

# EmberZNet Utilities Guide

For the EM250 and the EM260 Network Co-Processors

2 October 2008

120-4020-000D



Ember Corporation  
47 Farnsworth Street  
Boston, MA 02210  
+1 (617) 951-0200  
[www.ember.com](http://www.ember.com)



Copyright © 2008 by Ember Corporation

All rights reserved.

The information in this document is subject to change without notice. The statements, configurations, technical data, and recommendations in this document are believed to be accurate and reliable but are presented without express or implied warranty. Users must take full responsibility for their applications of any products specified in this document. The information in this document is the property of Ember Corporation.

Title, ownership, and all rights in copyrights, patents, trademarks, trade secrets and other intellectual property rights in the Ember Proprietary Products and any copy, portion, or modification thereof, shall not transfer to Purchaser or its customers and shall remain in Ember and its licensors.

No source code rights are granted to Purchaser or its customers with respect to all Ember Application Software. Purchaser agrees not to copy, modify, alter, translate, decompile, disassemble, or reverse engineer the Ember Hardware (including without limitation any embedded software) or attempt to disable any security devices or codes incorporated in the Ember Hardware. Purchaser shall not alter, remove, or obscure any printed or displayed legal notices contained on or in the Ember Hardware.

Ember, Ember Enabled, EmberZNet, InSight, and the Ember logo are trademarks of Ember Corporation.

All other trademarks are the property of their respective holders.



---

## Contents

<b>About This Guide</b> .....	<b>i</b>
Purpose .....	<b>i</b>
Audience.....	<b>i</b>
Document Organization .....	<b>i</b>
Documentation Conventions .....	<b>ii</b>
<b>1 Introduction to EmberZNet Utilities</b> .....	<b>1</b>
Tool Overview .....	<b>1</b>
File Format Overview.....	<b>1</b>
<b>2 EM2xx_LOAD</b> .....	<b>4</b>
Introduction.....	<b>4</b>
Purpose .....	<b>4</b>
Usage .....	<b>4</b>
Example .....	<b>5</b>
Scripting .....	<b>6</b>
<b>3 EM2XX_READ</b> .....	<b>7</b>
Introduction.....	<b>7</b>
Purpose .....	<b>7</b>
Usage .....	<b>7</b>
Example 1 .....	<b>8</b>
Example 2.....	<b>8</b>
Scripting .....	<b>9</b>
<b>4 EM2XX_CONVERT</b> .....	<b>10</b>
Introduction.....	<b>10</b>
Purpose .....	<b>10</b>
Usage .....	<b>10</b>
Example .....	<b>11</b>
Scripting .....	<b>11</b>
<b>5 EM2XX_PATCH</b> .....	<b>12</b>
Introduction.....	<b>12</b>

	<b>Purpose .....</b>	<b>12</b>
	<b>Usage .....</b>	<b>12</b>
	<b>Example (.hex file) .....</b>	<b>13</b>
	<b>Example (token) .....</b>	<b>14</b>
	<b>Example (EUI64).....</b>	<b>14</b>
	<b>Scripting .....</b>	<b>15</b>
<b>6</b>	<b>Error Messages.....</b>	<b>16</b>
	<b>Introduction.....</b>	<b>16</b>
	<b>Syntax Error .....</b>	<b>16</b>
	<b>Instructional Error .....</b>	<b>17</b>
	<b>Index .....</b>	<b>19</b>

---

## About This Guide

### Purpose

This document describes how and when to use several of the stand-alone utilities supplied with EmberZNet software.

Ember 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.

### Audience

This document is intended for project managers and for software engineers who are responsible for building an embedded mesh networking solution using the Ember radio, the EmberZNet network stack, and tools.

This document assumes that the reader has a solid understanding of embedded systems design and programming in the C language. Experience with networking and radio frequency systems is useful but not expected.

### Document Organization

Developer kit customers are eligible for training and technical support. You can use the Ember website, [www.ember.com](http://www.ember.com) to obtain information about all Ember products and services, and to sign up for product support.

You can also contact Ember technical support at [support@ember.com](mailto:support@ember.com).

