xFD	-	EDF Block
0xFE	-	User Data

Table 5 - Asset Type

subType of Bitmap	Description
0	ARGB1555
1	L1
2	L4
3	L8
4	RGB332
5	ARGB2
6	ARGB4
7	RGB565
17	L2
19	RGB8
20	ARGB8
21	PALETTEDARGB8
22	RGB6
23	ARGB6
24	LA1
25	LA2
26	LA4
27	LA8
28	YCBCR
37808	ASTC_4x4
37809	ASTC_5x4
37810	ASTC_5x5
37811	ASTC_6x5
37812	ASTC_6x6
37813	ASTC_8x5
37814	ASTC_8x6
37815	ASTC_8x8
37816	ASTC_10x5
37817	ASTC_10x6
37818	ASTC_10x8
37819	ASTC_10x10
37820	ASTC_12x10
37821	ASTC_12x12

Table 6 - subType of Bitmap

subType of DXT1	Description
0xdd00	L1_RGB565
0xdd01	L1_PALETTED565
0xdd02	L2_RGB565
0xdd03	L2_PALETTED565

Table 7 - subType of DXT1

subType of Animation	Description
0xaa00	Animation address in Flash (Pre-BT82X)
0xaa01	Animation address in RAM_G (Pre-BT82X)

Table 8 - subType of Animation

subType of Audio	Container	Description
0xad00	RAW	Linear Sample Format
0xad01	RAW	uLaw Sample Format
0xad02	RAW	4-bit IMA ADPCM Sample Format
0xad03	RAW	S16 Sample Format
0xad04	RAW	S16S Sample Format (stereo)
0xad10	WAV	Linear Sample Format
0xad11	WAV	uLaw Sample Format
0xad12	WAV	4-bit IMA ADPCM Sample Format
0xad13	WAV	S16 Sample Format
0xad14	WAV	S16S Sample Format (stereo)

Table 9 - subType of Audio

3. Flash Programmer

Detect Flash

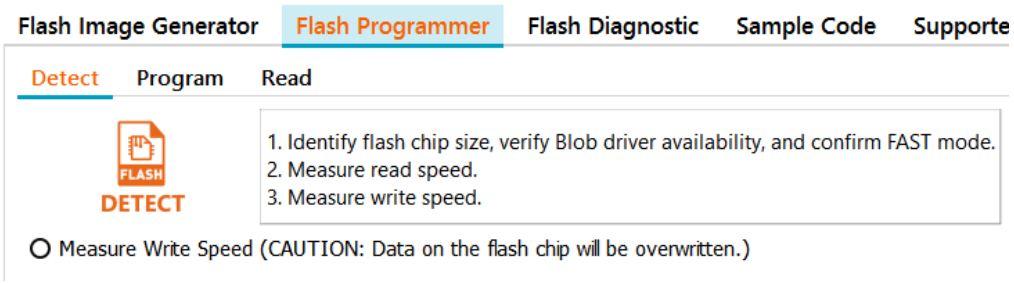


Figure 33 - Detect Flash

Measure Write Speed: The process will involve the writing of dummy bytes to the flash chip, which will result in the overwriting of any pre-existing data.

DETECT: Detect flash information.

Program Flash

Figure 34 - Program Flash – NOR

Figure 35 - Program Flash - NAND

Flash Image: This file will be utilized to program, update, write data to the flash chip, or verify the flash content.

Destination: Must be aligned as described below and within the range of flash chip size.

- PROGRAM: Must be 4096-byte aligned.
- UPDATE: Must be 4096-byte aligned.
- WRITE: Must be 256-byte aligned.

WRITE NAND: Erase the NAND flash chip, then write the flash image.

PROGRAM: Write data to the flash chip in its factory state (all bits set to 1). This operation is two or three times faster, as it does not compare the data before writing.

UPDATE: Updates the flash chip, erasing if necessary. The flash is not completely cleared; only the required partitions are updated.

UPDATE & VERIFY: Performs the update and then compares the updated data with the source data.

VERIFY: Compare the input binary file with flash content.

ERASE: Clears all data from the flash chip.

Read Flash

Read the content of flash chip into .bin file.

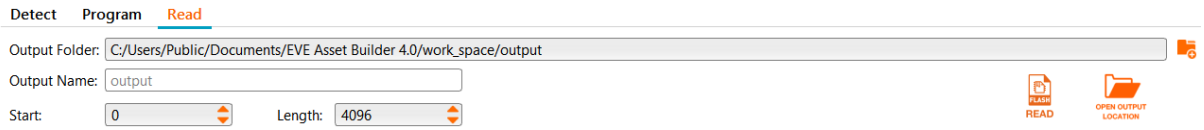


Figure 36 - Read Flash

Output Name: Select the name for the output file.

Start: The offset indicating where to begin reading.

Length: The number of bytes to be read.

READ: Start reading all data in the flash chip to the output file.

4. Flash Diagnostic

The purpose of this feature is to troubleshoot the flash chip.

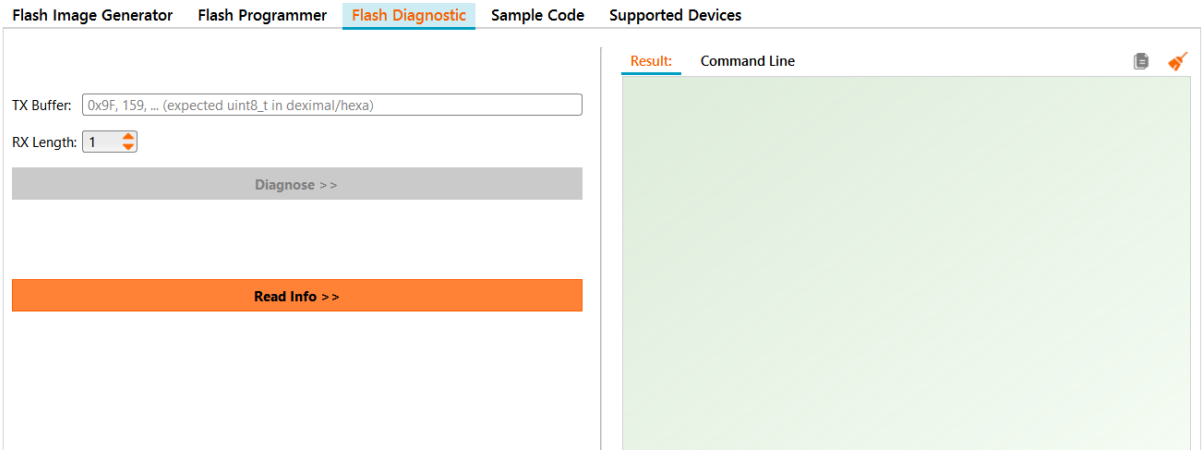


Figure 37 - Flash Diagnosis

Diagnose

TX Buffer: A buffer that will be sent to flash chip, with commas between each byte.

RX Length: Enter the length that you expect to receive from the flash chip.

Note: Refer to the flash chip's manual for details on supported commands and expected data lengths.

Read Info

This will report:

1. Flash RDID
2. Flash status
3. Flash size (in Mbytes and Mbits)

5. Sample Code

The pseudo codes are provided here to showcase the procedure of interacting the EVE connected flash.

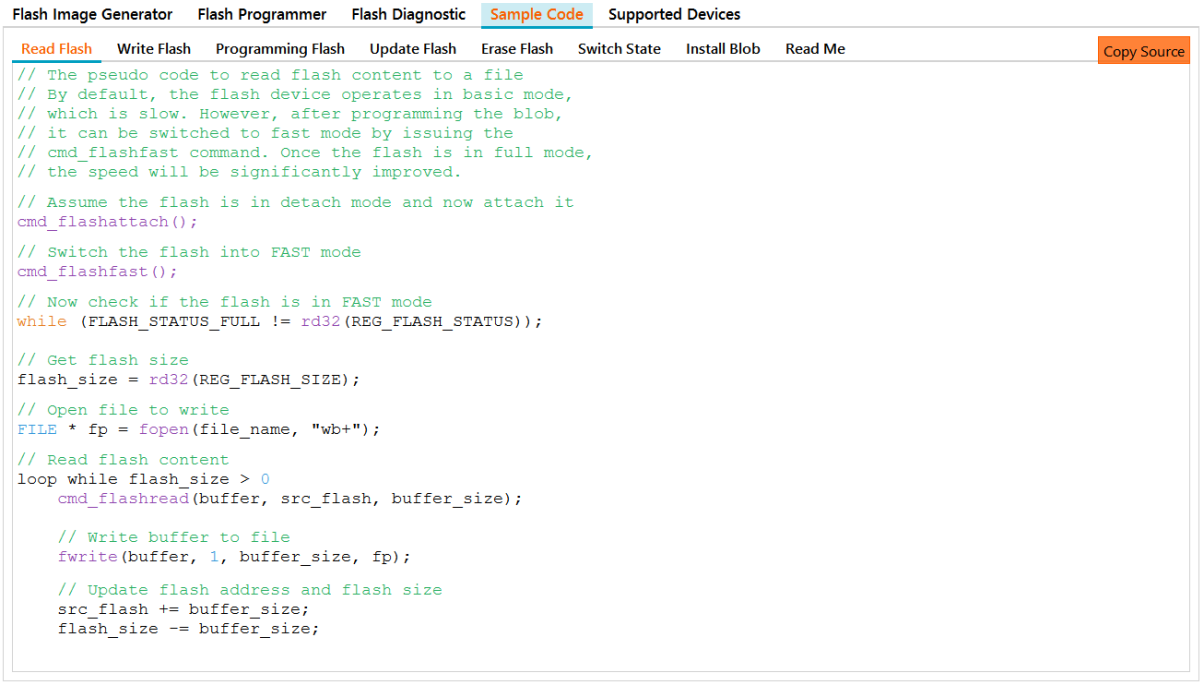


Figure 38 - Flash Sample Code

Copy Source: Copy pseudo code in the current tab to the clipboard.

6. Supported Devices

This tab lists all flash chips supported by BT82Xcan , allowing users to view key details such as NAND/NOR type and flash size.

