UG107: ISA3 Utilities Guide

Developers using Silicon Labs' network stack software, such as EmberZNet PRO or Silicon Labs Thread, and development tools on the EM3x platforms must also use one of the standalone ISA3 utilities installed with the rest of the software. These utilities are software tools that perform tasks not integrated into the normal Integrated Development Environment (IDE). This document describes how and when to use these utilities.

Note: This document applies only to the Em3x platforms. See *UG162: Simplicity Commander Reference Guide* for similar functions for the EFR32xG platforms.

KEY POINTS

- em3xx_load
- em3xx_convert
- em3xx_buildimage

1. Introduction

This document describes how and when to use the standalone ISA3 utilities. Silicon Labs recommends that you review this document to familiarize yourself with each utility and its intended use. You can refer to specific sections of this document to access operational information as needed. This document is intended for software and hardware engineers who are responsible for building an embedded mesh networking solution using the EM35x SoC. It assumes that the reader has a solid understanding of embedded systems design.

Both the em3xx_convert and em3xx_buildimage utilities may be used from the command line with the Wireless Gecko (EFR32) portfolio as well. Indeed, em3xx_convert must be present in your development environment in order to generate .EBL files for the EFR32. However, em3xx_load cannot be used to program the EFR32.

The ISA3 utilities may vary with the specific version of your installed product. If you do not have one of these utilities, or require assistance with the procedures described in this document, contact Customer Support at www.silabs.com/support.

The utilities included in this document are listed in the following table.

Table 1.1. Utilities Described in this Document

Utility	Description
em3xx_load	Programming utilities for EM35x SoC.
em3xx_convert	Converts files from one format to another.
em3xx_buildimage	Builds and modifies a single flash image from multiple sources.

1.1 Tool Overview

The Ember networking device family requires various kinds of device programming. Gang programmers are frequently used in a manufacturing environment, while single device programmers are used most often in a development or pilot production environment.

The ISA3 utilities have been created to address this and related needs and are used with the Ember Debug Adapter (ISA3). Each tool in the ISA3 family has a very specific function, described in detail in the following chapters.

1.2 File Format Overview

The ISA3 utilities work with different file formats: .s37, .ebl, and .hex. Each file format serves a slightly different purpose. The ISA3 utilities contains a conversion tool, em3xx_convert, which converts applications from .s37 file format to .ebl. The file formats are summarized below.

1.2.1 Motorola S-record file format

Silicon Labs uses the IAR Embedded Workbench as its IDE. The IDE produces Motorola S-record files, s37 specifically, as its output. An application image in s37 format can be loaded into a target EM35x using the em3xx_load utility. The s37 format can represent any combination of any byte of flash in the EM35x. The em3xx_convert utility can be used to convert the s37 format to ebl format for use with Silicon Labs bootloaders. The em3xx_buildimage utility can be used to read multiple s37 files, and ebl and hex files; output an s37 file for combining multiple files into a single file; and modify individual bytes of a file.

1.2.2 Ember Bootloader (ebl) file format

The Ember Bootloader (ebl) file format is generated by the em3xx_convert utility. This file format can only represent an application image.

The ebl file format is designed to be an efficient and fault-tolerant image format for use with the Ember bootloader to upgrade an application without the need for special programming devices. The bootloader can receive an ebl file either Over The Air (OTA) or via a serial cable and reprogram the flash in place.

Although the ebl file format is intended for use with a bootloader, the em3xx_load utility is also capable of directly programming an ebl image. This file format is generally used in later stage development, and for upgrading manufactured devices in the field. The standalone bootloader should never be loaded onto the device as an ebl image. Use the s37 file format when loading the bootloader itself.

The em3xx_buildimage utility can accept ebl files as inputs but cannot generate ebl files as an output.

1.2.3 Intel HEX-32 file format

Production programming uses the standard Intel HEX-32 file format (see https://en.wikipedia.org/wiki/Intel_HEX for more information). The normal development process for EM35x chips involves creating and programming images using the s37 and ebl file formats. The s37 and ebl files are intended to hold applications, bootloaders, manufacturing data, and other information to be programmed during development. The s37 and ebl files, though, are not intended to hold a single image for an entire chip. For example, it is often the case that there is an s37 file for the bootloader, an s37 file for the application, and an s37 file for manufacturing data. Because production programming is primarily about installing a single, complete image with all the necessary code and information, the file format used is Intel HEX-32 format. While s37 and hex files are functionally the same—they simply define addresses and the data to be placed at those addresses—Silicon Labs has adopted the conceptual distinction that a single hex file contains a single, complete image often derived from multiple s37 files. The em3xx_buildimage utility can be used to read multiple hex files, and ebl and s37 files; output a hex file for combining multiple files into a single file; and modify individual bytes of a file.