If you have any questions about your Developer Kit, contact your Ember account representative at one of the following locations:

<b>United States</b>	343 Congress Street Boston, MA 02210 Telephone: 617-951-0200 Fax: 617-951-0999
<b>Asia/Pacific</b>	HK Spinners Industrial Bldg, Phase 5 5/F Flat D 760-762 Cheung Sha Wan Road Kowloon Hong Kong Telephone: +852-8120-5375

<b>Europe</b>	Unit 300 Science Park Milton Road Cambridge CB4 0XL, UK Telephone: 44 (0) 1223 423322 Fax: 44 (0) 1223 423390
---------------	---------------------------------------------------------------------------------------------------------------------------

## Documentation Conventions

Notation	Meaning	Example
<b>Bold</b>	A GUI label	<b>Node Key</b>
UPPERCASE	A keyboard key	ENTER
	Delimits a hierarchy of menu options, which ends with the option to choose.	Open   Save
Courier	Identifies file names and program identifiers, such as constants and function names.	<pre>EMBER_SLEEPY_END_DEVICE serial.h emberStartScan()</pre>

# 1

## Introduction to EmberZNet Utilities

The designer who uses Ember's ZigBee Network Stack software (EmberZNet) and development tools will have occasional need to use one of the stand-alone utilities installed with the rest of your software. These utilities are software tools that perform a single specialized task or a small range of tasks not integrated into the normal IDE.

These tools may vary with the specific version of your installed Ember product. If you do not have one of these utilities, you can contact [support@ember.com](mailto:support@ember.com) to learn how to get a copy.

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

Utility	Description	Chapter
em2xx_load	Loads program & data into the flash of the device.	2
em2xx_read	Reads flash out of the device into a file.	3
em2xx_convert	Converts files from one format to another.	4
em2xx_patch	Updates specific addresses in either an image or the device with new data.	5

### Tool Overview

Ember'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.

Two sets of tools have been created to address this and related needs; these are called the em2xx and EM2USB utilities. The em2xx family of utilities is created for use with the InSight Adapter while the EM2USB family is used with the InSight USBLink. This document covers the use of the em2xx utilities. Use of the EM2USB utilities is discussed in InSight USB Link User's Guide (120-4022-000). This document is installed with the InSight USBLink Software installer. It may also be obtained by contacting Ember's customer support ([support@ember.com](mailto:support@ember.com)) or on our website [www.ember.com](http://www.ember.com).

Each tool in the em2xx family has a very specific function described in detail in the following chapters.

### File Format Overview

The em2xx family of tools works with three different file formats. These formats are xdv/xpv, ebl and hex. Each file format serves a slightly different purpose. The em2xx family contains a conversion tool em2xx\_convert that converts applications from one file format to another. Having said this, not all file formats are two way convertible. For instance, it is possible to convert a xdv/xpv file into an .ebl file but not the other way around. The file formats are as follows:

## XDV / XPV File Format

These are the two ASCII files created by the xIDE compiler as its raw output. The .xpv file contains executable “Program” code, while the .xdv file contains constant “Data” and initializers. These file formats can normally contain either a bootloader or an application image, but never a simulated EEPROM. It is possible to manually create an .xdv file that can contain manufacturing tokens. This format is generally used during early development. It can be used for low volume manufacturing, but is slightly inefficient as multiple load steps are required for each separate section of the flash.

## EBL File Format

This is the Ember Bootloader Format that is generated by the `em2xx_convert` utility included with xIDE for EM250 as a post-build step. This file format can only represent an application image.

The ebl format is designed to be an efficient and fault-tolerant image format for use with Ember’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 in place.

Although intended for use with a bootloader, the `em2xx_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 xdv/xpv file format when loading the bootloader itself.

## Hex File Format

The em2xx family of utilities also supports a .hex file that conforms to the Intel Hex-32 file format. This format includes a complete image of both the main flash and information flash, including an application image and optionally a bootloader, simulated EEPROM, and manufacturing tokens.

The hex file format is intended to be used in situations where a complete and clean application image needs to be put on a chip. When you load a .hex image, `em2xx_load` will automatically erase the entire chip to ensure that the image put on the chip is an exact duplicate of the image in the .hex file. For this reason, Ember does *not* recommend that you use the .hex image during debug and development, since any persisted information such as bindings, panId, network status, and so on will be erased during the upload process.