Flash Image Generator | Flash Programmer | Flash Diagnostic | Sample Code | Supported Devices

Device Prefix	Technology	Manufacturer	Size Mbits	Size MBytes
AS5F31G04SND	NAND	Alliance	1024	128
AS5F12G04SND	NAND	Alliance	2048	256
AS5F32G04SND	NAND	Alliance	2048	256
AS5F14G04SND	NAND	Alliance	4096	512
AS5F34G04SND	NAND	Alliance	4096	512
AS5F18G04SND	NAND	Alliance	8192	1024
AS5F38G04SND	NAND	Alliance	8192	1024
MX35LF2	NAND	Macronix	2048	256
MX35LF4	NAND	Macronix	4096	512
MT29F2G01	NAND	Micron	2048	256
W25N01GV	NAND	Winbond	1024	128
S25FL064L	NOR	Cypress	64	8
S25FL1	NOR	Cypress	64	8
GD25Q64C	NOR	Gigadevice	64	8
IS25WP064	NOR	ISSI	64	8
IS25LP256	NOR	ISSI	256	32
MX25	NOR	Macronix	64	8
MX25L64	NOR	Macronix	64	8
MX25L256	NOR	Macronix	256	32
MT25QL128	NOR	Micron	128	16
W25Q	NOR	Winbond	64	8
W25Q128J	NOR	Winbond	128	16

Figure 39 - Supported Flash Chip

G. Asset Compressor

1. Compressor

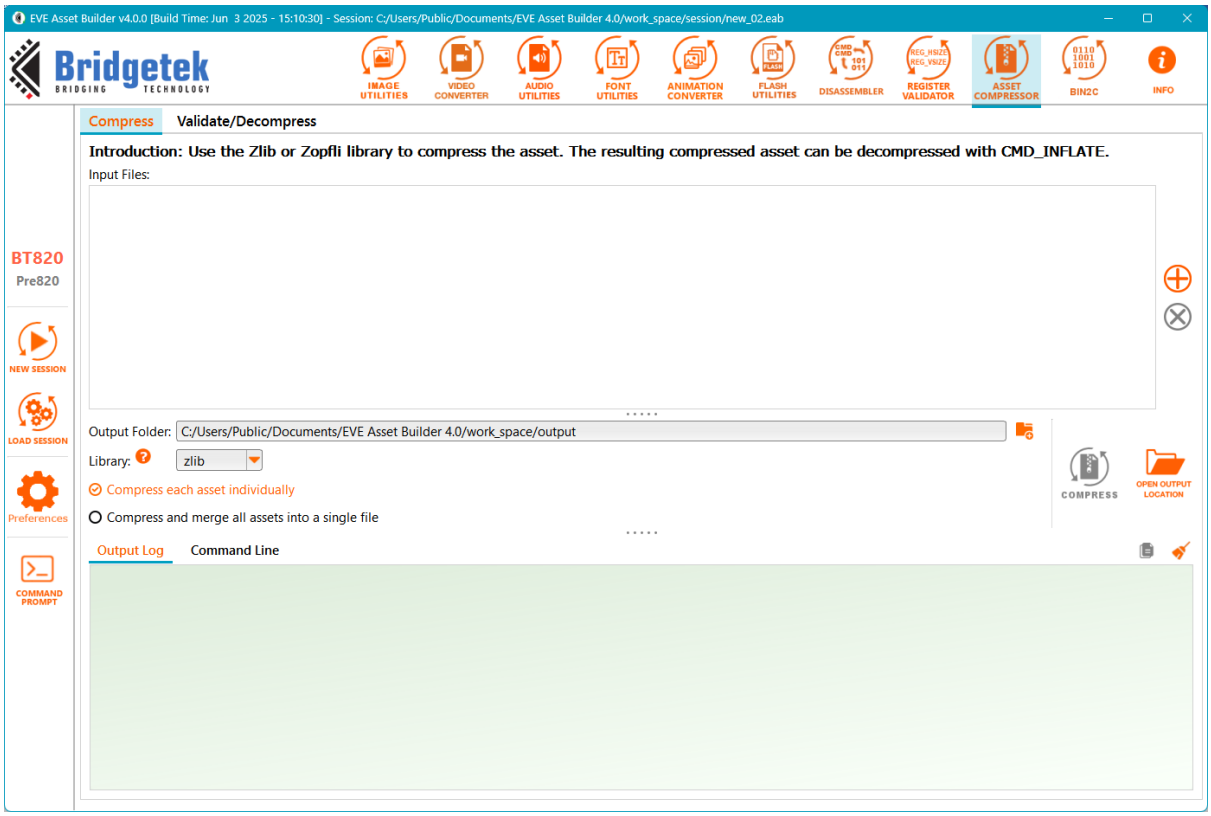


Figure 40 - Asset Compressor

Input Files: Add asset files to compress.

Library: Select library to compress. They are **zlib** and **zopfli.**

Compress each asset individually: Each input will have one compressed output.

Compress and merge all assets into one file: All inputs will be compressed and combined into only one file.

Note: Zopfli achieves higher compression than Zlib but takes much longer to perform the compression. The decompressing speed of Zopfli's output is practically unaffected compared to Zlib's.

2. Validator/Decompressor

Check if the input assets are valid for ***cmd_inflate***. This tool supports decompressing the assets if required.

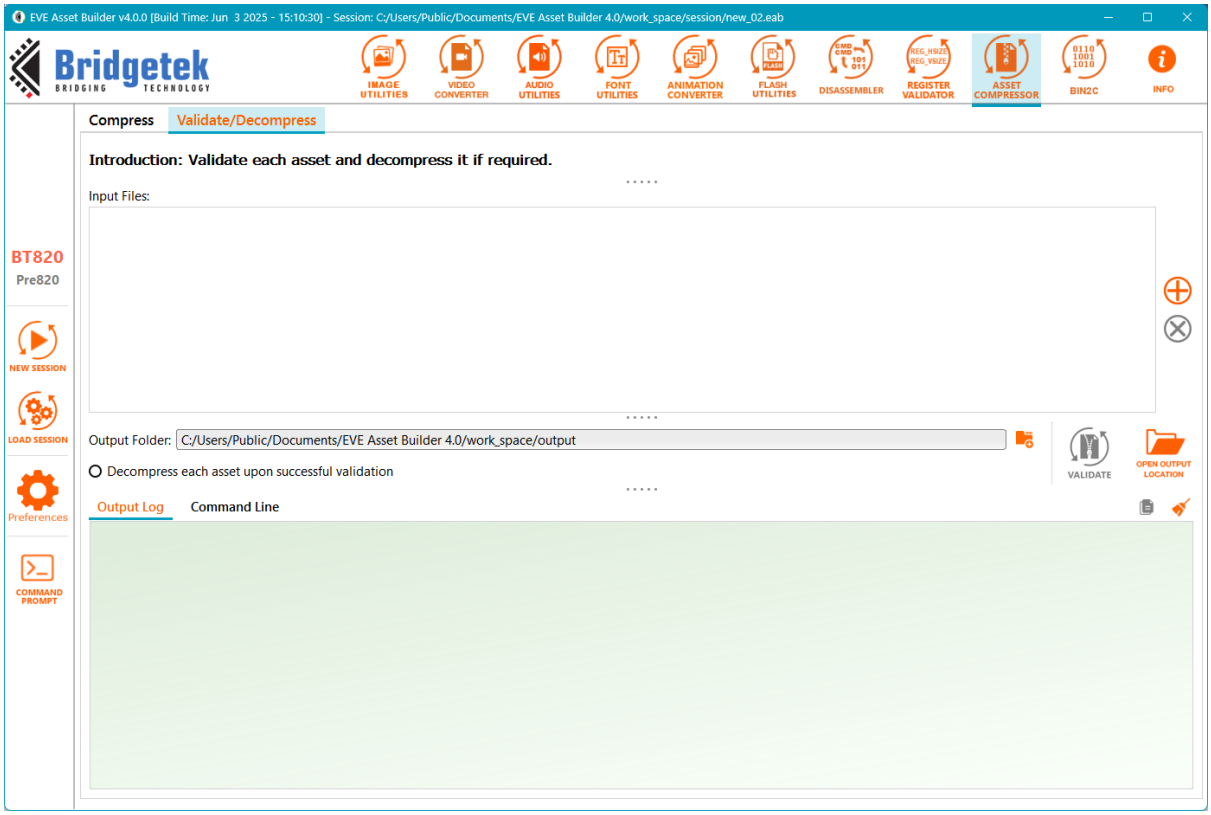


Figure 41 - Compressed Asset Validator

Input Files: Add files to validate.

Decompress each asset upon successful validation: Unpack an asset that passed validation when ticked.

VALIDATE: Verify the validity of the input files for ***cmd_inflate***, and proceed with decompressing them if required.

H. Binary To C-Array Converter

Bin2C is a utility designed to transform a binary file into a C array. The generated output is a compilable C source file. It supports 1, 2, and 4-byte data types, allowing users to specify the endianness (little or big) for *word* (2 bytes) and *long* (4 bytes) data types.

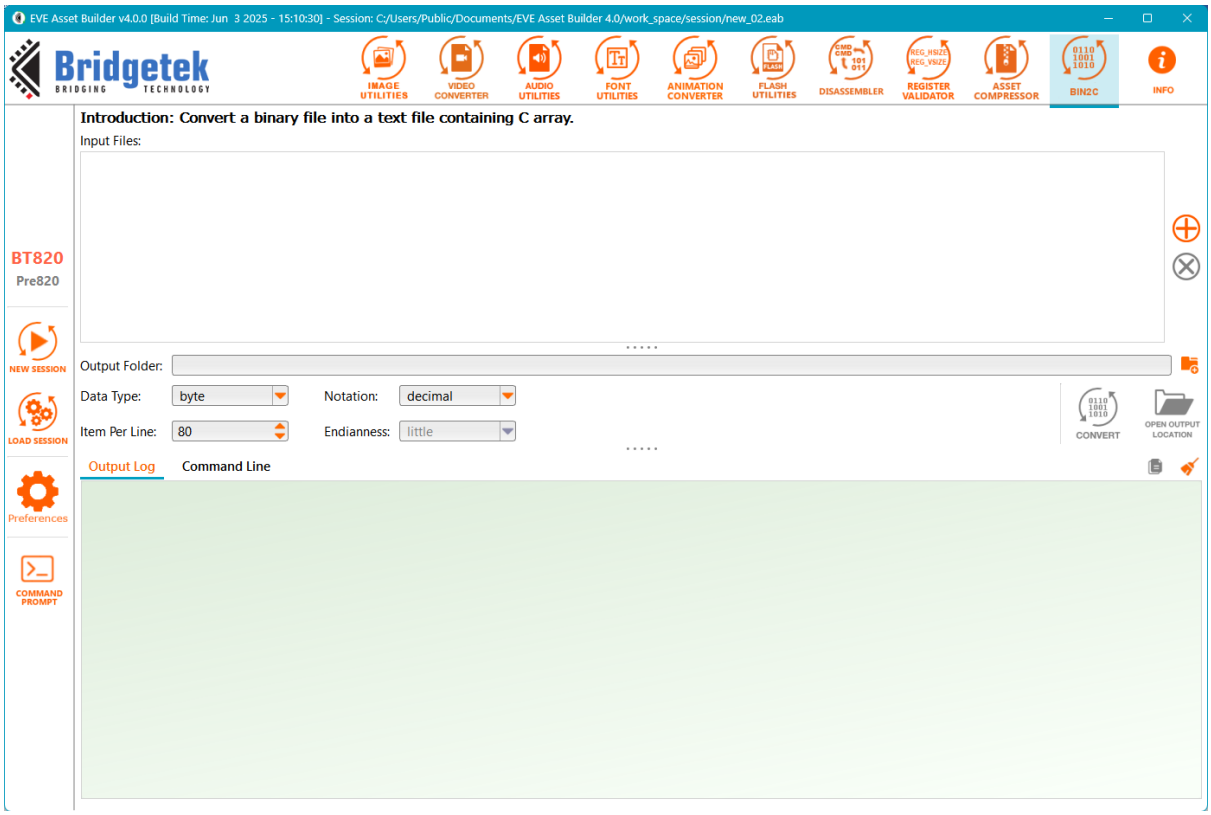


Figure 42 - Bin2C Converter

Input Files: A list of binary files to be converted.