Note: Both the em3xx_load and em3xx_buildimage utilities are capable of working identically with s37 and hex files. All functionality that can be performed with s37 files can be performed with hex files. em3xx_convert, though, is not capable of working with hex files. Ultimately, with respect to production programming, em3xx_load allows the developer to load a variety of sources onto a physical chip while em3xx_buildimage allows the developer to merge a variety of sources into a final image file and modify individual bytes in that image if necessary.

The following table summarizes the inputs and outputs for the different file formats.

Table 1.2. File Format Summary

Utility	Inputs				Outputs			
	ebl	**s37**	**hex**	**chip**	**ebl**	**s37**	**hex**	**chip**
em3xx_load	X	X	X	X		X	X	X
em3xx_convert		X			X			
em3xx_buildimage	X	X	X			X	X	

2. em3xx_load

2.1 Introduction

The em3xx_load utility is used in any loading operation. The default installation directory for em3xx_load is C:\Program Files\Ember \ISA3 Utilities\bin\.

2.2 Purpose

The em3xx_load utility is a command line (DOS console) application that can be used to program the flash memory space of the EM35x via the Serial Wire/JTAG interface.

2.3 Usage

To use the em3xx_load utility, you should be working from the command line.

Note: You can use the --help command at any time to print out full usage information. The examples listed in this document do not describe every feature of em3xx_load. Refer to the –-help command for full documentation of em3xx_load.

All em3xx_load commands, modifiers, and options are case sensitive.

The target of the command can be specified with --usb or --ip target options. If neither is specified, then the target option defaults to --usb 0. This default target option is used in these examples for brevity.

The path to the utility executable and the image files (<pathToUtility>/em3xx_load.exe <pathToImage>/sink.s37) must be specified when invoking em3xx_load but is left out of the examples for brevity.

See the EM35x Development Kit *Quick Start Guide* for the Debug Adapter (ISA3) configuration.

2.4 Example (load)

Command line input

```
$ em3xx_load.exe sensor.s37
```

Uploads the image in sensor.s37 to the EM35x. The flash pages encompassing the bytes defined in the .s37 file will be erased first. Once the chip is programmed, the application will be run.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load version 2.0b21.1314644979
Connecting to ISA via USB Device 0
DLL version 1.1.17, compiled Jun 21 2011 10:23:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Parse .s37 format for flash
Reset Chip
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Mark application image valid
Verifying bootloader and application
Run (by toggling nRESET)
DONE
```

2.5 Example (ip address)

Command line input

```
$ em3xx_load.exe --ip 192.168.220.244 sensor.s37
```

Uploads the image in sensor.s37 to the EM35x using an ISA defined by the IP address. The flash pages encompassing the bytes defined in the .s37 file will be erased first. Once the chip is programmed, the application will be run.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load version 2.0b21.1314644979
Connecting to ISA via IP address 192.168.220.244
DLL version 1.1.17, compiled Jun 21 2011 10:23:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Parse .s37 format for flash
Reset Chip
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Mark application image valid
Verifying bootloader and application
Run (by toggling nRESET)
DONE
```

2.6 Example (bootload image)

Command line input

```
$ em3xx_load.exe serial-uart-bootloader.s37 sensor.s37
```

This command loads the bootloader and application images simultaneously.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Parse .s37 format for flash
Parse .s37 format for flash
Reset Chip
WARNING: Replacing bootloader
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Mark application image valid
Verifying bootloader and application
Run (by toggling nRESET)
DONE
```

2.7 Example (enable write protect)

Command line input

```
$ em3xx_load.exe --programwrprot 000000
```

This command programs write protection enabled for all flash pages.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Setting Option Byte 4 to 0x00
Setting Option Byte 5 to 0x00
Setting Option Byte 6 to 0x00
Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

2.8 Example (disable write protect)

Command line input

```
$ em3xx_load.exe --programwrprot FFFFFF
```

This command programs write protection disabled for all flash pages.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Setting Option Byte 4 to 0xFF
Setting Option Byte 5 to 0xFF
Setting Option Byte 6 to 0xFF
Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

2.9 Example (erase and unprogram write protect)

Command line input

```
$ em3xx_load.exe --erasewrprot
```

This command erases the write protection option bytes for all flash pages, leaving those option bytes in an unprogrammed state. An unprogrammed state is equivalent to disabled write protection.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Erasing Option Byte 4
Erasing Option Byte 5
Erasing Option Byte 6
Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

2.10 Example (print flash contents)

Command line input

```
$ em3xx_load.exe --read @08040800-0804080F
```

This command prints all 16 bytes that comprise the 8 option bytes to the screen.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Getting memory from 0x08040800 through 0x0804080F
{address:  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F}
08040800: A5 5A FF 00 FF 00 FF 00 FF FF FF FF FF FF FF 00
Run (by toggling nRESET)
DONE
```