A hex file can be generated by using `em2xx_convert` included with xIDE for the EM250 with xdv/xpv or ebl files as source images, or by using `em2xx_read` to create an image from a chip that has already been programmed. Pages of data (1024 bytes) that consist entirely of the value 0xFF are not included in the .hex file created by `em2xx_read.exe`. For this reason, `em2xx_load.exe` performs a complete erase of the application flash area before performing a load from a .hex file. This default behavior ensures that the loaded application image will be identical to the original, and also matches the behavior of programming a .hex file on to a brand new chip during volume production.

The manufacturing tokens included in this format may also be directly manipulated by using the `-filetarget` option of `em2xx_patch`. This file format is intended to be used for volume production and allows a very efficient way to fully program the flash of new chips without the need for multiple steps. This format is also used by gang programmers and distributors that offer programming services for the EM250 and EM260.

Table 1. File Format Summary

	Inputs				Outputs		
	Ebl	Hex	xpv/xdv	chip	hex	Ebl	chip
em2xx_load	X	X	X				X
em2xx_read				X	X		
em2xx_patch		X*		X*	X		X
em2xx_convert	X		X		X	X	

**Note:** \*em2xx\_patch makes modifications to an existing hex image or chip directly. The list of modifications to make can be specified on the command line, or as an xpv/xdv file.

# 2 EM2xx\_LOAD

## Introduction

The `em2xx_load` utility is used in application loading operations. By default, it may be found in the following locations:

<u>Device</u>	<u>Directory</u>
EM250	C:\Program Files\Ember\xIDE_EM250\SIF\bin\
EM260	C:\Program Files\Ember\EmberZNet3.0\tool\

## Purpose

The `em2xx_load` utility is a command line (DOS console) application that can be used to program the flash memory space of the EM250 via the SIF interface.

## Usage

To use the `em2xx_load` utility, you should be working from the command line. The `[-h]` option can be used at any time to print out the usage information listed below. All command options are case sensitive.

```
em2xx_load [-sid<slaveid>] [-id<podid>] [-bt1 <filename>]
           [-erase] [-ramCodeOnly] [-noprogram] [-noverify]
           [-Mfg] [-Override] [-Run] [-Reset] <filename(s)>
```

<u>Arguments</u>	<u>Description</u>
<code>[-sid&lt;slaveid&gt;]</code>	Specify SIF Slave id # or IP address/hostname
<code>[-id&lt;podid&gt;]</code>	Specify SIF Pod id # or IP address/hostname
<code>[-bt1 &lt;filename&gt;]</code>	Specify bootloader image to use if needed
<code>[-erase]</code>	Perform full erase of the application flash space before programming. This option defaults to true when loading from a file in the .hex format.
<code>[-ramCodeOnly]</code>	Load ram code only image. Used with <code>-Run</code> jumps to start of ram (F600)
<code>[-noprogram]</code>	Do not Program the flash
<code>[-noverify]</code>	Do not Verify the flash
<code>[-Mfg]</code>	Load Manuf flash page *ONCE* via .xdv
<code>[-Override]</code>	Enables re-programming of addresses in the manufacturing token area. This option should be used with caution and only in development

scenarios. There is slight risk of a chip becoming corrupted and inoperable if a serious error (for example: power failure, crash, loss of connection) occurs during the re-programming of the manufacturing tokens

```
[-Run]           Start node via SW_RESET afterward
[-Reset]        SW_RESET then stop node afterward
{filename}*     Loads .xpv+.xdv, .ebl or .hex for each file/project named.
```

\* If the filename(s) are project name(s) or .xpv/.xdv filename(s), the program loads both .xpv and .xdv project files. If the filename(s) are .ebl filename(s), the program loads just the .ebl project file. If the file is a .hex file, the program loads all accessible memory indicated within the .hex file.

## Example

```
./em2xx_load.exe -sid192.168.223.18 -Run -Override ./test/traffic.hex
```

Uploads the application information included in traffic.hex up to the em250 through the adapter with IP 192.168.223.18. Because this command loads a .hex file, the chip will first be completely erased. (all customer data, application, and SIMEE gone). This erase operation does not extend to the manufacturing token portion of flash.

Once the chip is programmed, the application will be run. If the hex image provided contains manufacturing tokens, the -Override flag must be included because it is required to write the manufacturing tokens more than once.

