TivaWare™ Host Tools

USER'S GUIDE

SW-TM4C-TOOLS-UG-1.1

Copyright

Texas Instruments
108 Wild Basin, Suite 350
Austin, TX 78746
www.ti.com/tiva-c

Revision Information

This is version 1.1 of this document, last updated on July 02, 2013.

Table of Contents

1 Introduction

The Texas Instruments® TivaWare™ Host Tools are tools that run on the development system, not on the embedded target. They are provided to assist in the development of firmware for the Tiva™ family of ARM® Cortex™-M based microcontrollers.

These tools reside in the `tools` subdirectory of the firmware development package source distribution.

2 Binary Packaging Utility

Usage:

```
binpack [OPTION]...
```

Description:

Prepares binary images for download to a system using a CRC-checking version of the boot loader. This tool calculates and embeds length and CRC32 into the image informatoin header placed at the top of the interrupt vector table. This information is used by the boot loader on each reset to ensure that the image has not been corrupted.

Applications which intend to use binpack and the CRC-enabled boot loader must be built to include an 8 word image information header appended to the end of the interrupt vector table typically found in the startup C or assembler file of the project. The first two words of this structure must be initialized with values 0xFF01FF02 and 0xFF03FF04 respectively and the following 6 words must be unused. Binpack will write the image length, in bytes, into the third word of the image information header, immediately following the two marker words, and the CRC32 for the image into the fourth word. The CRC32 value uses the standard ANSI X 3.66 polynomial of 0x04C11DB7. It is calculated after the length field has been written into the header and spans all bytes of the image excluding the 4 bytes which will be occupied by the CRC value itself.

The utility will also, optionally, add an 8-byte header to the output file containing information on the address at which the binary image is to be flashed. This is used by some download tools but note that LMFlash does not require this header and images intended for use with LMFlash must not include it. To add this header, invoke binpack with the “-d” command line option. The header structure used is exactly the same as added by the dfuwrap utility. The first half word contains the fixed value 0x0001, the second half word contains the image flash address expressed as a number of kilobytes (address / 1024), and the remaining 4 bytes are written with the size of the binary image excluding the header. In all cases, multi-byte values are stored in little-endian format with the least significant byte first.

The source code for this utility is contained in `tools/binpack`, with a pre-built binary contained in `tools/bin`.

Arguments:

-a ADDR specifies the address of the binary.

-h displays usage information.

-i FILE specifies the name of the input binary file.

-o FILE specifies the name of the output file. If not specified, the default of firmware.bin will be used.

-q specifies that only error information should be output.

-d specifies that the header structure should be appended to the beginning of the output file.

-x specifies that the output file should be overwritten without prompting.

Example:

The following example packages program.bin making it suitable for use with LMFlash and the CRC-checking boot loader. The image header is skipped and only the CRC and length values are embedded within the image.

```
binpack -i program.bin -o program_out.bin
```

This second example generates another output file for program.bin, this time including the 8

byte header which contains both the image length and the address at which the image is to be flashed, 0x1800 in this case.

```
binpack -i program.bin -o program_out_with_header.bin -a 0x1800 -d
```

3 Data Logger

Usage:

`logger`

Description:

Provides a Windows front-end for the ek-lm4f232 data logger application (qs-logger) and allows captured data to be logged to file and displayed on the screen in several strip charts.

The qs-logger application provides a virtual COM port via USB and the logger application opens this and parses data received from the board as it is captured. All control, other than setting up the file and deciding which captured channels' data to display, is performed using the menus provided by the qs-logger application on the ek-lm4f232 board.

The device driver required to support the qs-logger application's virtual COM port on Windows can be found in `windows_drivers`.

The source code for this utility is contained in `tools/logger`, with a pre-built binary contained in `tools/bin`.

4 FreeType Rasterizer

Usage:

```
ftrasterize [OPTION]... [INPUT FILE]
```

Description:

Uses the FreeType font rendering package to convert a font into the format that is recognized by the graphics library. Any font that is recognized by FreeType can be used, which includes TrueType®, OpenType®, PostScript® Type 1, and Windows® FNT fonts. A complete list of supported font formats can be found on the FreeType web site at `http://www.freetype.org`.