Data Type: Set data type of the C array, which can be byte, word (2 bytes), or long (4 bytes).

Notation: Set the number notation as either decimal (base 10) or hexadecimal (base 16).

Item Per Line: The number of array items to display on each line.

Endianness: Select little-endian or big-endian when reading the binary file.

I. Disassembler

This tool disassembles a binary file or a text file containing a string of hexadecimal digits to human-readable command text. All display list and coprocessor commands are supported.

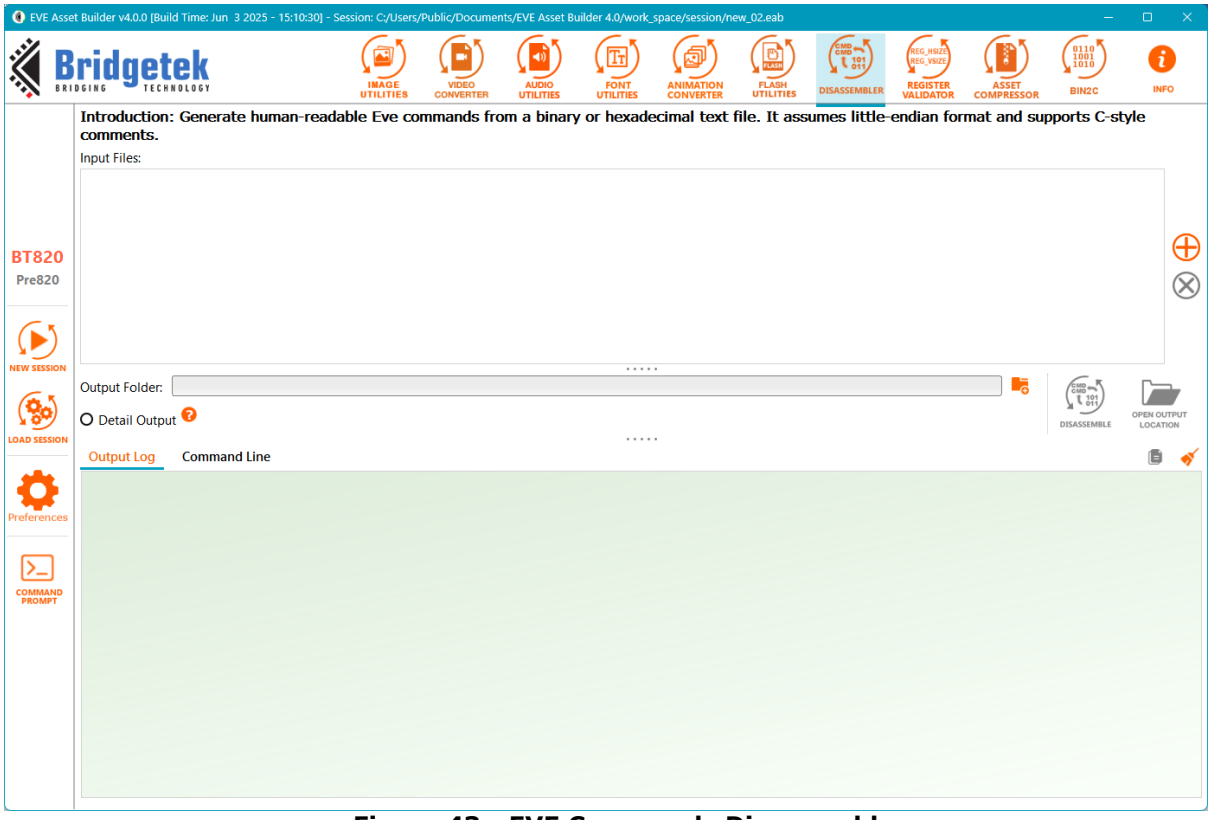


Figure 43 - EVE Commands Disassembler

Input Files: List of files to be disassembled. Little-endianness is assumed. C-style comments allowed.

Detail Output: When checked, the output format will be the same as screenshot below.

```
Address    Value (Hex)    Instruction       Parameter      Interpretation Of Parameter
   0       0x26000007     CLEAR             0x0000 0007    CLEAR(1, 1, 1)
   4       0x22000000     SAVE_CONTEXT      -              SAVE_CONTEXT()
   8       0x27000002     VERTEX_FORMAT     0x0000 0002    VERTEX_FORMAT(2)
  12       0x0500001C     BITMAP_HANDLE     0x0000 001C    BITMAP_HANDLE(28)
  16       0x1F000001     BEGIN             0x0000 0001    BEGIN(BITMAPS)
  20       0x95C87E54     VERTEX2II         0x15C8 7E54    VERTEX2II(174, 135, 28, 84)
  24       0x97887E65     VERTEX2II         0x1788 7E65    VERTEX2II(188, 135, 28, 101)
  28       0x98E87E78     VERTEX2II         0x18E8 7E78    VERTEX2II(199, 135, 28, 120)
  32       0x9A487E74     VERTEX2II         0x1A48 7E74    VERTEX2II(210, 135, 28, 116)
  36       0x23000000     RESTORE_CONTEXT   -              RESTORE_CONTEXT()
  40       0x00000000     DISPLAY           -              DISPLAY()
```

Figure 44 - Disassembler - Detail Output

Sample inputs

- **Binary:**

copro.bin

0000014b	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00000000	07	00	00	26	4e	ff	ff	ff	00	10	00	00	50	ff	ff	ff
00000010	00	00	00	00	40	00	00	00	43	ff	ff	ff	00	00	00	00
00000020	00	00	b5	00	b9	00	00	00	01	00	00	1f	00	60	23	91

Decode

copro.decode.txt - WordPad

File Home View

```
CLEAR(1, 1, 1)
CMD_FLASHSOURCE(4096)
CMD_INFLATE2(0, OPT_FLASH)
CMD_SETBITMAP(ARGB1555, 0, 181, 185)
BEGIN(BITMAPS)
VERTEX2II(137, 54, 0, 0)
```

- **Hexadecimal string:**

hexa_string.txt

File Home Vie

```
0x26000007
0x22000000
0x27000002
0x0500001c
0x1f000001
0x95c87e54
0x97887e65
0x98e87e78
0x9a487e74
0x23000000
0x00000000
```

Decode

hexa_string.decode.txt - WordPad

File Home View

```
CLEAR(1, 1, 1)
SAVE_CONTEXT()
VERTEX_FORMAT(2)
BITMAP_HANDLE(28)
BEGIN(BITMAPS)
VERTEX2II(174, 135, 28, 84)
VERTEX2II(188, 135, 28, 101)
VERTEX2II(199, 135, 28, 120)
VERTEX2II(210, 135, 28, 116)
RESTORE_CONTEXT()
DISPLAY()
```

J. Register Validator

The tool is designed to validate sets of register values, review them, and highlight the mandatory requirements for these display configuration registers.

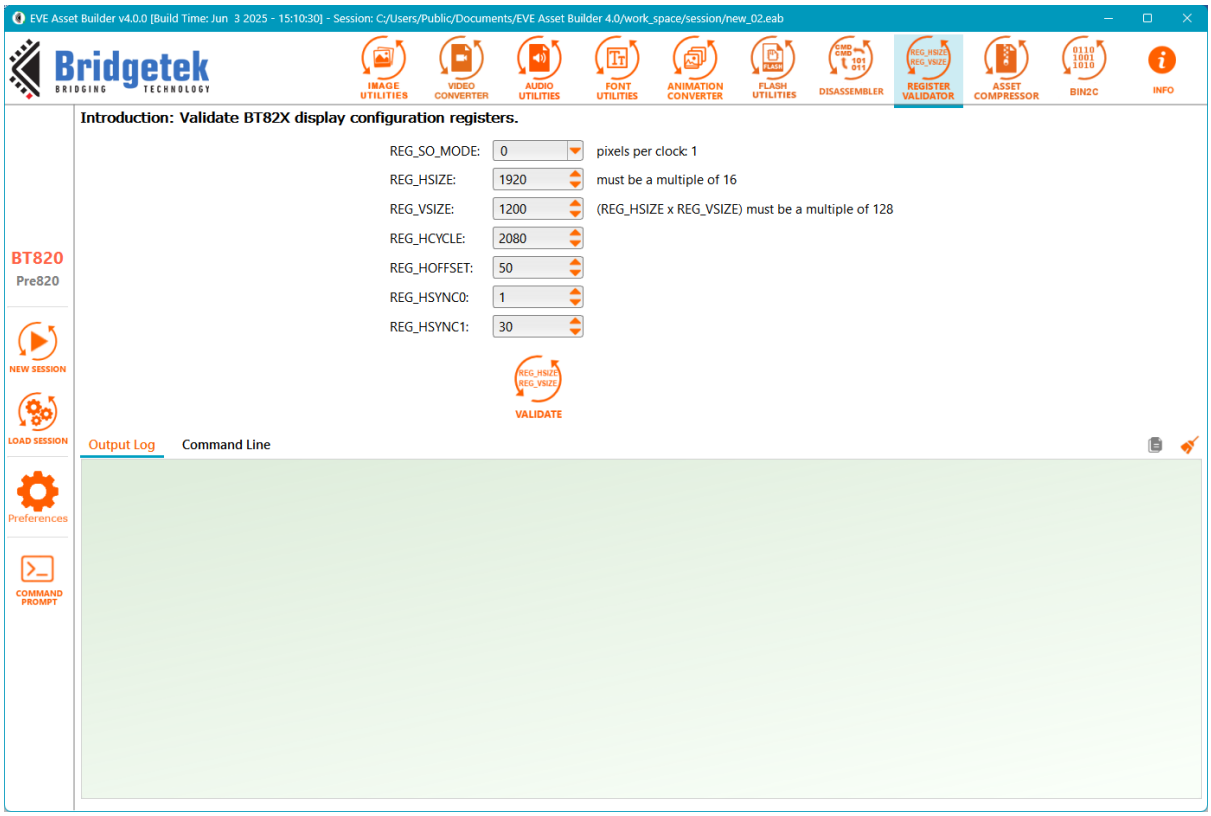


Figure 45 - Register Validator

VALIDATE: Check the register values and report errors.

VII. Utilities For Pre82X

A. Image Utilities

Within this utility, there are three-tab pages that offer a range of image-related functionalities. The first tab **Converter** is dedicated to the conversion of PNG, JPEG, and BMP images to the EVE-compatible image format. The second tab **PNG/JPEG Validator** is designed for the purpose of assessing the compatibility of **PNG** and **JPG** files with ***cmd_loadimage***. The third tab **JPEG Optimizer** allows users to reduce the size of input JPEG files while maintaining their EVE-compatibility.