**Caution:** The use of the -Override flag will cause ALL manufacturing tokens to be briefly erased and rewritten. Please be sure that you do not cut power to the device during this process. Any loss of power could result in the deletion of critical Ember manufacturing tokens which would make the device unusable.

This example transaction would look like the following when executed:

## Command line input

```
./em2xx_load.exe -sid192.168.223.18 -Run -Override ./test/traffic.hex
```

## Command line output:

```
em2xx_load.exe Rev 1.9 b2 (Jan 18 2007 10:09:31)
Loading .hex file ./test/traffic.hex...
SIF Slave id change from 1 to 7 (pod 7) succeeded
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= DbgEna performed
XAP2= Reset performed
Programming FLASH...
XAP2= Run performed
Erasing entire FLASH device...
FLASH device entirely erased.
BOOTI 0000-0013 (0014)
BOOTL 0014-13ff (13ec)
STKIC 1400-143f (0040)
BOOTL 1440-bfff (abc0)
BONST c000-ddff (1e00)
FLASH e000-ebff (0c00)
FLASH ee00-ffff (0200)
```

```
Total Flash Used=ec00 words
MFGcu 0040-01ff (01c0)
Total Manuf Used=01c0 words
XAP2= Stop performed
Verifying FLASH...
```

```
****Verification: SUCCESS****
XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET
```

## Scripting

Although the em2xx\_load utility is designed to function as a stand-alone 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.

# 3

## EM2XX\_READ

### Introduction

The em2xx\_read utility is used to read application data from a chip. By default, em2xx\_read is located in the following locations.

<u>Device</u>	<u>Directory</u>
EM250	C:\Program Files\Ember\xIDE_EM250\SIF\bin\
EM260	C:\Program Files\Ember\EmberZNet3.0\tool\

### Purpose

em2xx\_read is a command line (DOS console) application that can be used to examine (dump) or generate a .hex file from the flash of the EM250 via SIF. These .hex files can then be used for production programming or debugging.

em2xx\_read can be used to create a .hex file representing the entire flash image from a chip. Pages of data (1024 bytes) which consist entirely of the value 0xFF are not included in the .hex file created by em2xx\_read.exe. For this reason, em2xx\_load.exe performs a complete erase of the application flash area before performing a load from a .hex file. This default behavior ensures that the loaded application image will be identical to the original.

### Usage

To use the em2xx\_read utility, you should be working from the command line. The [-h] option can be used at any time to print out the usage information below. All command options are case sensitive.

```
em2xx_read [-sid<slaveid>] [-id<podid>] [-Mfg] [-Run]
[-Reset] [-excludeMfg] [-outfile <filename>]
```

#### Arguments

[-sid<slaveid>]

[-id<podid>]

[-Mfg]

[-excludeMfg]

[-includeSIMEE]

#### Description

Specify SIF Slave id # or IP address/hostname

Specify SIF Pod id # or IP address/hostname

Subsequent addresses refer to manufacturing token area word offsets

Ignores manufacturing tokens when writing unloaded data to a .hex file. By default customer area manufacturing tokens are included in any .hex file that this application creates.

Includes SIMEE portion of data (addresses 0xF000 - 0xFFFF) when writing to a .hex file.

<code>[-Run]</code>	Start em2xx node via SW_RESET afterward
<code>[-Reset]</code>	SW_RESET then stop em2xx node afterward
<code>{startAddr[+size -endAddr]}*</code>	Flash/Manuf range(s) to unload
<code>[-outfile &lt;filename&gt;]</code>	Writes entire address space out to a file of type provided. Currently only .hex is supported.

## Example 1

```
$ ./em2xx_read.exe -sid192.168.223.18 -Run -outfile ./test/unload.hex
```

This example reads all data on the chip located at IP 192.168.223.18, including application data, bootloader and customer manufacturing tokens, restarts the chip and then writes it all out to a single file called unload.hex.

This example transaction would look like the following:

### Command line input

```
$ ./em2xx_read.exe -sid192.168.223.18 -Run -outfile ./test/unload.hex
```

### Command line output

```
em2xx_read.exe Rev 0.1 b4 (Jan 18 2007 10:09:33)
SIF Slave id change from 1 to 7 (pod 7) succeeded
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= DbgEna performed
XAP2= Reset performed
fAddrLo=0, fAddrHi=efff
mAddrLo=40, mAddrHi=1be
XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET
```

## Example 2

```
$ ./em2xx_read.exe -sid192.168.223.206 -Mfg @40+24
```

This example outputs the first 24 bytes from the customer manufacturing token area. Since the -Run flag is not included, once the tokens are output to the screen, the target device is stopped and ready for debugging.