FreeType is used to render the glyphs of a font at a specific size in monochrome, using the result as the bitmap images for the font. These bitmaps are compressed and the results are written as a C source file that provides a tFont structure describing the font.

The source code for this utility is contained in `tools/ftrasterize`, with a pre-built binary contained in `tools/bin`.

Arguments:

-a specifies the index of the font character map to use in the conversion. If absent, Unicode is assumed when “-r” or “-u” is present. Without either of these switches, the Adobe Custom character map is used if such a map exists in the font, otherwise Unicode is used. This ensures backwards compatibility. To determine which character maps a font supports, call ftrasterize with the “-d” option to show font information.

-b specifies that this is a bold font. This does not affect the rendering of the font, it only changes the name of the file and the name of the font structure that are produced.

-c FILENAME specifies the name of a file containing a list of character codes whose glyphs should be encoded into the output font. Each line of the file contains either a single decimal or hex character code in the chosen codepage (Unicode unless “-a” is provided), or two character codes separated by a comma and a space to indicate all characters in the inclusive range. Additionally, if the first non-comment line of the file is “REMAP”, the output font is generated to use a custom codepage with character codes starting at 1 and incrementing with every character in the character map file. This switch is only valid with “-r” and overrides “-p” and “-e” which are ignored if present.

-d displays details about the first font whose name is supplied at the end of the command line. When “-d” is used, all other switches are ignored. When used without “-v”, font header information and properties are shown along with the total number if characters encoded by the font and the number of contiguous blocks these characters are found in. With “-v”, detailed information on the character blocks is also displayed.

-f FILENAME specifies the base name for this font, which is used as a base for the output file names and the name of the font structure. The default value is “font” if not specified.

-h shows command line help information.

-i specifies that this is an italic font. This does not affect the rendering of the font, it only changes the name of the file and the name of the font structure that are produced.

-m specifies that this is a monospaced font. This causes the glyphs to be horizontally centered in a box whose width is the width of the widest glyph. For best visual results, this option should only be used for font faces that are designed to be monospaced (such as Computer Modern TeleType).

-s SIZE specifies the size of this font, in points. The default value is 20 if not specified. If the size provided starts with "F", it is assumed that the following number is an index into the font’s fixed size table. For example “-s F3” would select the fourth fixed size offered by the font. To determine whether a given font supports a fixed size table, use ftrasterize with the “-d” switch.

-p NUM specifies the index of the first character in the font that is to be encoded. If the value is not provided, it defaults to 32 which is typically the space character. This switch is ignored if "-c" is provided.

-e NUM specifies the index of the last character in the font that is to be encoded. If the value is not provided, it defaults to 126 which, in ISO8859-1 is tilde. This switch is ignored if "-c" is provided.

-v specifies that verbose output should be generated.

-w NUM encodes the specified character index as a space regardless of the character which may be present in the font at that location. This is helpful in allowing a space to be included in a font which only encodes a subset of the characters which would not normally include the space character (for example, numeric digits only). If absent, this value defaults to 32, ensuring that character 32 is always the space. Ignored if "-r" is specified.

-n overrides -w and causes no character to be encoded as a space unless the source font already contains a space.

-u causes ftrasterize to use Unicode character mapping when extracting glyphs from the source font. If absent, the Adobe Custom character map is used if it exists or Unicode otherwise.

-r specifies that the output should be a relocatable, wide character set font described using the tFontWide structure. Such fonts are suitable for encoding characters sets described using Unicode or when multiple contiguous blocks of characters are to be stored in a single font file. This switch may be used in conjunction with "-y" to create a binary font file suitable for use from a file system.

-y writes the output in binary rather than text format. This switch is only valid if used in conjunction with "-r" and is ignored otherwise. Fonts generated in binary format may be accessed by the graphics library from a file system or other indirect storage assuming that simple wrapper software is provided.

-o NUM specifies the codepoint for the first character in the source font which is to be translated to a new position in the output font. If this switch is not provided, no remapping takes place. If specified, this switch must be used in conjunction with -t which specifies where remapped characters are placed in the output font. Ignored if "-r" is specified.