1. Image Converter

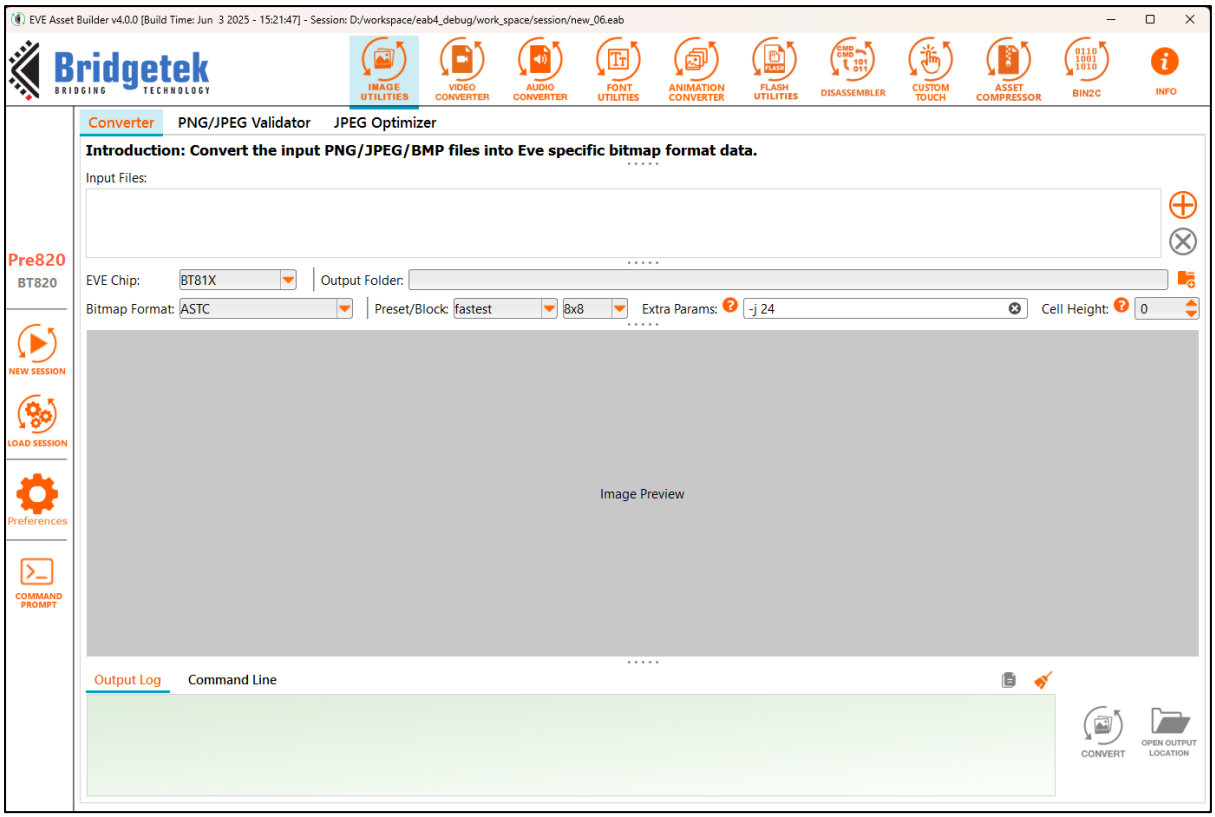


Figure 46 - Image Converter

Input Files: PNG/JPG/BMP images to be converted. Users can change the order of images by using *Ctrl+Up* or *Ctrl+Down*.

EVE Chip: Refer to EVE Chip Series

Bitmap Format: The output format of images is one of the following:

- ARGB1555, RGB565, ARGB4
- RGB232, ARGB2,
- L1, L2, L4, L8,
- PALETTED565, PALETTED4444, PALETTED8
- ASTC
- DXT1_L1_RGB565, DXT1_L1_PALETTED565
- DXT1_L2_RGB565, DXT1_L2_PALETTED565

Dithering: Enabling dithering can help reduce noise levels, but it may negatively affect compression efficiency.

Figure 47 - Image Converter - ASTC Options

ASTC Preset/Block: see ASTC Encoder

Extra Params: Users can explore various options to determine the optimal balance for their specific needs, considering factors like memory availability, performance demands, and visual quality.

Cell Height: Specifies the cell height used in Multi-Cell Bitmap mode. EAB includes padding data in ASTC-based multi-cell bitmaps to ensure each cell aligns to a 64-byte boundary in memory, enabling proper rendering on EVE chips.

2. PNG/JPEG Validator

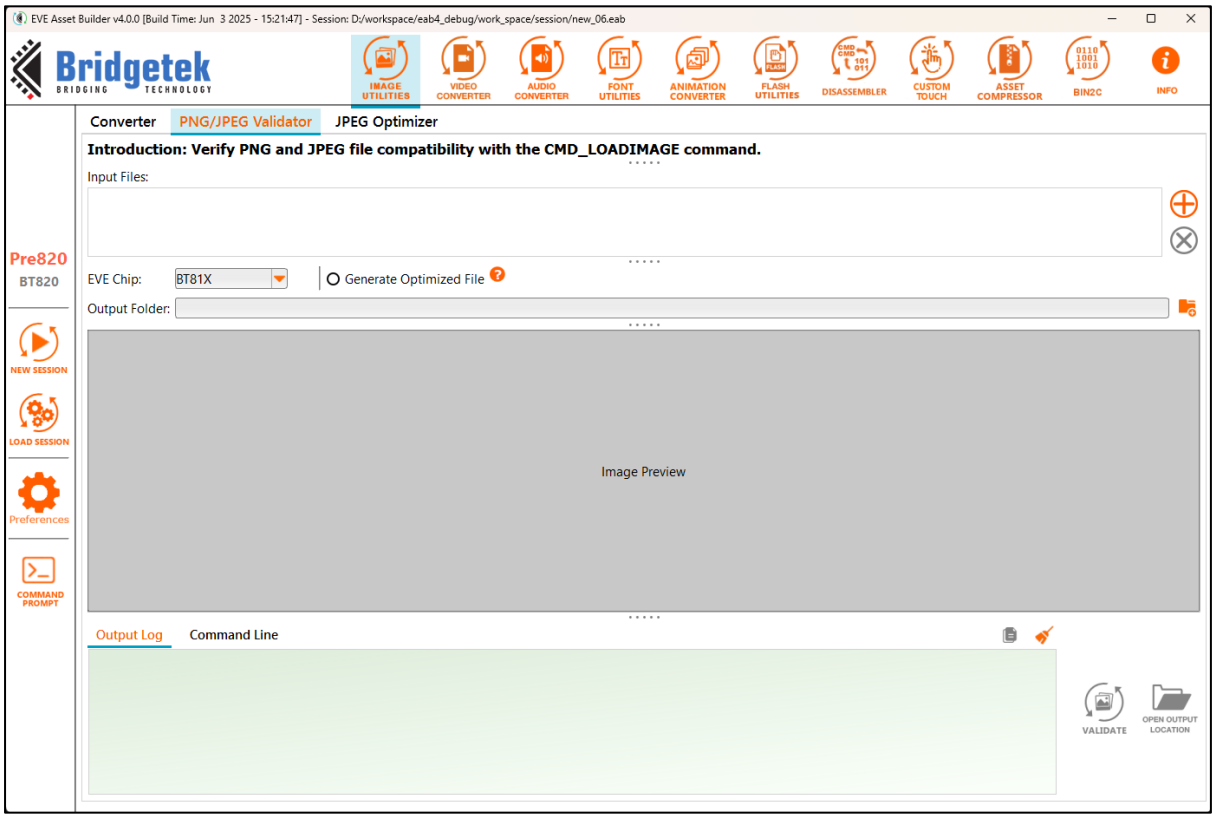


Figure 48 - PNG/JPEG Validator

EVE Chip: Refer to EVE Chip Series

Generate Optimized File:

☑ If checked, input images are optimized and saved to the output folder. EAB then validates the optimized images.

☐ If unchecked, EAB validates the original input images without performing optimization.

3. JPEG Optimizer

This tool reduces JPEG file size by adjusting quality or converting images to grayscale.

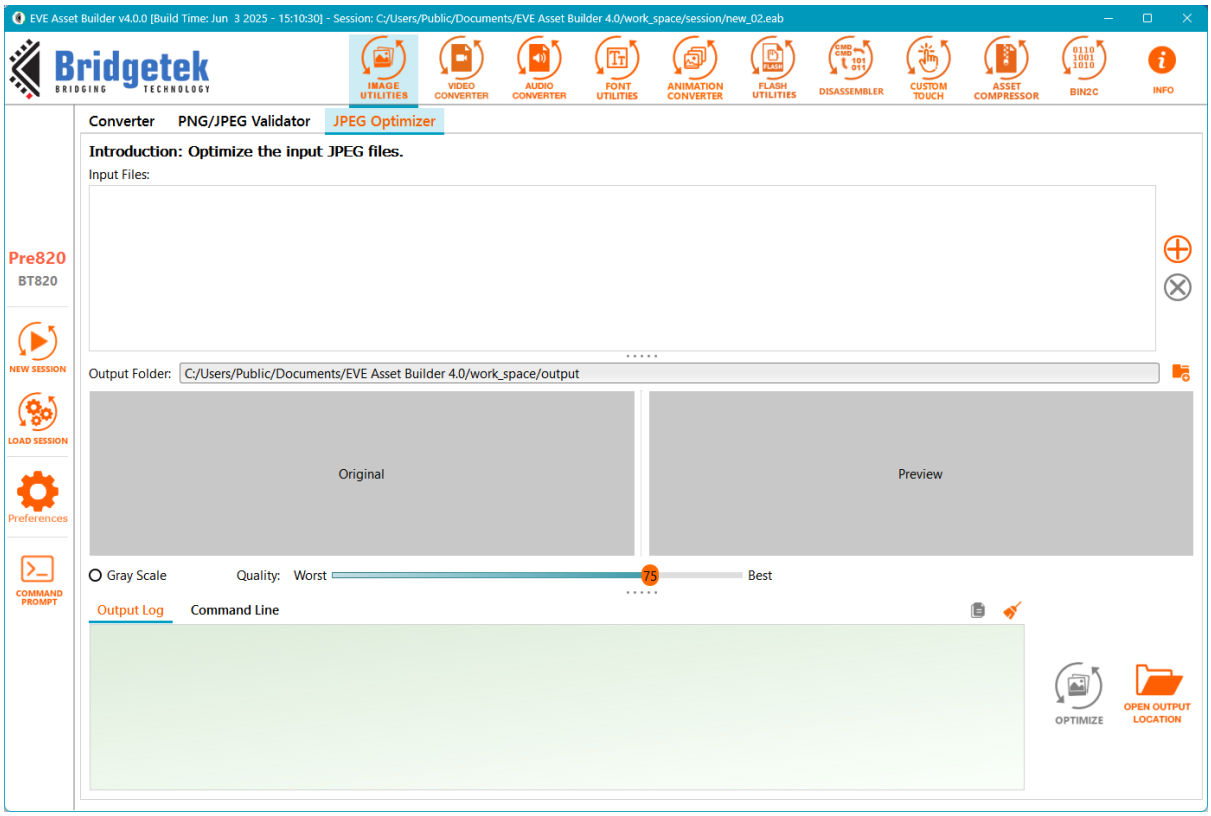


Figure 49 - JPEG Optimizer

Gray Scale: Transform the image into grayscale.