2.11 Example (read flash contents to file)

Command line input

```
$ em3xx_load.exe --read @mfb @fib @cib myreadfile.s37
```

This command uses the address aliases - mfb, fib, and cib - to read the entire flash contents of the chip into the output file, myreadfile.s37.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Getting memory from 0x08000000 through 0x0802FFFF
Getting memory from 0x08040000 through 0x080407FF
Getting memory from 0x08040800 through 0x08040FFF
Create image file
Run (by toggling nRESET)
DONE
```

2.12 Example (patch flash)

Command line input

```
$ em3xx_load.exe --patch @08040FFE=12 @08040FFF=34
```

This command patches the very last two bytes of the CIB to be 0x12 and 0x34.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Setting memory at 0x08040FFE to 0x12
Setting memory at 0x08040FFF to 0x34
Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

2.13 Example (patch using input file)

Command line input

```
$ em3xx_load.exe --patch sensor.ebl @08040FFE=12 @08040FFF=34
```

This command uses patch to simultaneously program an image and program the very last two bytes of the CIB to be 0x12 and 0x34.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Parse .ebl format for flash
Reset Chip
Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

2.14 Example (print CIB tokens)

Command line input

```
$ em3xx_load --cibtokensprint
```

Prints all known CIB manufacturing tokens to screen.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load version 2.0b21.1314644979
Connecting to ISA via USB Device 0
DLL version 1.1.21, compiled Oct 24 2011 16:19:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
'General' token group
TOKEN_MFG_CIB_OBS             [16 byte array ] : A55AFF00FF00FF00 FF00FF00FF00FF00
TOKEN_MFG_CUSTOM_VERSION      [16-bit integer] : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64       [ 8 byte array ] : FFFFFFFFFFFFFFFF
TOKEN_MFG_STRING              [16 byte string] : "0720093200460613" (16 of 16 chars)
                                                 3037323030393332 3030343630363133
TOKEN_MFG_BOARD_NAME          [16 byte string] : "ETRX357" (7 of 16 chars)
                                                 45545258333537FF FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID            [16-bit integer] : 0x1010
TOKEN_MFG_PHY_CONFIG          [16-bit integer] : 0xFFFF
TOKEN_MFG_BOOTLOAD_AES_KEY    [16 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE        [ 8 byte array ] : FFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM    [16-bit integer] : 0xFFFF
TOKEN_MFG_SYNTH_FREQ_OFFSET   [16-bit integer] : 0xFFFF
TOKEN_MFG_OSC24M_SETTLE_DELAY [16-bit integer] : 0xFFFF
'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device Implicit Cert [48 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                                        FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                                        FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CA Public Key        [22 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                                        FFFFFFFFFFFF
Device Private Key   [21 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
                                        FFFFFFFFFF
CBKE Flags           [ 1 byte array ] : FF
'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token group
Install Code Flags [ 2 byte array ] : FFFF
Install Code       [16 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CRC                [16-bit integer] : 0xFFFF
DONE
```

2.15 Example (dump CIB tokens)

Command line input

```
$ em3xx_load --cibtokensdump
```

Dump all known CIB manufacturing tokens to screen, in a format that can piped or copied and pasted into a file that can be written back to the chip using --cibtokenspatch.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load version 2.0b21.1314644979
Connecting to ISA via USB Device 0
DLL version 1.1.21, compiled Oct 24 2011 16:19:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
#'General' token group
TOKEN_MFG_CIB_OBS              : A55AFF00FF00FF00FF00FF00FF00FF00
TOKEN_MFG_CUSTOM_VERSION       : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64        : FFFFFFFFFFFFFFFF
TOKEN_MFG_STRING               : "0720093200460613"
TOKEN_MFG_BOARD_NAME           : "ETRX357"
TOKEN_MFG_MANUF_ID             : 0x1010
TOKEN_MFG_PHY_CONFIG           : 0xFFFF
TOKEN_MFG_BOOTLOAD_AES_KEY     : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE         : FFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM     : 0xFFFF
TOKEN_MFG_SYNTH_FREQ_OFFSET    : 0xFFFF
TOKEN_MFG_OSC24M_SETTLE_DELAY : 0xFFFF
#'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device                      Implicit                       Cert                          :
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CA Public Key        : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Device Private Key   : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
#'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token group
Install Code       : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
DONE
```

2.16 Example (patch CIB tokens)

Command line input

```
$ em3xx_load.exe --cibtokenspatch sample-tokens.txt
```

The --cibtokenspatch command takes a text file as input. The text file details the tokens that should be modified, including the ability to erase a token. You can specify as many tokens as you want to program in the text file.

The following is a sample text file that would write six of the tokens.