-t NUM specifies the output font character index for the first character remapped from a higher codepoint in the source font. This should be used in conjunction with "-o". The default value is 0. Ignored if "-r" is specified.

-z NUM specifies the codepage identifier for the output font. This switch is only valid if used with "-r" and is primarily intended for use when performing codepage remapping and custom string tables. The number provided when performing remapping must be in the region between CODEPAGE_CUSTOM_BASE (0x8000) and 0xFFFF.

INPUT FILE specifies the name of the input font file. When used with "-r", up to four font filenames may be provided in order of priority. Characters missing from the first font are searched for in the remaining fonts. This allows the output font to contain characters from multiple different fonts and is helpful when generating multi-language string tables containing different alphabets which do not all exist in a single input font file.

Examples:

The following example produces a 24-point font called test and containing ASCII characters in the range 0x20 to 0x7F from test.ttf:

```
ftrasterize -f test -s 24 test.ttf
```

The result will be written to `fonttest24.c`, and will contain a structure called `g_sFontTest24` that describes the font.

The following would render a Computer Modern small-caps font at 44 points and generate an output font containing only characters 47 through 58 (the numeric digits). Additionally, the first character in the encoded font (which is displayed if an attempt is made to render a character which is not included in the font) is forced to be a space:

```
ftrasterize -f cmscdigits -s 44 -w 47 -p 47 -e 58 cmcsc10.pfb
```

The output will be written to `fontcmscdigits44.c` and contain a definition for `g_sFontCmscdigits44` that describes the font.

To generate some ISO8859 variant fonts, a block of characters from a source Unicode font must be moved downwards into the [0-255] codepoint range of the output font. This can be achieved by making use of the -t and -o switches. For example, the following will generate a font containing characters 32 to 255 of the ISO8859-5 character mapping. This contains the basic western European alphanumerics and the Cyrillic alphabet. The Cyrillic characters are found starting at Unicode character 1024 (0x400) but these must be placed starting at ISO8859-5 character number 160 (0xA0) so we encode characters 160 and above in the output from the Unicode block starting at 1024 to translate the Cyrillic glyphs into the correct position in the output:

```
ftrasterize -f cyrillic -s 18 -p 32 -e 255 -t 160 -o 1024 -u unicode.ttf
```

When encoding wide character sets for multiple alphabets (Roman, Arabic, Cyrillic, Hebrew, etc.) or to deal with ideograph-based writing systems (Hangul, Traditional or Simplified Chinese, Hiragana, Katakana, etc.), a character block map file is required to define which sections of the source font's codespace to encode into the destination font. The following example character map could be used to encode a font containing ASCII plus the Japanese Katakana alphabets:

```
##############################################################################
#
# katakana.txt - Unicode block definitions for ASCII and Katakana.
#
##############################################################################

# ASCII characters
0x20, 0x7E

# Katakana alphabet
0x30A0, 0x30FF
0x31F0, 0x32FF
0xFF00, 0xFFEF
```

Assuming the font "unicode.ttf" contains these glyphs and that it includes fixed size character renderings, the fifth of which uses an 8x12 character cell size, the following ftrasterize command line could then be used to generate a binary font file called fontkatakana8x12.bin containing this subset of characters:

```
ftrasterize -f katakana -s F4 -c katakana.txt -y -r -u unicode.ttf
```

In this case, the output file will be fontkatakana8x12.bin and it will contain a binary version of the font suitable for use from external memory (SDCard, a file system, serial flash memory, etc.) via a tFontWrapper and a suitable font wrapper module.

5 GIMP Script For Texas Instruments TivaWare Graphics Library Button

Description:

This is a script-fu plugin for GIMP (`http://www.gimp.org`) that produces push button images that can be used by the push button widget. When installed into `${HOME}/.gimp-2.4/scripts`, this will be available under Xtns->Buttons->LMI Button. When run, a dialog will be displayed allowing the width and height of the button, the radius of the corners, the thickness of the 3D effect, the color of the button, and the pressed state of the button to be selected. Once the desired configuration is selected, pressing OK will create the push button image in a new GIMP image. The image should be saved as a raw PPM file so that it can be converted to a C array by `pnmtoc`.

This script is provided as a convenience to easily produce a particular push button appearance; the push button images can be of any desired appearance.