Quality: From 0-95. Reducing image size significantly also results in the worst quality.

OPTIMIZE: Apply optimization to each JPEG image.

B. Video Converter

Convert video file into **MJPEG** format and mono audio channel. Output is an AVI file and a sample project to guide how to playback video on EVE device.

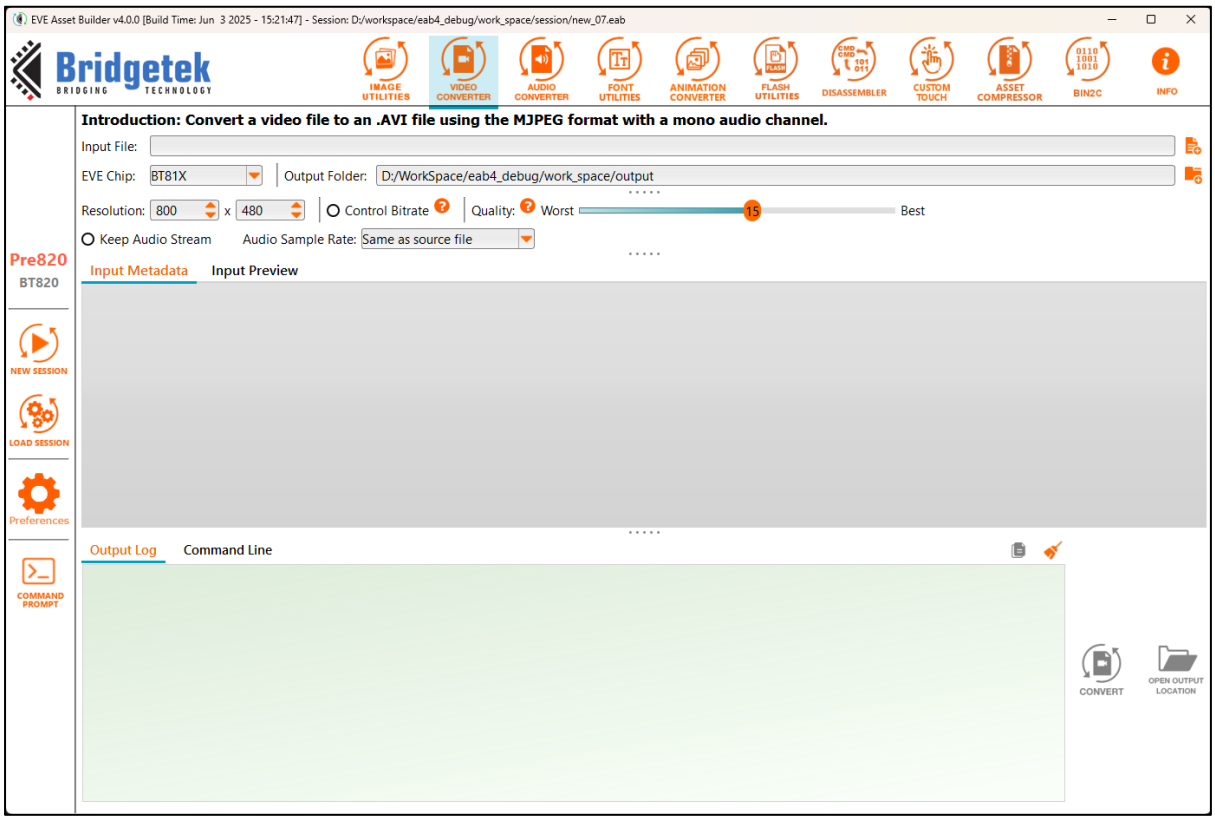


Figure 50 - Video Converter

Input: The original video to be converted.

EVE Chip: Refer to EVE Chip Series.

Note: A warning will be shown if the product of width, height, and 2 (representing 2 bytes per pixel) exceeds the capacity of RAM_G.

Resolution: Must be specified. The video resolution will be adjusted to this size.

Quality: From 1 (best) to 31 (worst).

Control Bitrate: Allow users to control the output bitrate. Enabling this mode will disregard the *Quality* setting.

- Average (kbit/s): Set the average bit rate for the output video
- Max (kbit/s): Set max bit rate for the output video
- Buffer Size (kbit): Set buffer size for the output video

Keep Audio Stream: Users can keep or remove the audio stream.

Audio Stream: see Audio Format and Sample Rate

C. Audio Converter

Convert a WAV or MP3 audio file into a format optimized for EVE devices, ensuring seamless processing and playback.

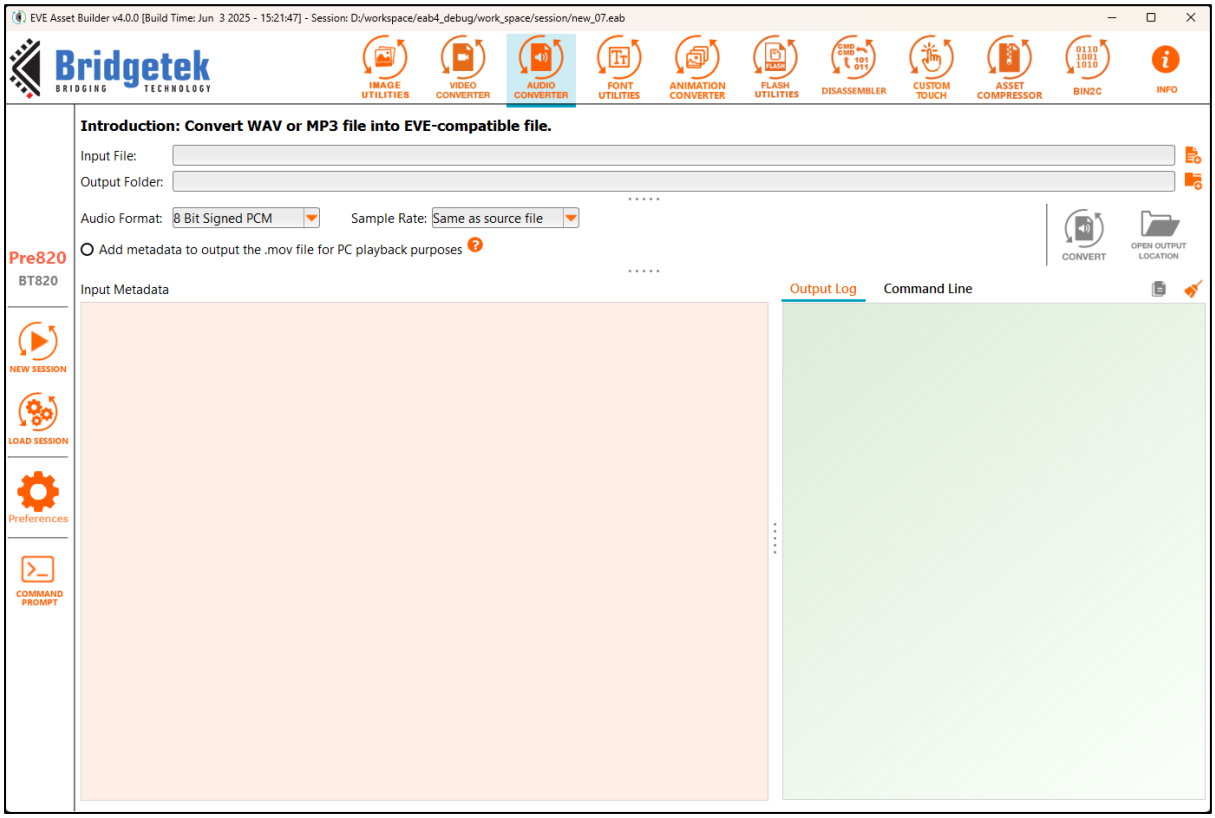


Figure 51 - Audio Converter

Input File: Choose the audio file to be converted.

Audio Format, Sample Rate: see Audio Format and Sample Rate

Add metadata to output the .mov file for PC playback purposes: The converted file requires a header for PC playback, but this header makes it unplayable on EVE devices. This option is designed to generate an audio output for testing on PCs before final use.

D. Font Utilities

1. Font Converter

Extract characters from the OpenType or TrueType font file into a specific font structure. Unprintable characters will be filtered out. Since each glyph is converted to a bitmap individually, combination characters are not supported.

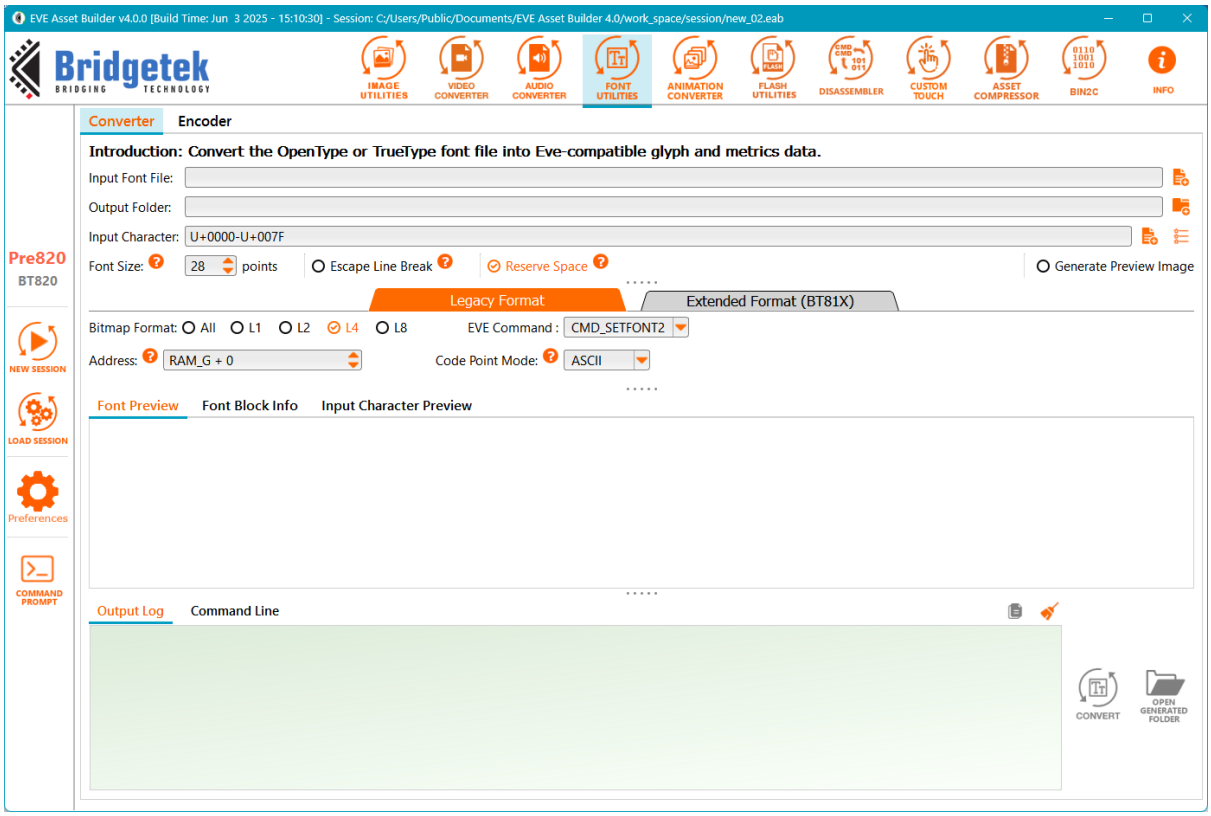


Figure 52 - Font Converter

Input Font File: Font file to be converted.