```
# This token file sets the EUI64 and MFG string:
TOKEN_MFG_CUSTOM_VERSION: 0x01FE
TOKEN_MFG_CUSTOM_EUI_64: 08090A0B0C0D0E0F
TOKEN_MFG_STRING: "Ember Rules!"
TOKEN_MFG_MANUF_ID: 0x1234
TOKEN_MFG_PHY_CONFIG: 0xFFF8
TOKEN_MFG_OSC24M_BIAS_TRIM: 0xFFF3
```

All blank lines are ignored. Lines that start with '#' are comment lines and will also be ignored. Token declarations are of the format "<token name> : <token data>". The token name is the same name used in the c source code on the EM35x. The token data is specified in one of three forms: byte array, integer, or string. Byte arrays are a series of hexadecimal characters interpreted as a little endian number. Integers are 8-bit, 16-bit, or 32-bit numbers interpreted as big endian. Strings are specified as quoted ASCII text. To specify that a token should be erased, set the token data to the keyword `!ERASE!`.

Any tokens not specified in the file are left untouched by the tool.

The meaning of the token data in the sample text file shown above is as follows:

- TOKEN_MFG_CUSTOM_VERSION – Set the version to 1, matching CURRENT_MFG_CUSTOM_VERSION defined in token-manufacturing.h.
- TOKEN_MFG_CUSTOM_EUI_64 - Define a custom EUI64 as an array of bytes. Showing this value in big endian format results in 0x0F0E0D0C0B0A0908.
- TOKEN_MFG_STRING – Define a plaintext manufacturing string.
- TOKEN_MFG_MANUF_ID – A 16-bit ID denoting the manufacturer of the device, often set to match your ZigBee-assigned manufacturer code.
- TOKEN_MFG_PHY_CONFIG – Bit 0 is cleared, indicating boost mode. Bit 1 is cleared, indicating an external PA is connected to the alternate TX path. Bit 2 is cleared, indicating an external PA is connected to the bi-directional RF path.
- TOKEN_MFG_OSC24M_BIAS_TRIM – Set the OSC24M bias trim value to 3.

For more information about the tokens that can be written, use the command:

```
$ ./em3xx_load.exe –-cibtokenspatch-help
```

Command line output

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Writing to address 0x08040810 for token 'TOKEN_MFG_CUSTOM_VERSION'
Writing to address 0x0804081A for token 'TOKEN_MFG_STRING'
Writing to address 0x0804083A for token 'TOKEN_MFG_MANUF_ID'
Writing to address 0x0804083C for token 'TOKEN_MFG_PHY_CONFIG'
Writing to address 0x080408EE for token 'TOKEN_MFG_OSC24M_BIAS_TRIM'
NOTE: Writing Custom EUI64 '08090A0B0C0D0E0F'.  Address 0x08040812.
Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

2.17 Scripting

Although the em3xx_load utility is designed to function as a standalone tool, it may also be integrated into a script designed for specific process integration requirements. The scripting environment should be able to run the utility as a command line tool. Command line syntax requirements are discussed in section 5. Error Messages.

3. em3xx_convert

3.1 Introduction

The em3xx_convert utility is used to convert files from one format to another. The default installation directory for em3xx_convert is C:\Program Files\Ember\ISA3 Utilities\bin\.

3.2 Purpose

The em3xx_convert utility is intended for converting s37 application image files into the ebl bootloader file format, and for encrypting EBL files for use by the secure bootloader.

3.3 Usage

To use the em3xx_convert utility, you should be working from the command line.

Note: You can use the --help command at any time to print out full usage information. The examples listed in this document do not describe every feature of em3xx_convert. Refer to the --help command for full documentation of em3xx_convert.

All em3xx_convert commands, modifiers, and options are case sensitive.

3.4 Convert .s37 to .ebl

Command line input

```
$ em3xx_convert.exe sensor.s37 sensor.ebl
```

Convert an s37 application image file into the Ember ebl bootloader file format. This is the primary function of em3xx_convert.

This example transaction would look like the following when executed.

Command line output

```
em3xx_convert (Version 1.0b13.1256666220)
Parse .s37 format for flash
Create ebl image file
DONE
```

3.5 Convert S37 to EBL using --imageinfo and -- timestamp

Command line input

```
$ ./em3xx_convert.exe --imageinfo "info string" --timestamp ffffffff sensor.s37 sensor.ebl
```

Generate an ebl output file from the s37 input file while overriding the ebl header imageInfo and timestamp fields.

This example transaction would look like the following when executed.

Command line output