This script is located in `tools/lmi-button/lmi-button.scm`.

6 NetPNM Converter

Usage:

```
pnmtoc [OPTION]... [INPUT FILE]
```

Description:

Converts a NetPBM image file into the format that is recognized by the TivaWare Graphics Library. The input image must be in the raw PPM format (in other words, with the `P4`, `P5` or `P6` tags). The NetPBM image format can be produced using GIMP, NetPBM (`http://netpbm.sourceforge.net`), ImageMagick (`http://www.imagemagick.org`), or numerous other open source and proprietary image manipulation packages.

The resulting C image array definition is written to standard output; this follows the convention of the NetPBM toolkit after which the application was modeled (both in behavior and naming). The output should be redirected into a file so that it can then be used by the application.

To take a JPEG and convert it for use by the graphics library (using GIMP; a similar technique would be used in other graphics programs):

1. Load the file (File->Open).
2. Convert the image to indexed mode (Image->Mode->Indexed). Select “Generate optimum palette” and select either 2, 16, or 256 as the maximum number of colors (for a 1 BPP, 4 BPP, or 8 BPP image respectively). If the image is already in indexed mode, it can be converted to RGB mode (Image->Mode->RGB) and then back to indexed mode.
3. Save the file as a PNM image (File->Save As). Select raw format when prompted.
4. Use `pnmtoc` to convert the PNM image into a C array.

This sequence will be the same for any source image type (GIF, BMP, TIFF, and so on); once loaded into GIMP, it will treat all image types equally. For some source images, such as a GIF which is naturally an indexed format with 256 colors, the second step could be skipped if an 8 BPP image is desired in the application.

The source code for this utility is contained in `tools/pnmtoc`, with a pre-built binary contained in `tools/bin`.

Arguments:

-c specifies that the image should be compressed. Compression is bypassed if it would result in a larger C array.

Example:

The following will produce a compressed image in foo.c from foo.ppm:

```
pnmtoc -c foo.ppm > foo.c
```

This will result in an array called `g_pucImage` that contains the image data from `foo.ppm`.

7 Serial Flash Downloader

Usage:

```
sflash [OPTION]... [INPUT FILE]
```

Description:

Downloads a firmware image to a Tiva board using a UART connection to the TivaWare Serial Flash Loader or the TivaWare Boot Loader. This has the same capabilities as the serial download portion of the LM Flash Programmer tool.

The source code for this utility is contained in `tools/sflash`, with a pre-built binary contained in `tools/bin`.

Arguments:

-b BAUD specifies the baud rate. If not specified, the default of 115,200 will be used.

-c PORT specifies the COM port. If not specified, the default of COM1 will be used.

-d disables auto-baud.

-h displays usage information.

-l FILENAME specifies the name of the boot loader image file.

-p ADDR specifies the address at which to program the firmware. If not specified, the default of 0 will be used.

-r ADDR specifies the address at which to start processor execution after the firmware has been downloaded. If not specified, the processor will be reset after the firmware has been downloaded.

-s SIZE specifies the size of the data packets used to download the firmware date. This must be a multiple of four between 8 and 252, inclusive. If using the Serial Flash Loader, the maximum value that can be used is 76. If using the Boot Loader, the maximum value that can be used is dependent upon the configuration of the Boot Loader. If not specified, the default of 8 will be used.

INPUT FILE specifies the name of the firmware image file.

Example:

The following will download a firmware image to the board over COM2 without auto-baud support:

```
sflash -c 2 -d image.bin
```

8 String Table Generator

Usage:

```
mkstringtable [INPUT FILE] [OUTPUT FILE]
```

Description:

Converts a comma separated file (.csv) to a table of strings that can be used by the TivaWare Graphics Library. The source .csv file has a simple fixed format that supports multiple strings in multiple languages. A .c and .h file will be created that can be compiled in with an application and used with the graphics library's string table handling functions. If encoding purely ASCII strings, the strings will also be compressed in order to reduce the space required to store them. If the CSV file contains strings encoded in other codepages, for example UTF8, the "-s" command line option must be used to specify the encoding used for the string and "-u" must also be used to ensure that the strings are stored correctly.