Input Character: Choose the character set that will be converted.

Font Size: Specify the intended font size. Any character exceeding 255 pixels in width will be eliminated.

Escape Line Break: BT817/8 chip support to overwrite line break. The ASCII code 0x0A is interpreted as a regular code point instead of a line break.

Reserve Space: Allocate the code point 0x20 for the space character, for ***cmd_fillwidth*** to function properly.

Generate Preview Image: These images help users check if the characters are rendered as expected.

Font Preview Area: Display sample characters by using the selected font.

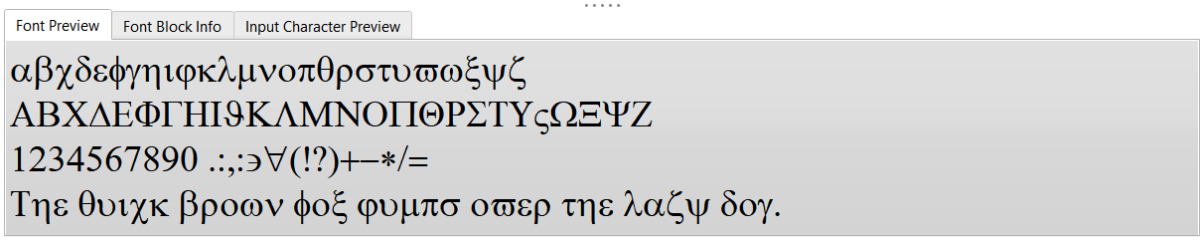


Figure 53 - Font Preview Area

Font Block Info:

- Show the number of glyphs that are valid for the selected font.
- Show a statistic of the selected font, including Block Name, Unicode Range, and Valid Glyphs/All Glyphs.

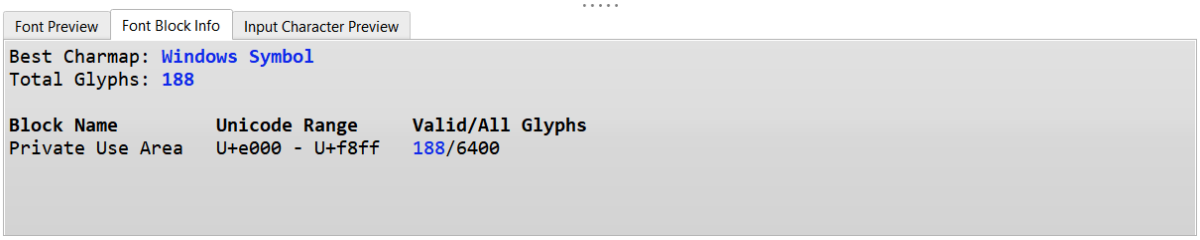


Figure 54 - Font Block Info

Input Character Preview:

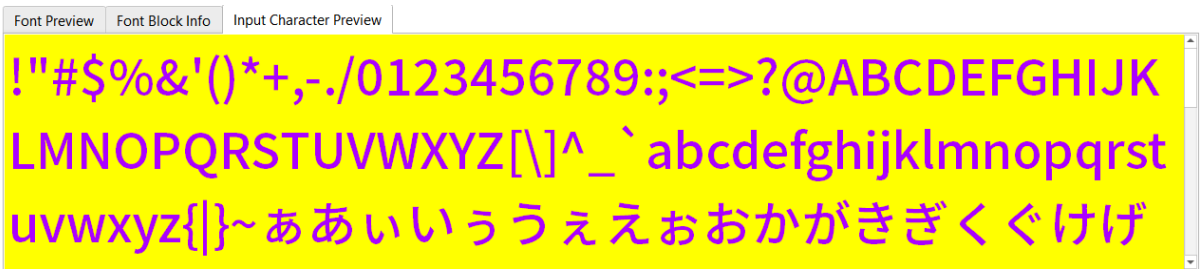


Figure 55 - Input Character Preview

Legacy Format

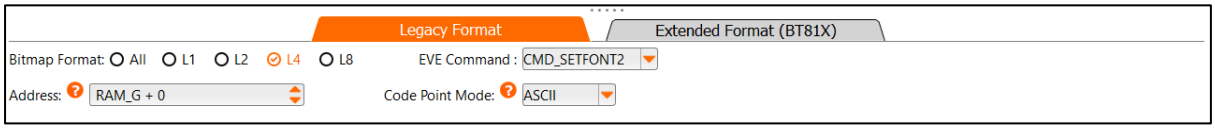


Figure 56 - Legacy Format

Bitmap Format: Any combination of bitmap types L1, L2, L4, L8

EVE Command: Select ***cmd_setfont*** or ***cmd_setfont2***. Converted files will be compatible with the chosen command.

Code Point Mode: Select **ASCII** or **Ordinal**. In ASCII mode, the First Character will be fixed to 32, and the Input Character will be fixed to U+0000-U+007F.

Address: The converted font will be loaded to the offset address in RAM_G, which must be a multiple of 4.

First Characters: Set the first index for converted characters. The valid range is 1-127.

Output:

- The output consists of a 148-byte metric block along with raw bitmap data.
- In addition, this tool generates example C code to demonstrate usage.
- The output data is formatted for a single bitmap handle.

Extended Format (BT81X)

Convert the input characters into extended font format, which can handle a full range of Unicode codepoint. This format supports BT81X EVE series.

Figure 57 - Extended Format (BT81X)

Bitmap Format: Any combination of bitmap types L1, L2, L4, L8 and ASTC.

Code Point Mode:

- UTF-8: Addressing the characters with UTF-8 code point.
- Ordinal: Addressing the characters with index 0, 1, 2, …, n

Address: Address of glyph data. For ASTC, the location can be RAM_G or FLASH. For L1->L8, the location must be in RAM_G.

Note: The maximum number of input characters is limited to 8192 due to the size of RAM_G. This is determined by dividing 1 MB of RAM_G by 128-byte glyphs.

- **ASTC**: 8192 characters
- **L8**: 8192 characters
- **L4**: L8 * 2
- **L2**: L8 * 4
- **L1**: L8 * 8

ASTC Preset/Block: see ASTC Encoder

Note: For *auto* option, block footprint depends on font size:

- If font size > 52: block footprint = 10x8
- If font size < 19: block footprint = 8x5
- Otherwise: block footprint = 8x8

FG/BG Color: Specify the foreground/background color for the converted characters. These colors are applicable only for ASTC format. The default colors are white/transparent if not set.

Output: .glyph file contains graphic data and .xfont file contains the font metrics block.

E. Animation Converter

Convert a **GIF** file or a series of **PNG/JPEG/BMP** files into EVE-compatible animation files. Animation is supported by the BT81X chip and above.

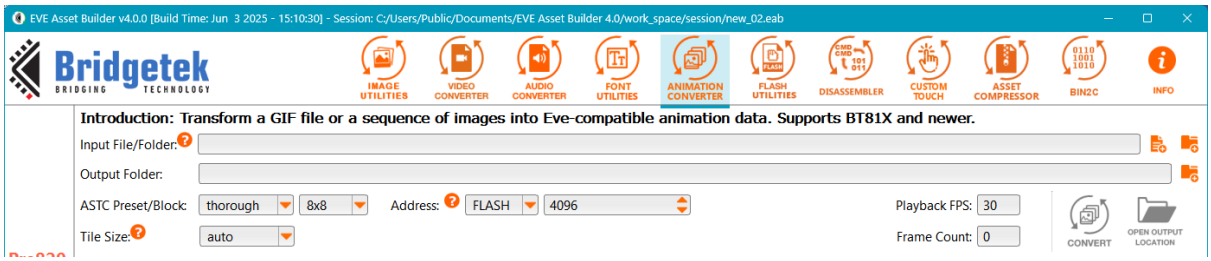


Figure 58 - Animation Converter

Address: Select either **FLASH** or **RAM_G**. The address offset must be a multiple of 64. Starting from BT817/8, users can load animation assets to RAM_G to improve rendering speed.

Notes: Refer to Animation Converter for details on other fields.

Output

- An *.anim.flash* file that can be directly written to flash and then used to render animations, or
- An "*.anim.ram_g*" to load to RAM_G.

F. Flash Utilities

1. Common

EVE Chip: Select either BT815/6 or BT817/7A/8 based on the board.

Programmer Module: can be ***FT4222***, ***MPSSE***, or ***Raspberry Pi Pico.***
Make sure the selected module is present on your device and all required drivers are installed. If your device has no module present, you may need to purchase these modules and connect them with your device properly. At any time, only one Programmer Module can be used.

SPI Clock Rate (Mhz): set SPI clock rate that is used to communicate from programmer module to EVE chip.