```
em3xx_convert (Version 1.0b13.1256666220)
Parse .s37 format for flash
Setting EBL timestamp to 0xffffffff
Setting EBL imageInfo string to [info string]
Create ebl image file
DONE
```

3.6 Printing EBL Information

```
$ ./em3xx_convert.exe --print sensor.ebl
```

This will print information about the version contained in the EBL such as the timestamp it was created, AAT contents (application address table), and the target Phy, Micro, and Platform. Here is an example of the output:

```
$ em3xx_convert --print build/sensor-cortexm3-iar-em357-em3xx-dev0680/sensor.ebl
em3xx_convert (Version 3.1b15.1361560194)
Found EBL Tag = 0x0000, length  140, [EBL Header]
  Version:     0x0201
  Signature:   0xE350 (Correct)
  Flash Addr:  0x08002000
  AAT CRC:     0x8645BC42
  AAT Size:    128 bytes
    HalAppBaseAddressTableType
      Top of Stack:       0x200026A8
      Reset Vector:       0x08018F49
      Hard Fault Handler: 0x08018EC7
      Type:               0x0AA7
      HalVectorTable:     0x08002100
    Platform:         0x04 (CortexM3)
    Micro:            0x03 (em357)
    Phy:              0x03 (em3XX)
    Full AAT Size:    172
    Ember Version:    5.0.0.0
    Ember Build:      122
    Timestamp:        0x51310348 (Fri Mar 01, 2013 19:36:40 GMT [+0000])
    Image Info String:''
    Image CRC:        0x7C2937DC
    Customer Version: 0x00000000
Found EBL Tag = 0xFD03, length 1924, [Erase then Program Data]
  Flash Addr: 0x08002080
Found EBL Tag = 0xFD03, length 2052, [Erase then Program Data]
  Flash Addr: 0x08002800
(45 additional tags of the same type and length.)
Found EBL Tag = 0xFD03, length  188, [Erase then Program Data]
  Flash Addr: 0x08019800
Found EBL Tag = 0xFC04, length    4, [EBL End Tag]
  CRC: 0x1E19D76F
The CRC of this EBL file is valid (0xdebb20e3)
File has 48 bytes of end padding.
```

3.7 Encryption Key Generation

The em3xx_convert tool supports encrypting the EBL contents for use by a secure bootloader. To that end a key-file must be created that has a 128-bit key that will be used by the AES-CCM algorithm. The tool itself has the ability to randomly generate a key and this is the recommended way to create a key.

This is done with `--generate` option and supplying an output file.

```
$ ./em3xx_convert.exe --generate secret-key.txt
em3xx_convert (Version 3.1b15.1362167660)
Using Windows' Cryptographic random number generator
```

3.8 Encrypting an EBL File

The em3xx_convert tool supports encrypting EBL contents for use by a secure bootloader. A key file must have been previously created for this process. The tool supports transforming an S37 or unencrypted EBL into an encrypted EBL file. The following example shows how to encrypt a previously generated EBL file.

```
$ em3xx_convert --encrypt secret-key.txt build/sensor-cortexm3-iar-em357-em3xx-dev0680/sensor.ebl
em3xx_convert (Version 3.1b15.1361560194)
Unencrypted  input file:  build/sensor-cortexm3-iar-em357-em3xx-dev0680/sensor.ebl
Encrypt output file:      build/sensor-cortexm3-iar-em357-em3xx-dev0680/sensor.ebl.encrypted
Randomly generating nonce
Using Windows' Cryptographic random number generator
Created ENCRYPTED ebl image file
DONE
```

Note: To use an encrypted EBL a secure bootloader must be loaded on the chip.

3.9 Decrypting an EBL file

The em3xx_convert tool can also decrypt a previously encrypted EBL. The key file used to encrypt the EBL must be present and passed in via the command-line. The following is an example of how this can be done.