This example transaction would look like the following:

### Command line input

```
$ ./em2xx_read.exe -sid192.168.223.206 -Mfg @40+24
```

## Command line output

```

$ ./em2xx_read.exe -sid192.168.223.206 -Mfg @40+24
em2xx_read.exe Rev 0.1 b4 (Jan 18 2007 10:09:33)
SIF Slave id change from 1 to 7 (pod 7) succeeded
XAP2= DbgEna performed
  XAP2= Stop performed
  XAP2= DbgEna performed
  XAP2= Reset performed
Manuf:
###--> addrHi=57 addrLo=40
#0040:  FFFF.FFFF.FFFF.FFFF.FFFF.FFFF.FFFF.FFFF .....
#0048:  FFFF.FFFF.FFFF.FFFF.FFFF.FF64.6576.3034 .....dev04
#0050:  3730.20FF.FFFF.FFFF.FFFF.0000.FFFF.FFFF 70 .....
  XAP2= Stop performed
  XAP2= Reset performed
XAP is stopped, ready to debug ...

```

## Scripting

Although the `em2xx_read` utility is designed to function as a stand-alone 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.

# 4 EM2XX\_CONVERT

## Introduction

The `em2xx_convert` utility replaces the `em2xx_ebl` .ebl conversion utility. It is used to convert files from one format to another. By default, `em2xx_convert` can be found in the following locations:

<u>Device</u>	<u>Directory</u>
EM250	C:\Program Files\Ember\xIDE_EM250\SIF\bin\
EM260	Not applicable

## Purpose

The `em2xx_convert` utility is intended for use in converting xIDE .xpv/.xdv<sup>1</sup> application files into ember's .ebl bootloader format or the Intel Hex format (.hex).

When creating a .hex file with `em2xx_convert`, you have the option to include a .xpv/.xdv representation of the application bootloader and/or a .xdv representation of the customer manufacturing tokens. For a complete explanation of the three file formats, see Chapter 1 above.

The file format conversions supported by `em2xx_convert` are as follows:

em2xx format conversions supported			
Input	.ebl		.xpv/.xdv
Output	.ebl	.hex	

## Usage

To use the `em2xx_convert` utility, you should be working from the command line. All command options are case sensitive.

```
em2xx_convert [-silent] [-D{PLAT|MICRO|PHY|BOARD}=<value>]
[-bt1 <filename>] [-Mfg <filename>]
-outfile <filename> <filename>
```

<u>Arguments</u>	<u>Description</u>
<code>[-silent]</code>	Operate silently.
<code>[-D{PLAT MICRO PHY BOARD}=&lt;value&gt;]</code>	Define build parameter
<code>[-bt1 &lt;filename&gt;]</code>	Specify bootloader image to include (use in creation of .hex files only)
<code>[-Mfg &lt;filename&gt;]</code>	Specify manufacturing token image to include (use in creation of .hex files only)

<sup>1</sup> The .xdv file contains the manufacturing tokens.

`-outfile <filename>` Provide the name of the output file with `.ebl` or `.hex` extension. The extension of the `<filename>` determines the type of file created. Defaults to `ebl` format in case where extension is not recognized.

`<filename>` Name of the source application file. Converts files of type `.xpv/.xdv`, `.ebl`, and `.hex` into file of type indicated by `-outfile` argument.

## Example

```
./em2xx_convert.exe -bt1
./test/standalone-bootloader-em250.xdv -Mfg ./test/mfg-tokens.xdv -outfile
./test/traffic.hex ./test/traffic.xdv
```

Creates a file called `./test/traffic.hex` which includes customer manufacturing tokens, the standalone bootloader and the training application into a single file written in the Intel Hex format. This hex file is then ready for use in production programming.