The format of the input .csv file is simple and easily edited in any plain text editor or a spreadsheet editor capable of reading and editing a .csv file. The .csv file format has a header row where the first entry in the row can be any string as it is ignored. The remaining entries in the row must be one of the GrLang* language definitions defined by the graphics library in `grlib.h` or they must have a `#define` definition that is valid for the application as this text is used directly in the C output file that is produced. Adding additional languages only requires that the value is unique in the table and that the name used is defined by the application.

The strings are specified one per line in the .csv file. The first entry in any line is the value that is used as the actual text for the definition for the given string. The remaining entries should be the strings for each language specified in the header. Single words with no special characters do not require quotations, however any strings with a "," character must be quoted as the "," character is the delimiter for each item in the line. If the string has a quote character "" it must be preceded by another quote character.

The following is an example .csv file containing string in English (US), German, Spanish (SP), and Italian:

```
LanguageIDs,GrLangEnUS,GrLangDE,GrLangEsSP,GrLangIt
STR_CONFIG,Configuration,Konfigurieren,Configuracion,Configurazione
STR_INTRO,Introduction,Einfuhrung,Introduccion,Introduzione
STR_QUOTE,Introduction in ""English"","Einfuhrung, in Deutsch",Prueba,Verifica
...
```

In this example, `STR_QUOTE` would result in the following strings in the various languages:

- GrLangEnUs – `Introduction in "English"`
- GrLangDE – `Einfuhrung, in Deutsch`
- GrLangEsSP – `Prueba`
- GrLangIt – `Verifica`

The resulting .c file contains the string table that must be included with the application that is using the string table and two helper structure definitions, one a tCodepointMap array containing a single entry that is suitable for use with GrCodepageMapTableSet() and the other a tGrLibDefaults structure which can be used with GrLibInit() to initialize the graphics library to use the correct codepage for the string table.

While the contents of this .c file are readable, the string table itself may be unintelligible due to the compression or remapping used on the strings themselves. The .h file that is created has the definition for the string table as well as an enumerated type `enum SCOMP_STR_INDEX`

that contains all of the string indexes that were present in the original .csv file and external definitions for the tCodepointMap and tGrLibDefaults structures defined in the .c file.

The code that uses the string table produced by this utility must refer to the strings by their identifier in the original .csv file. In the example above, this means that the value `STR_CONFIG` would refer to the "Configuration" string in English (GrLangEnUS) or "Konfigurieren" in German (GrLangDE).

This utility is contained in `tools/bin`.

Arguments:

-u indicates that the input .csv file contains strings encoded with UTF8 or some other non-ASCII codepage. If absent, mkstringtable assumes ASCII text and uses this knowledge to apply higher compression to the string table.

-c NUM specifies the custom codepage identifier to use when remapping the string table for use with a custom font. Applications using the string table must set this value as the text codepage in use via a call to GrStringCodepageSet() and must ensure that they include an entry in their codepage mapping table specifying this value as both the source and font codepage. A macro, GRLIB_CUSTOM_MAP_xxx, containing the required tCodePointMap structure is written to the output header file to make this easier. A structure, g_GrLibDefaultxxxx, is also exported and a pointer to this may be passed to GrLibInit() to allow widgets to make use of the string table's custom codepage. Valid values to pass as NUM are in the 0x8000 to 0xFFFF range set aside by the graphics library for application-specific or custom text codepages.

-r indicates that the output string table should be constructed for use with a custom font and codepage. Character values in the string are remapped into a custom codepage intended to minimize the size of both the string table and the custom font used to display its strings. If "-r" is specified, an additional .txt output file is generated containing information that may be passed to the ftrasterize tool to create a compatible custom font containing only the characters required by the string table contents.

-s STR specifies the codepage used in the input .csv file. If this switch is absent, ASCII is assumed. Valid values of STR are "ASCII", "utf8" and "iso8859-n" where "n" indicates the ISO8859 variant in use and can have values from 1 to 11 or 13 to 16. Care must be taken to ensure that the .csv file is correctly encoded and makes use of the encoding specified using "-s" since a mismatch will cause the output strings to be incorrect. Note, also, that support for UTF-8 and ISO8859 varies from text editor to text editor so it is important to ensure that your editor supports the desired codepage to prevent string corruption.