```
$ em3xx_convert.exe --decrypt secret-key.txt build/sensor-cortexm3-iar-em357-em3xx-dev0680/sensor.ebl.encrypted
em3xx_convert (Version 3.1b15.1362167660)
Unencrypted output file:  build/sensor-cortexm3-iar-em357-em3xx-dev0680/sensor.ebl
Encrypt input file:       build/sensor-cortexm3-iar-em357-em3xx-dev0680/sensor.ebl.encrypted
MAC matches. Decryption successful.
Created DECRYPTED ebl image file
DONE
```

3.10 Scripting

Although the em3xx_convert utility is designed to function as a standalone tool, it may also be integrated into a script designed for specific process integration requirements. The scripting environment should be able to run the utility as a command line tool. Command line syntax requirements are discussed in section 5. Error Messages.

4. em3xx_buildimage

4.1 Introduction

The em3xx_buildimage utility is used for building and modifying flash images. The default installation directory for em3xx_load is C:\Program Files\Ember\ISA3 Utilities\bin\.

4.2 Purpose

The em3xx_buildimage utility is a command line (DOS console) application. While the em3xx_buildimage utility can accept s37, hex, and ebl files as inputs and produce an s37 or hex file as an output, the most common usage is to combine multiple files into a single hex file that could be used for production programming. In addition to accepting image files as inputs, the utility can also be used for defining individual bytes, manipulating read and write protection, setting CIB tokens, patch operations, and reading information from existing image files. em3xx_buildimage is not strictly for hex file generation or for production programming. It is for any situation where multiple sources, files or command line address-data pairs, need to be combined into a single image. This includes patching an existing image. The resultant image output can be s37 in addition to hex, but the result can only be hex or s37.

All the available commands to em3xx_buildimage are derived from em3xx_load. That means that the behavior these commands perform will have the same result on the file as if they were performed on an actual chip.

The resultant output file defines bytes at a page granularity. If any input parameter defines a byte in a given hardware flash page, the resultant output file will define all of the bytes for that flash page. Any byte that is added automatically to the output file, because it wasn't defined in the input file but was part of a page that did have a defined byte, will be set to the erased state of 0xFF. The only exception to this rule is the read protection option byte. If the generated output file instantiates any bytes in the CIB and the read protection option byte is not defined, then read protection will be automatically defined and default to the disabled state.

4.3 Usage

To use the em3xx_buildimage utility, you should be working from the command line.

Note: You can use the --help command at any time to print out full usage information. The examples listed in this document do not describe every feature of em3xx_buildimage. Refer to the --help command for full documentation of em3xx_buildimage.

All em3xx_buildimage arguments are case sensitive.

Every invocation of em3xx_buildimage must specify a target chip, a chip image, and a command.

The examples use 'em357' as the target chip. Use 'efr32' if you are working with EFR32-related firmware.

The path to the utility executable and the image files (<pathToUtility>/em3xx_buildimage.exe <pathToImage>/sink.s37) must be specified when invoking em3xx_buildimage but is left out of the examples for brevity.

4.4 Example (basic)

Command line input

```
$ em3xx_buildimage.exe --chip em357 --chipimage myimage.hex serial-uart-bootloader.s37 sensor.ebl
```

Builds an EM357 chip image output file, myimage.hex, from the two input files, serial-uart-bootloader.s37, and sensor.ebl. If the file myimage.hex does not exist, it is created. If the file already exists, it is overwritten.

This example transaction would look like the following when executed.

Command line output

```
em3xx_buildimage version 2.0b07.1269849514
Targeting chip EM357
Parse .s37 format for flash
Parse .ebl format for flash
Verifying bootloader and application
Create image file
DONE
```

4.5 Example (enabling read protection)

Command line input

```
$ em3xx_buildimage.exe --chip em357 --chipimage myimage.hex serial-uart-bootloader.s37 sensor.ebl --
enablerdprot
```

Builds an EM357 chip image output file, myimage.hex, from the two input files, serial-uart-bootloader.s37 and sensor.ebl. At the same time, enable read protection by setting option byte 0 to the value 0x00. If the file myimage.hex does not exist, it is created. If the file already exists, it is overwritten.

This example transaction would look like the following when executed.

Command line output

```
em3xx_buildimage version 2.0b07.1269849514
Targeting chip EM357
Parse .s37 format for flash
Parse .ebl format for flash
Enabling read protection by setting Option Byte 0 to 0x00
Verifying bootloader and application
Create image file
DONE
```

4.6 Example (define specific bytes)

Command line input

```
$ em3xx_buildimage.exe --chip em357 --chipimage myimage.hex serial-uart-bootloader.s37 sensor.ebl @0804083C=FD
@080409FF=42
```

Builds an EM357 chip image output file, myimage.hex, from the two input files, serial-uart-bootloader.s37 and sensor.ebl. At the same time, set the byte at the address 0x0804083C to the value 0xFD and the byte at the address 0x080409FF to the value 0x42. Since these bytes are in the CIB, the entire CIB will be defined in the output file and the other bytes set to the value 0xFF; the read protection option byte will be set to 0xA5. If the file myimage.hex does not exist, it is created. If the file already exists, it is overwritten.

This example transaction would look like the following when executed.

Command line output

```
em3xx_buildimage version 2.0b07.1269849514
Targeting chip EM357
Parse .s37 format for flash
Parse .ebl format for flash
Setting memory at 0x0804083C to 0xFD
Setting memory at 0x080409FF to 0x42
Verifying bootloader and application
Create image file
DONE
```

4.7 Example (patch an image with specific bytes)

Command line input