This example transaction would look like the following:

## Command line input

```
$ ./em2xx_convert.exe -bt1 ./test/standalone-bootloader-em250.xdv -Mfg./test/mfg-
tokens.xdv -outfile ./test/traffic.hex ./test/traffic.xdv
```

## Command line output

```
em2xx_convert.exe Rev 1.1 b6 (Jan 18 2007 10:09:28)
Loading code file ./test/standalone-bootloader-em250.xpv...
// ROM START ADDRESS 0
// WORDSIZE 16
Loading data file ./test/standalone-bootloader-em250.xdv...
// ROM START ADDRESS 28
// WORDSIZE 8
Loading code file ./test/mfg-tokens.xpv...
Loading data file ./test/mfg-tokens.xdv...
// ROM START ADDRESS 5000
// WORDSIZE 8
Loading code file ./test/traffic.xpv...
// ROM START ADDRESS 0
// WORDSIZE 16
Loading data file ./test/traffic.xdv...
// ROM START ADDRESS 0
// WORDSIZE 8
fAddrLo=0, fAddrHi=efff
mAddrLo=40, mAddrHi=57
Successfully created file ./test/traffic.hex in the Ember Intel Hex (hex)
format
```

## Scripting

Although the `em2xx_convert` utility is designed to function as a stand-alone 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 the next section.

# 5

## EM2XX\_PATCH

### Introduction

The em2xx\_patch utility is used to modify applications that are currently loaded on a chip or are represented in a .hex file. By default, it is found in the following locations:

<u>Device</u>	<u>Directory</u>
EM250	C:\Program Files\Ember\xIDE_EM250\SIF\bin\
EM260	C:\Program Files\Ember\EmberZNet3.0\tool\

### Purpose

The em2xx\_patch utility is a command line (DOS console) application that is used to program selected portions of the flash memory space of the EM250 via the SIF interface. Em2xx\_patch can also be used to update specific portions of a .hex file by using the -filetarget option instead of the -sid option.

### Usage

To use the em2xx\_patch utility, you should be working from the command line. All command options are case sensitive.

```
em2xx_patch [-sid<slaveid>] [-id<podid>] [-loadCert <certFile>] [-loadInst
<instFile>] [-filetarget] <filename> [-noprogram] [-noverify] [-Mfg] [-Override]
[-Run] [-Reset] {@addr=value>*
```

#### Arguments

[-sid<slaveid>]

#### Description

Specify SIF Slave id # or IP address/hostname

[-id<podid>]

Specify SIF Pod id # or IP address/hostname

[-loadCert <certFile>]

Loads the security certificate onto the target device. The format of the certificate file is as follows:

CA Public Key: <22 hex bytes>

Device Implicit Cert: <48 hex bytes>

Device Private Key: <21 hex bytes>

[-loadInst <InstFile>]

Loads the installation code onto the target device. The format of the installation file is as follows:

EUI: <8 hex bytes>

Install Code: <16 hex bytes>

CRC: <2 hex bytes>

	Additional logic is performed on the EUI. If the EUI is an Ember EUI, then the installation code load is aborted if the EUI does not match the manufacturer's EUI on the chip. If the EUI is not an Ember EUI, then the EUI will be written into the custom EUI area.
<code>[-filetarget &lt;filename&gt;]</code>	Patch can target a file instead of a device. Use this option to indicate the .hex file to write the patch into. If the <code>-filetarget</code> option is not specified, Patch assumes a device target.
<code>[-noprogram]</code>	Do not Program the flash
<code>[-noverify]</code>	Do not Verify the flash
<code>[-Mfg]</code>	Required if the addresses provided for the patch are part of the manufacturing token area of flash on a chip or in a hex file.  When interacting with a chip, will normally only allow writing if the address was not previously programmed.
<code>[-Override]</code>	Enables re-programming of addresses in the manufacturing token area. This option should be used with caution and only in development scenarios.  There is slight risk of a chip becoming corrupted and inoperable if a serious error (for example: power failure, crash, loss of connection) occurs during the re-programming of the manufacturing tokens.
<code>[-Run]</code>	Start node via SW_RESET afterward
<code>[-Reset]</code>	SW_RESET then stop node afterward
<code>{@addr=value}*</code>	Patches Flash addr or Manuf offset with value
<code>addr</code>	Flash word address or (after <code>-Mfg</code> ) Manuf word offset
<code>value</code>	Numeric word (decimal or 0x hex), or quoted string.