-t indicates that a character map file with a .txt extension should be generated in addition to the usual .c and .h output files. This output file may be passed to ftrasterize to generate a custom font containing only the glyphs required to display the strings in the table. Unlike the character map file generated when using "-r", the version generated by "-t" will not remap the codepage of the strings in the table. This has the advantage of leaving them readable in a debugger (which typically understands ASCII and common codepages) but will generate a font that is rather larger than the font that would have been generated using a remapped codepage due to the additional overhead of encoding many small blocks of discontiguous characters.

-i indicates that the string table should be written into a binary file rather than included within the .c output file. The binary output name will use the same filename as the .c and .h files but will have a .bin file extension. When a binary file is generated, the .c file will contain all structures usually generated but will not contain the array containing the string table data. A binary string table is position-independent and may be stored anywhere in memory on condition that it is aligned on a 32-bit word boundary. The binary string table is used in exactly the same way as the linked .c version except that the parameter passed

to GrStringTableSet() must be the address at which the application located the first word of the string table data rather than a label exported from the string table's .c file.

-l FILENAME specifies an additional header file which is parsed for additional language IDs. Although the tool recognizes all the GrLangXxx labels found in grlib.h, applications may define their own language IDs in a header file and pass this to mkstringtable. This header will be included in the output C files allowing custom language labels to be used without the need to edit the mkstringtable output files later. Header files passed with the "-l" parameter must contain only comments, blank lines and definitions of the form "#define LangIDLabel 0x0000" with the label's value given in hex and in the range 0x0001 to 0xFFFF.

INPUT FILE specifies the input .csv file to use to create a string table.

OUTPUT FILE specifies the root name of the output files as `<OUTPUT FILE>.c` and `<OUTPUT FILE>.h`. The value is also used in the naming of the string table variable.

Example:

The following will create a string table in `str.c`, with prototypes in `str.h`, based on the ASCII input file `str.csv`:

```
mkstringtable str.csv str
```

In the produced `str.c`, there will be a string table in `g_pucTablestr`.

The following will create a string table in `widestr.c`, with prototypes in `widestr.h`, based on the UTF8 input file `widestr.csv`. This form of the call should be used to encode string tables containing accented characters or non-Western character sets:

```
mkstringtable –u widestr.csv widestr
```

In the produced `widestr.c`, there will be a string table in `g_pucTablewidestr`.

9 USB Bulk Data Transfer Example

Description:

usb_bulk_example is a Windows command line application which communicates with the TivaWare usb_dev_bulk example. The application finds the Tiva device on the USB bus then, if found, prompts the user to enter strings which are sent to the application running on the Tiva board. This application then inverts the case of the alphabetic characters in the string and returns the data back to the USB host where it is displayed.

The source code for this application is contained in `tools/usb_bulk_example`. The binary is installed as part of the "Windows-side examples for USB kits" package (SW-USB-win) shipped on the release CD and downloadable from http://www.ti.com/tivaware. A Microsoft Visual Studio project file is provided to allow the application to be built.

10 USB DFU Library

Description:

LMDFU is a Windows dynamic link library offering a high level interface to the USB Device Firmware Upgrade functionality provided by the TivaWare USB boot loader (boot_usb). This DLL is used by the dfuprog utility and also by the LMFlash application to allow download and upload of application images to or from a Tiva-based board via USB.

The source code for this DLL is contained in `tools/lmdfu`. The DLL binary is installed as part of the "TivaWare Embedded USB drivers" package (SW-USB-windrivers) shipped on the release CD and downloadable from http://www.ti.com/tivaware. A Microsoft Visual Studio 2008 project file is provided to allow the application to be built.

11 USB DFU Programmer

Usage:

```
dfuprog [OPTION]...
```

Description:

Downloads images to a Texas Instruments Tiva microcontroller running the USB Device Firmware Upgrade boot loader. Additionally, this utility may be used to read back the existing application image or a subsection of flash and store it either as raw binary data or as a DFU-downloadable image file.

The source code for this utility is contained in `tools/dfuprog`. The binary for this utility is installed as part of the "Windows-side examples for USB kits" package (SW-USB-win) shipped on the release CD and downloadable from http://www.luminarymicro.com/products/software_updates.html. A Microsoft Visual Studio project file is provided to allow the application to be built.