```
$ em3xx_buildimage.exe --chip em357 --chipimage myimage.hex --patch @0804083C=FD @08
0409FF=42
```

Assuming the file myimage.hex was previously built, this command will patch the existing file with the two argument address-data pairs. If the file myimage.hex was not previously built, this file will be generated and only contain CIB data. But because the file will only contain CIB data, image verification would produce an error so the modifier --raw must be used. More data/image files may be added to this file using further patch operations.

This example transaction would look like the following when executed.

Command line output

```
em3xx_buildimage version 2.0b07.1269849514
Targeting chip EM357
Parse .hex format for flash
Setting memory at 0x0804083C to 0xFD
Setting memory at 0x080409FF to 0x42
Verifying bootloader and application
Create image file
DONE
```

4.8 Example (patching CIB tokens)

Command line input

```
$ em3xx_buildimage.exe --chip em357 --chipimage myimage.hex --cibtokenspatch cibtokens.txt
```

The functionality of the command --cibtokenspatch is identical to the functionality of this command in em3xx_load. The only difference is this command operates on the file myimage.hex instead of a chip. If myimage.hex does not exist, the file will be generated. Refer to the documentation of em3xx_load --cibtokenspatch for further details on the behavior of this command and the input file.

Note: em3xx_buildimage's implementation of --cibtokenspatch does not work with or know about the Smart Energy tokens since these tokens are specific to a given chip and not an image.

This example transaction would look like the following when executed.

Command line output

```
em3xx_buildimage version 2.0b07.1269849514
Targeting chip EM357
Parse .hex format for flash
Writing to address 0x08040810 for token 'TOKEN_MFG_CUSTOM_VERSION'
Writing to address 0x08040812 for token 'TOKEN_MFG_CUSTOM_EUI_64'
Writing to address 0x0804081A for token 'TOKEN_MFG_STRING'
Writing to address 0x0804083A for token 'TOKEN_MFG_MANUF_ID'
Writing to address 0x0804083C for token 'TOKEN_MFG_PHY_CONFIG'
Writing to address 0x080408EE for token 'TOKEN_MFG_OSC24M_BIAS_TRIM'
Verifying bootloader and application
Create image file
DONE
```

4.9 Example (dumping CIB tokens)

Command line input

```
$ em3xx_buildimage --chip em357 --chipimage myimage.hex --cibtokensdump
```

The functionality of the command --cibtokensdump is identical to the functionality of this command in em3xx_load. The only difference is this command operates on the file myimage.hex instead of a chip. Refer to the documentation of em3xx_load --cibtokensdump for further details on the behavior of this command and the input file.

Note: em3xx_buildimage's implementation of --cibtokensdump does not work with or know about the Smart Energy tokens since these tokens are specific to a given chip and not an image.

This example transaction would look like the following when executed.

Command line output

```
em3xx_buildimage version 2.0b21.1314644979
Targeting chip EM357
Parse .hex format for flash
#'General' token group
TOKEN_MFG_CIB_OBS             : A55AFF00FF00FF00FF00FF00FF00FF00
TOKEN_MFG_CUSTOM_VERSION      : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64       : FFFFFFFFFFFFFFFF
TOKEN_MFG_STRING              : "0720093200460613"
TOKEN_MFG_BOARD_NAME          : "ETRX357"
TOKEN_MFG_MANUF_ID            : 0x1010
TOKEN_MFG_PHY_CONFIG          : 0xFFFF
TOKEN_MFG_BOOTLOAD_AES_KEY    : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE        : FFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM    : 0xFFFF
TOKEN_MFG_SYNTH_FREQ_OFFSET   : 0xFFFF
TOKEN_MFG_OSC24M_SETTLE_DELAY : 0xFFFF
DONE
```

4.10 Example (read)

Command line input

```
$ em3xx_buildimage.exe --chip em357 --chipimage sensor.ebl --read @mfb mainflash.s37
```

The command --read behaves the same as em3xx_load's implementation of --read, except for that the chip image file is read instead of a chip. In this example, the entire main flash block is read from the file sensor.ebl and the output file mainflash.s37 is generated, containing the entire MFB as found in the file sensor.ebl.

This example transaction would look like the following when executed.

Command line output