2. Flash Image Generator

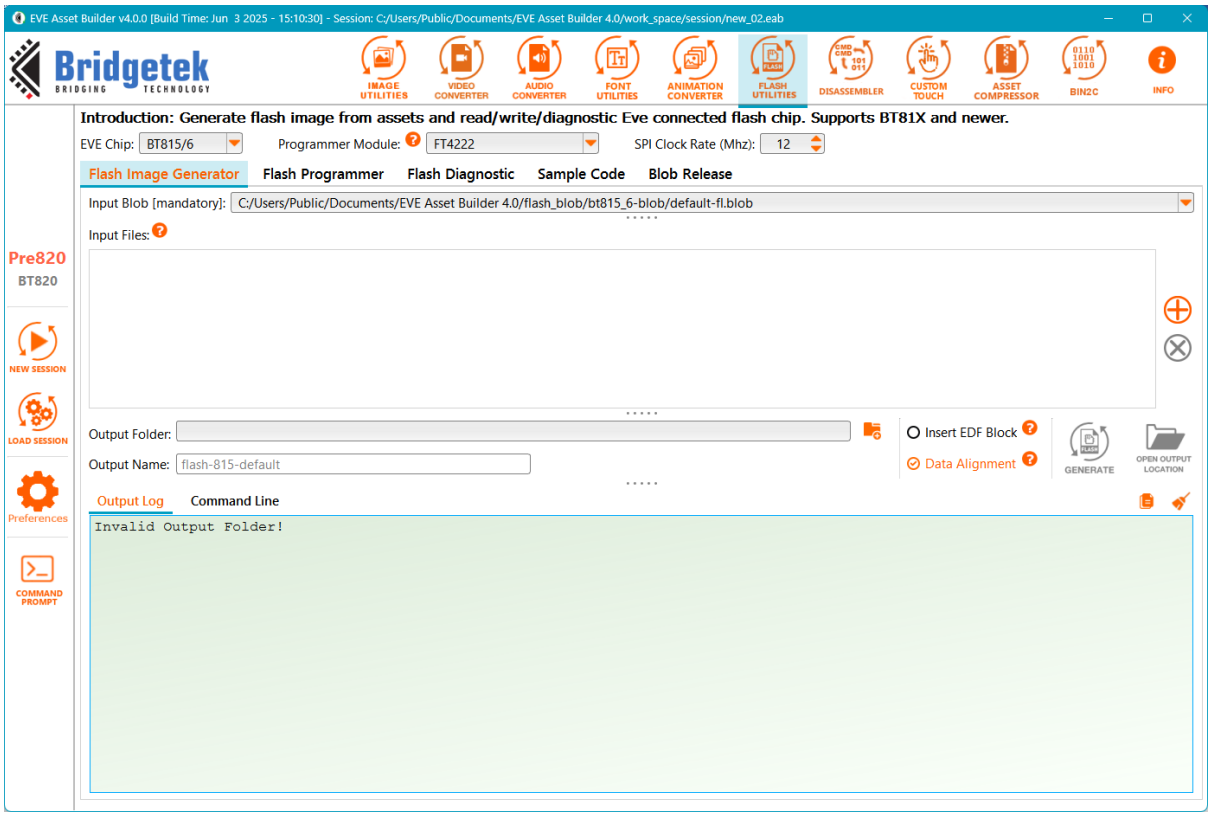


Figure 59 - Flash Image Generator

Input Blob: The flash driver used by firmware, always on the first block (4096 bytes).

Input Files: Select the files for flash content.

Data Alignment: If left unchecked, alignment is not applied. By default, each asset is aligned to 64 bytes, and the entire flash file is aligned to 256 bytes.

Insert EDF Block: EDF block will be generated and inserted after the blob driver.

Output Name: Set the generated flash image name.

Output: A .bin file to write to the flash chip,
A .map file keeps the offset and size of each asset as the format:
<asset name> : <offset> : <size>,
An .edf file has the details of each asset, as described in EDF Block.

Note:

- To update the address of the graphic data in the xfont file, users should place it in the same folder as the glyph file.
- To include xfont in the flash bin, the user should place xfont after the corresponding glyph.
- To modify the type and subtype of a raw/bin file, users need to position the corresponding json file in the same folder.

3. Flash Programmer

Detect Flash

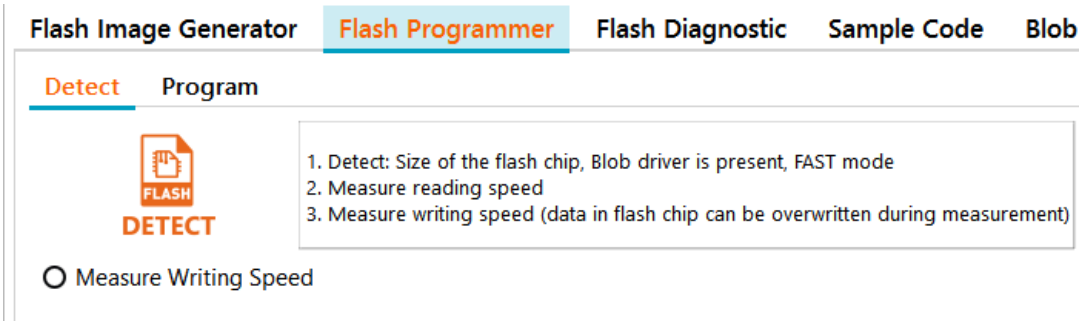


Figure 60 - Detect Flash

Measure Writing Speed: The process will involve writing dummy bytes to the flash chip, which will overwrite any pre-existing data.

DETECT: Detect flash information, read speed, and write speed.

Program Flash

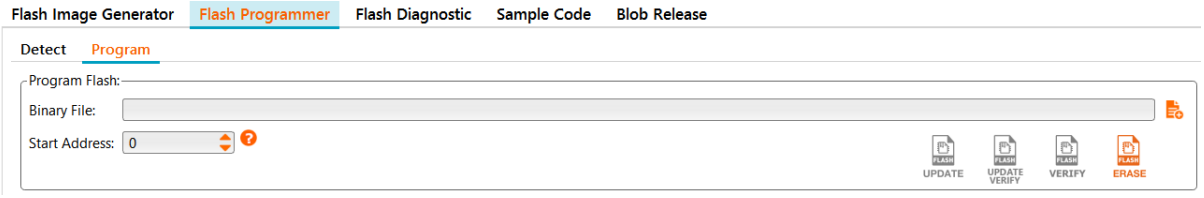


Figure 61 - Program Flash

Start Address: Must be 4096-byte aligned and within the range 0-256 Mbytes.

Binary File: Select file to write to flash chip or to verify flash content.

ERASE: Erase all data in the flash chip.

UPDATE: Update data to the flash chip, erasing if necessary. Flash is not cleared out completely, only updates the required partitions.

UPDATE VERIFY: Same as Update, then compare the updated data to source data.

VERIFY: Compare the input binary file with flash content.

PROGRAM: Write directly to the factory-state flash chip. It is roughly two to three times faster since it bypasses data comparison before writing. This feature is available for BT817/8 and newer versions.

Read Flash

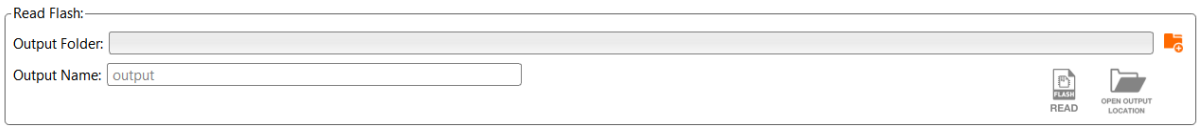


Figure 62 - Read Flash

Output Name: Input a name for the output file.

READ: Copy all data from the flash chip to the output file.

Note: The entire flash chip content is read. Therefore, the output file size will be the same as the flash chip size.

4. Flash Diagnostic

The purpose of this feature is to troubleshoot the flash chip.

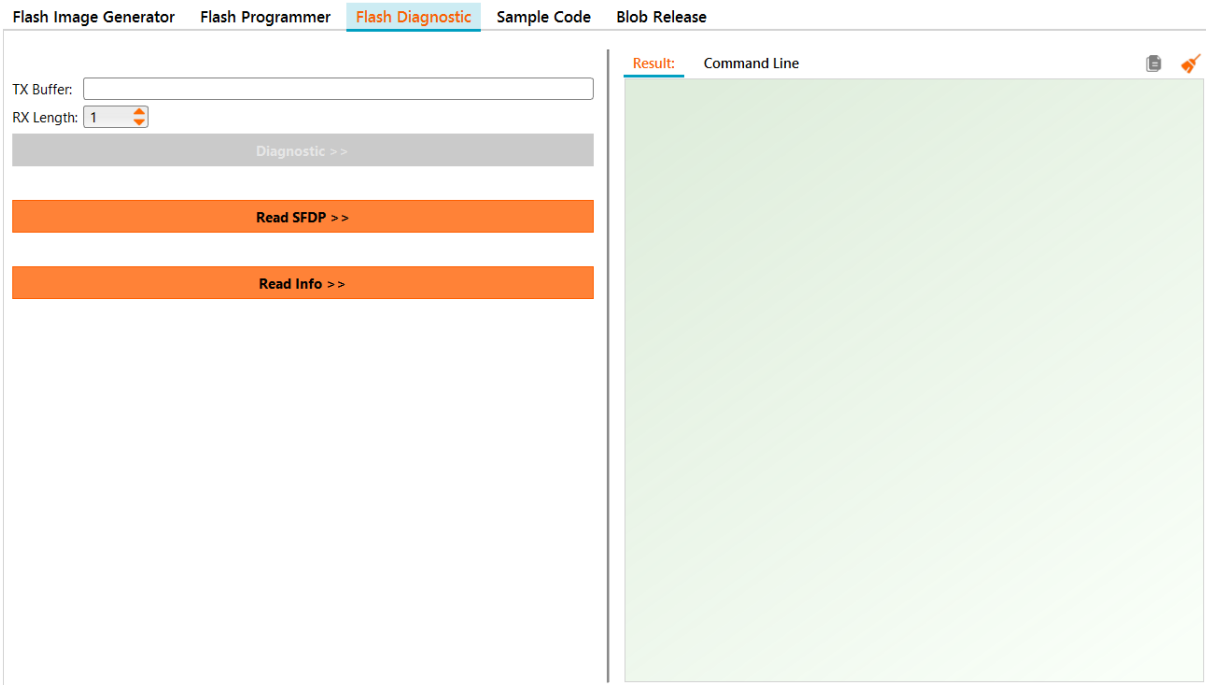


Figure 63 - Flash Diagnostic

Diagnoses

TX Buffer: A buffer that will be sent to flash chip, with commas between each byte.

RX Length: Enter the length that you expect to receive from the flash chip.

Note: Refer to the flash chip's manual for details on supported commands and expected data lengths.

Read Info

This will report:

1. Flash RDID
2. Flash status
3. Flash size (in Mbytes and Mbits)

5. Sample Code

The pseudo codes are provided here to showcase the procedure of interacting the EVE connected flash.

Flash Image Generator | Flash Programmer | Flash Diagnostic | Sample Code | Blob Release

Read Flash | Write Flash | Programming Flash | Update Flash | Erase Flash | Switch State | Install Blob | Read Me | Copy Source

```
// The pseudo code to read flash content to a file

// By default, the flash device operates in basic mode,
// which is slow. However, after programming the blob,
// it can be switched to fast mode by issuing the
// cmd_flashfast command. Once the flash is in full mode,
// the speed will be significantly improved.

// Assume the flash is in detach mode and now attach it
cmd_flashattach();
// Switch the flash into FAST mode
cmd_flashfast();
// Now check if the flash is in FAST mode
while (FLASH_STATUS_FULL != rd8(REG_FLASH_STATUS));

// Get flash size
flash_size = rd32(REG_FLASH_SIZE);
// Open file to write
FILE * fp = fopen(file_name, "wb+");
// Read flash content
loop while flash_size > 0
    cmd_flashread(buffer, src_flash, buffer_size);

    // Write buffer to file
    fwrite(buffer, 1, buffer_size, fp);

    // Update flash address and flash size
    src_flash += buffer_size;
    flash_size -= buffer_size;
```

Figure 64 - Flash Sample Code

Copy Source: Copy pseudo code in the current tab to the clipboard.