Arguments:

-e specifies the address of the binary.

-u specifies that an image is to be uploaded from the board into the target file. If absent, the file will be downloaded to the board.

-c specifies that a section of flash memory is to be cleared. The address and size of the block may be specified using the -a and -l parameters. If these are absent, the entire writable area of flash is erased.

-f FILE specifies the name of the file to download or, if -u is given, to upload.

-b specifies that an uploaded file is to be stored as raw binary data without the DFU file wrapper. This option is only valid if used alongside -u.

-d specifies that the VID and PID in the DFU file wrapper should be ignored for a download operation.

-s specifies that image verification should be skipped following a download operation.

-a ADDR specifies the address at which the binary file will be downloaded or from which an uploaded file will be read. If a download operation is taking place and the source file provided is DFU-wrapped, this parameter will be ignored.

-l SIZE specifies the number of bytes to be uploaded when used in conjunction with -i or the number of bytes of flash to erase if used in conjunction with -c.

-i NUM specifies the zero-based index of the USB DFU device to access if more than one is currently attached to the system. If absent, the first device found is used.

-x specifies that destination file for an upload operation should be overwritten without prompting if it already exists.

-w specifies that the utility should wait for the user to press a key before it exits.

-v displays verbose output during the requested operation.

-h displays this help information.

-? displays this help information.

Example:

The following example writes binary file program.bin to the device flash memory at address 0x1800:

```
dfuprog -f program.bin -a 0x1800
```

The following example writes DFU-wrapped file program.dfu to the flash memory of the second connected USB DFU device at the address found in the DFU file prefix:

```
dfuprog -i 1 -f program.dfu
```

The following example uploads (reads) the current application image into a DFU-formatted file appimage.dfu:

```
dfuprog -u -f appimage.dfu
```

12 USB DFU Wrapper

Usage:

```
dfuwrap [OPTION]...
```

Description:

Prepares binary images for download to a particular position in device flash via the USB device firmware upgrade protocol. A Tiva-specific prefix and a DFU standard suffix are added to the binary.

The source code for this utility is contained in `tools/dfuwrap`, with a pre-built binary contained in `tools/bin`.

Arguments:

-a ADDR specifies the address of the binary.

-c specifies that the validity of the DFU wrapper on the input file should be checked.

-d ID specifies the USB device ID to place into the DFU wrapper. If not specified, the default of 0x0000 will be used.

-e enables verbose output.

-f specifies that a DFU wrapper should be added to the file even if one already exists.

-h displays usage information.

-i FILE specifies the name of the input file.

-o FILE specifies the name of the output file. If not specified, the default of image.dfu will be used.

-p ID specifies the USB product ID to place into the DFU wrapper. If not specified, the default of 0x00ff will be used.

-q specifies that only error information should be output.

-r specifies that the DFU header should be removed from the input file.

-v ID specifies the USB vendor ID to place into the DFU wrapper. if not specified, the default of 0x1cbe will be used.

-x specifies that the output file should be overwritten without prompting.

Example:

The following example adds a DFU wrapper which will cause the image to be programmed to address 0x1800:

```
dfuwrap -i program.bin -o program.dfu -a 0x1800
```

13 USB Dynamic Link Library

Description:

LMUSBDLL is a simple Windows dynamic link library offering low level packet read and write functions for some USB-connected TivaWare example applications. The DLL is written above the Microsoft WinUSB interface and is intended solely to ensure that various Windows-side example applications can be built without having to use WinUSB header files. These header files are not included in the Visual Studio tools and are only shipped in the Windows Device Driver Kit (DDK). By providing this simple mapping DLL which links to WinUSB, the user avoids the need for a multi-gigabyte download to build the examples.

The source code for this DLL is contained in `tools/lmusbdll`. The DLL binary is installed as part of the “Tiva Embedded USB drivers” package (SW-USB-windrivers) shipped on the release CD and downloadable from http://www.ti.com/tivaware. A Microsoft Visual Studio 2008 project file is provided to allow the DLL to be built on a PC which has the Windows Device Driver Kit installed.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265