```
em3xx_buildimage version 2.0b07.1269849514
Targeting chip EM357
Parse .ebl format for flash
Getting memory from 0x08000000 through 0x0802FFFF
Create image file
DONE
```

4.11 Scripting

Although the em3xx_buildimage utility is designed to function as a standalone tool, it may also be integrated into a script designed for specific process integration requirements. The scripting environment should be able to run the utility as a command line tool. Command line syntax requirements are discussed in section 5. Error Messages.

5. Error Messages

5.1 Introduction

All of the ISA3 utilities have been designed to use plain language error messages where possible. To facilitate scripting the utilities (em3xx_load, em3xx_convert, and em3xx_buildimage) use standard warning, error, and completion tags, simplifying parsing the utility output.

- Warnings: All warnings begin with “WARNING:” and will not halt execution but indicate a situation that is abnormal and should be studied carefully, understood, and corrected if needed.
- ERROR: All errors begin with “ERROR:” and will halt execution.
- DONE: All commands always close by printing “DONE”.

5.2 Example

Command line input

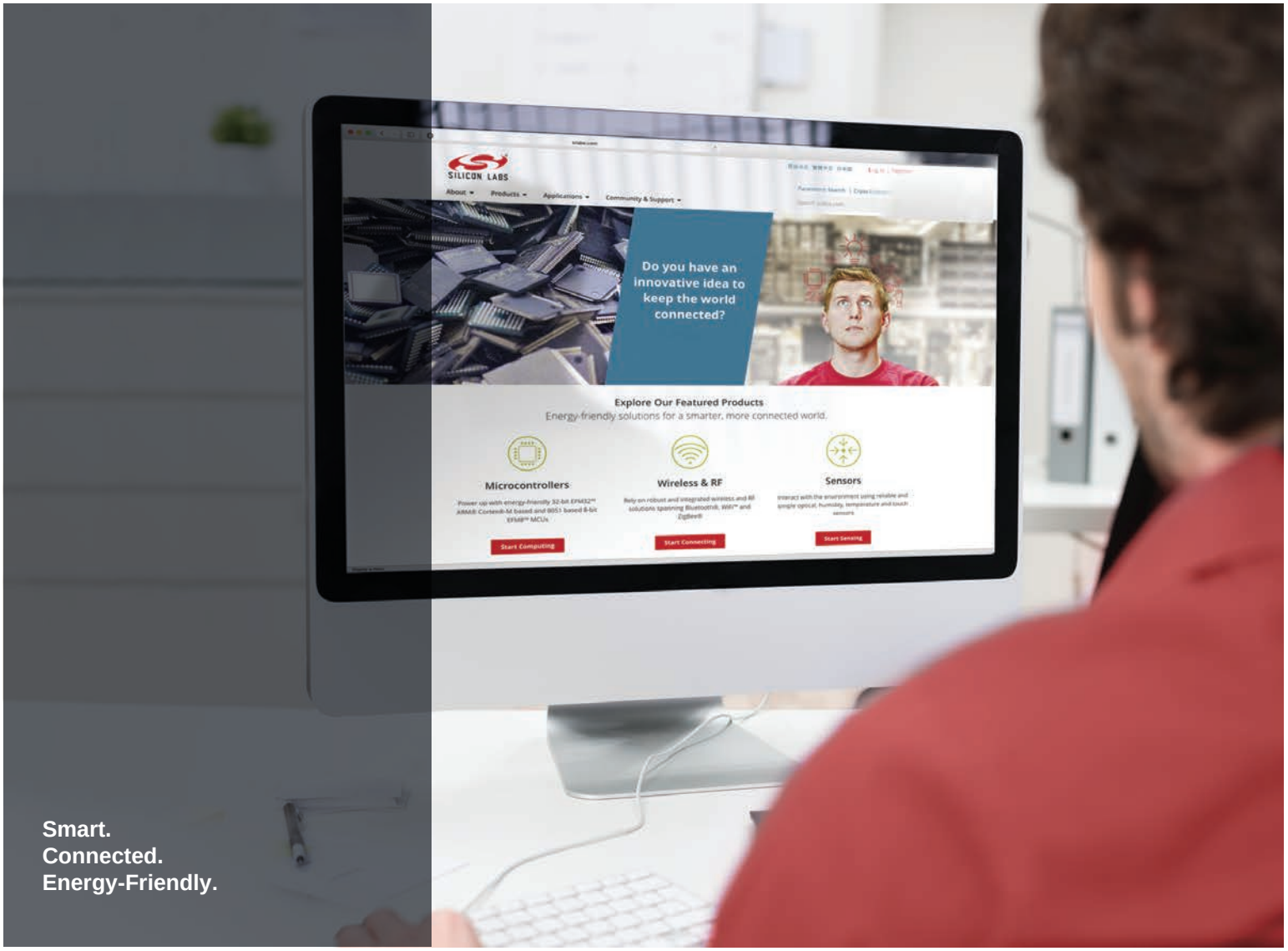
```
$ em3xx_convert.exe missingfile.s37 outputfile.ebl
```

This command fails because it cannot find the non-existent input file, missingfile.s37.

This example transaction would look like the following when executed.

Command line output

```
em3xx_convert (Version 1.0b13.1256666220)
ERROR: Could not open image file 'missingfile.s37'
DONE
```



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com