6. Blob Release

This tab displays the release notes for the blob files of BT815/6 and BT817/7A/8.

Flash Image Generator | Flash Programmer | Flash Diagnostic | Sample Code | Blob Release

```
BT815/6 BLOB Version 7
======================

Release note:
1) Fix multiple switching between cmd_flashdetach, cmd_flashattach and cmd_flashfast not working.


BT815/6 BLOB Version 6
======================

Release note:
1) Include patches for Display List instruction overfetch.
   For details, refer to https://brtchip.com/wp-content/uploads/2024/04/BRT_TN_005_BT81x_Errata_Technical_Note.pdf

2) This release contains three different BLOB:
   - default-fl.blob
   - cypress-fl.blob
   - macronix-fl.blob

2.1) default-fl.blob
     Always select this blob as default blob except for RDID indicate in 2.2 and 2.3.
     This blob includes coprocessor patches and support the list of flash chips as below:

     Cypress:
       NOR flash Series  |    Part Number
     ------------------------------------------
           S25FL-K       |  S25FL116K0XMFB040
           S25FL-K       |  S25FL132K0XMFA010
           S25FL-K       |  S25FL164K0XMFI011
           S25FS-S       |  S25FS128SAGMFI101

     GigaDevice:
```

Figure 65 - BLOB Release

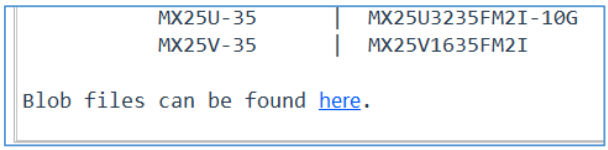


Figure 66 - Link to BLOB Folder

At the end of the text browser, there is a hyperlink to the Blob folder. Clicking on it will open that folder in File Explorer. This release contains three different BLOBs:

- default-fl.blob
- cypress-fl.blob
- macronix-fl.blob

default-fl.blob

Always select this blob as the default blob, except for RDID (Read Identification) indicated in Cypress and Macronix. This blob includes coprocessor patches and supports the following list of flash chips:

Cypress:

NOR flash Series	Part Number
S25FL-K	S25FL116K0XMFB040
S25FL-K	S25FL132K0XMFA010
S25FL-K	S25FL164K0XMFI011
S25FS-S	S25FS128SAGMFI101

GigaDevice:

NOR flash Series	Part Number
GD25Q-D	GD25Q256DYIGR
GD25Q-C	GD25Q127CSIGR
GD25Q-C	GD25Q64CSIGR
GD25Q-C	GD25Q32CSIGR
GD25Q-C	GD25Q64CSIG

ISSI:

NOR flash Series	Part Number
IS25LP-M	IS25LP512M-RMLE
IS25WP	IS25WP128-JBLE
IS25LP-A	IS25LP064A-JBLE
IS25WP-A	IS25WP064A-JBLE
IS25LP-D	IS25LP032D-JBLE
IS25LP-D	IS25LP016D-JBLE
IS25LP-A	IS25LP128-JBLE
IS25LP-D	IS25LP256D-JLLE

Micron:

NOR flash Series	Part Number
MT25QL-A	MT25QL128ABA8ESF-0AAT
MT25QL-A	MT25QL512ABB8ESF-0AAT
MT25QL-A	MT25QL256ABA1EW9-0SIT
MT25QL-A	MT25QL128ABA1EW7-0SIT
MT25QL-B	MT25QL01GBBB8ESF-0SIT

Winbond:

NOR flash Series	Part Number
W25Q-JV	W25Q128JVSIM
W25Q-NW	W25Q512NWEIM
W25Q-JV	W25Q16JVSSIQ
W25Q-FW	W25Q16FWSIQ
W25Q-JV	W25Q32JVSSIQ
W25Q-JV	W25Q64JVSSIQ
W25Q-FV	W25Q128FVSIG
W25Q-FV	W25Q64FVSSIG
W25M-JV	W25M512JVEIQ
W25Q-JV	W25Q256JVEIQ
W25Q-JV	W25Q128JVSIQ
W25Q-NW	W25Q01NWZEIM
W25Q-JV	W25Q128JVSIM
W25Q-NW	W25Q512NWEIM

ESMT/EON:

NOR flash Series	Part Number
EN35QX-A	EN35QX512A

cypress-fl.blob

This blob supports Cypress flash with RDID 0160h. It includes coprocessor patches and supports the list of flash chips as below:

NOR flash Series	Part Number
S25FL-L	S25FL128LAGMFA010
S25FL-L	S25FL128LAGMFM010
S25FL-L	S25FL256LAGNFV010
S25FL-L	S25FL256LAGNFM010
S25FL-L	S25FL064LABMFA001

macronix-fl.blob

This blob supports Macronix flash with RDID C220h, C223h and C225h. It includes coprocessor patches and supports the list of flash chips as below:

NOR flash Series	Part Number
MX25L-35	MX25L25635FZ2I-10G
MX25L-35	MX25L12835FM2I-10G
MX25L-45	MX25L12845EZNI-10G
MX25L-45	MX25L6445EM2I-10G
KH25L-35	KH25L12835FM2I-12G
MX25U-45	MX25U51245GZ4I00
MX25U-35	MX25U12835FZ2I-10G
MX25U-35	MX25U6435FM2I-10G
MX25U-35	MX25U3235FM2I-10G
MX25V-35	MX25V1635FM2I

G. Custom Touch Firmware Compiler

Compile a small program in a tiny C-like language and produces a loadable firmware image for touch functionality.

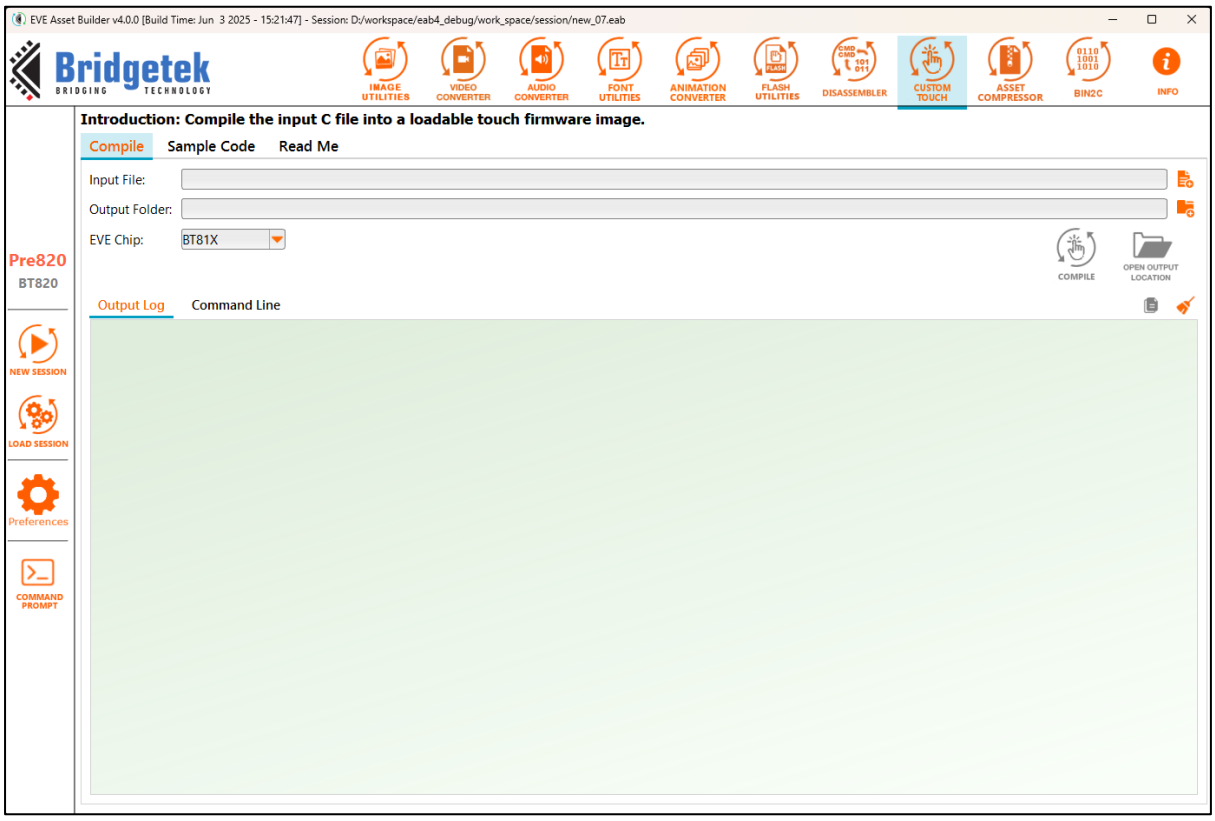


Figure 67 - Custom Touch Firmware Complier

Input File: Specify a small C-like language file for compilation.

EVE Chip: Select BT81X or FT81X/BT88X for which the touch firmware is intended.

Sample Code Tab: Provide source code for some touch controller ICs.

Read Me Tab: Introduction to the Custom Touch Firmware compiler.

Note: There are underlying command-line tools to compile the input file, which is called *jtc.exe* and *jtc_FT81X_BT88X.exe*, located at *<EAB Installation folder>/tools.*

Usage: jtc.exe <input.c>

It produces two output files: <input>.load.bin and <input>.load.h. Both files serve the same purpose and functionality, with the only difference being that .bin is in binary format, while .h is in text format.

Pseudo-code on how to use custom firmware

```
/*
Assume that EVE boot-up sequence is properly done before this routine.
The custom touch firmware is a piece of code that can be executed by Eve
coprocessor, so the routine will do:

1) Read the firmware into local memory or use the .load.h file
2) Write the firmware into RAM_CMD
3) Update the register REG_CMD_WRITE to start execution
4) Wait till the execution is done (REG_CMD_READ==REG_CMD_WRITE)
*/

// Read the custom touch firmware 'output.load.bin' into local memory
custom_touch_content = read("output.load.bin", "b");

// Write the custom touch firmware into RAM_CMD
EVE_Hal_wrMem(RAM_CMD, custom_touch_content,
sizeof(custom_touch_content));

// Update the write pointer register to start the execution
EVE_Hal_wr32(REG_CMD_WRITE, sizeof(custom_touch_content));

// Wait till Eve completes all the commands
while (EVE_Hal_rd32(REG_CMD_READ) == EVE_Hal_rd32(REG_CMD_WRITE));

// DONE
```

Contact Information

Refer to https://brtchip.com/contact-us/ for contact information.

Distributor and Sales Representatives

Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A – List of Figures

Appendix B – List of Tables

Appendix C – Revision History

Document Title	:	BRT_AN_098 EVE Asset Builder 4.0 User Guide
Document Reference No.	:	BRT_000473
Clearance No.	:	BRT#235
Product Page	:	https://brtchip.com/toolchains/
Document Feedback	:	Send Feedback

Revision	Changes	Date
Version 1.0	Initial Release for 4.0.0	09-07-2025