UM0924
User manual

EmberZNet™ utilities

1 Introduction

Purpose

This document describes how and when to use the standalone EmberZNet™ utilities supplied with EmberZNet™ software.

It is recommended 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.

Audience

This document is intended for software and hardware engineers who are responsible for building an embedded mesh networking solution using the STM32W108 system-on-chip. This document assumes that the reader has a solid understanding of embedded systems design.

Document organization

STM32W108 kit customers are eligible for training and technical support. You can use the STMicroelectronics web site, www.st.com/mcu, STM32W section to obtain information about all STM32W108 products and services, and to sign up for product support.

Contents

2 Introduction to EmberZNet utilities

Designers using EmberZNet's ZigBee Network Stack software (EmberZNet) and development tools must use one of the standalone 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).

These tools may vary with the specific version of the installed STMicrolectronics product. If you do not have one of these utilities, contact support at www.st.com/mcu, STM32W section to learn how to get a copy.

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

Table 1. Utilities covered in this document

Utility	Description	Section
em3xx_load	Programming utilities for STM32W108 system-on-chip.	*Section 3*
em3xx_convert	Converts files from one format to another.	*Section 4*

2.1 Tool overview

EmberZNet's 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.

2.2 File format overview

The STM32W108 family of tools works with different file formats: .bin, .s37 and .ebl (used for bootloaders). Each file format serves a slightly different purpose. The EmberZNet stack contains a conversion tool, em3xx_convert, which converts applications from one file format to another (.s37 to .ebl). The s37 and ebl file formats are summarized below.

2.2.1 S-record file format

EmberZNet sample applications use 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 STM32W108 using the em3xx_load utility. The s37 format can represent any combination of any byte of Flash memory in the STM32W108. The em3xx_convert utility can be used to convert the s37 format to ebl format for use with EmberZNet bootloaders.

2.2.2 Ebl bootloader file format

The ebl bootloader 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 EmberZNet's 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 memory 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.

*Table 2*summarizes the inputs and outputs for the different file formats.

Table 2. File format summary

	Inputs			Outputs		
	ebl	**s37**	**chip**	**ebl**	**s37**	**chip**
em3xx_load	X	X				X
em3xx_convert		X		X		

3 EM3XX_LOAD

3.1 Introduction

The em3xx_load utility is used in any loading operation. It is delivered within the stack (tools/em3xx folder).

3.2 Purpose

The em3xx_load utility is a command line (DOS console) application that can be used to program the Flash memory space of the STM32W108 via the Serial Wire/JTAG interface.

3.2.1 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.

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

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.

3.3 Example (load)

Command line input

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

Uploads the image in sensor.s37 to the STM32W108. The Flash memory 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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.4 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 1.0b27.1264522950)
Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.5 Example (enable write protect)

Command line input

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

This command programs write protection enabled for all Flash memory pages.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
```

```
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.6 Example (disable write protect)

Command line input

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

This command programs write protection disabled for all Flash memory pages.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.7 Example (erase and unprogram write protect)

Command line input

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

This command erases the write protection option bytes for all Flash memory 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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.8 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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.9 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 memory 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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.10 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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.11 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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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
```

3.12 Example (patch to a target file)

Command line input

```
$ em3xx_load.exe --patch serial-uart-bootloader.s37 sensor.ebl
@08040FFF=42 --targetfile mycompleteimage.s37
```

This command uses patch to combine a bootloader, an application, and custom CIB bytes into a single image file.

This example transaction would look like the following when executed.

Command line output

```
em3xx_load (Version 1.0b27.1264522950)

Parse .s37 format for flash
Parse .ebl format for flash
Setting memory at 0x08040FFF to 0x42
Create image file
DONE
```

3.13 Example (print CIB tokens)

Command line input

```
$ em3xx_load.exe --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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108


'General' token group
TOKEN_MFG_CIB_OBS           [16 byte array ] : A55AFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_CUSTOM_VERSION    [16-bit integer] : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64     [ 8 byte array ] : FFFFFFFFFFFFFFFF
TOKEN_MFG_STRING            [16 byte string] : "" (0 of 16 chars)
                                               FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME        [16 byte string] : "" (0 of 16 chars)
                                               FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID          [16-bit integer] : 0xFFFF
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

'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
```

3.14 Example (dump CIB tokens)

Command line input

```
$ em3xx_load.exe --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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108


#'General' token group
TOKEN_MFG_CIB_OBS           : A55AFFFFFFFFFFFFFFFFFFFFFFFFFFFF
TOKEN_MFG_CUSTOM_VERSION    : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64     : FFFFFFFFFFFFFFFF
TOKEN_MFG_STRING            : ""
TOKEN_MFG_BOARD_NAME        : ""
TOKEN_MFG_MANUF_ID          : 0xFFFF
TOKEN_MFG_PHY_CONFIG        : 0xFFFF
TOKEN_MFG_BOOTLOAD_AES_KEY  : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE      : FFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM  : 0xFFFF

#'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device Implicit Cert :
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CA Public Key        : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Device Private Key   : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

#'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token
group
```

```
Install Code        : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF


DONE
```

3.15 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: 0008E102000001FE
TOKEN_MFG_STRING: "ST 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 STM32W108. 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 are 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 0x0008E102000001FE.
- `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 1.0b27.1264522950)

Connecting to ISA via USB Device 0
DLL version 1.1.9, compiled Jan 29 2010 18:36:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting STM32W108
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 '0008E102000001FE'.  Address 0x08040812.

Create image file
Install RAM image
Verify RAM image
Install Flash image
Verify Flash image
Run (by toggling nRESET)
DONE
```

3.16 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*.

4 EM3XX_CONVERT

4.1 Introduction

The em3xx_convert utility is used to convert files from one format to another. It is delivered within the stack (tools/em3xx folder).

4.2 Purpose

The em3xx_convert utility is intended for converting s37 application image files into ebl bootloader file format.

4.3 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.*

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

4.4 Example (convert .s37 to .ebl)

Command line input

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

Convert an s37 application image file into 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 1.0b25.1261162476)

Parse .s37 format for flash
Create ebl image file
DONE
```

4.5 Example (using --imageinfo and -- timestamp)

Command line input

```
$ ./em3xx_convert.exe --imageinfo "info string" --timestamp
fffffff 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 1.0b25.1261162476)

Parse .s37 format for flash
Setting EBL timestamp to 0xffffffff
Setting EBL imageInfo string to [info string]
Create ebl image file
DONE
```

4.6 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*.

5 Error messages

5.1 Introduction

All of the EmberZNet utilities have been designed to use plain language error messages where possible. To facilitate scripting the utilities 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 1.0b25.1261162476)

ERROR: Could not open image file 'missingfile.s37'
DONE
```

6 Revision history

Table 3. Document revision history

Date	Revision	Changes
18-Mar-2010	1	Initial release.