## Example (.hex file)

```
$ ./em2xx_patch.exe -filetarget ./test/training-sm.hex -Mfg @5080=0xbbcc
```

This example changes the manufacturing token word located at address 5080 (which happens to be the very beginning of the customer manufacturing token space) in the hex file `./test/training-sm.hex` to value `0xbbcc`.

This example transaction would look like the following when executed:

### Command line input

```
$ ./em2xx_patch.exe -filetarget ./test/training-sm.hex -Mfg @5080=0xbbcc
```

### Command line output

```
em2xx_patch.exe Rev 1.10 b2 (Feb 27 2007 10:40:31)
Successfully created patched file ./test/training-sm.hex in the Intel Hex
format.
```

## Example (token)

```
$ ./em2xx_patch.exe -sid 192.168.223.206 -Run -Mfg
-Override @5080=0xbbcc
```

This example changes the manufacturing token word located at address 5080 (which happens to be the very beginning of the customer manufacturing token space) to value 0xbbcc. Note that because the token is being changed from an existing value, it is necessary to use the `Override` switch.

This example transaction would look like the following when executed:

### Command line input

```
$ ./em2xx_patch.exe -sid192.168.223.206 -Run -Mfg -Override @5080=0xbbcc
```

### Command line output

```
em2xx_patch.exe Rev 1.10 b2 (Feb 28 2007 16:56:24)
SIF Slave id already set to 1 (pod 1)
XAP2= DbgEna performed
XAP2= Stop performed
XAP2= DbgEna performed
XAP2= Reset performed
Programming FLASH...
XAP2= Run performed
MFGcu 0040-0040 (0001)
Total Manuf Used=0001 words
XAP2= Stop performed
Verifying FLASH...

****Verification: SUCCESS****
XAP2= Stop performed
XAP2= Reset performed
XAP2= Issuing SW_RESET
```

## Example (EUI64)

```
$ ./em2xx_patch.exe -sid 192.168.223.204 -Run -Mfg
-Override @41=0x000D @42=0x6F00 @42=0x000F @43=0xAE69
```

This example changes the EUI64 token words. From the token table you can confirm the start of the first of four word sized tokens, typically located at word offset 0x0041. In this example the EUI64 will be changed to the hex value `000D6F0000FAE69`. Note that because the token is being changed from an existing value, it is necessary to use the `-Override` switch.

**Note:** To be safe, it is a good idea to perform a read of the token locations before and after the patch to ensure that the process worked as intended.

This example transaction would look like the following when executed.

### Command line input

```
$ ./em2xx_patch.exe -sid 192.168.223.204 -Mfg -Override @41=0x000D @42=0x6F00
@43=0x000F @44=0xAE69
```

## Command line output

```

...em2xx_patch.exe Version 2.0 Build 4 Sif.dll Version 4.12 Build 0 SIF Slave id
already set to 1 (pod 1) XAP2= DbgEna performed XAP2= Stop performed XAP2=
DbgEna performed XAP2= Reset performed Programming FLASH...
XAP2= Run performed
MFGcu 0041-0044 (0004)
Total Manuf Used=0004 words
XAP2= Stop performed
Verifying FLASH...

****Verification: SUCCESS****
XAP2= Stop performed
XAP2= Reset performed
XAP is stopped, ready to debug ...

```

You can verify the patch using the em2xx\_read.exe utility:

## Command line input

```
$ ./em2xx_read.exe -sid192.168.223.206 -Mfg 41+4
```

## Command line output

```

...em2xx_read.exe Version 2.0 Build 4 Sif.dll Version 4.12 Build 0 SIF Slave id
already set to 1 (pod 1) XAP2= DbgEna performed XAP2= Stop performed XAP2=
DbgEna performed XAP2= Reset performed
Manuf:
###--> addrHi=44 addrLo=41
#0040:      .000D.6F00.000F.AE69.      .      .      .i.....
XAP2= Stop performed
XAP2= Reset performed
XAP is stopped, ready to debug ...

```

## Scripting

Although the em2xx\_patch utility is designed to function as a stand-alone 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.

# 6 Error Messages

## Introduction

All of the `em2xx` utilities have been designed to use plain language error messages when needed. Most of these error messages are self-explanatory, but some errors have special meaning that will be described in this chapter.

## Syntax Error

It is a common problem that typing command line instructions can sometimes introduce errors that create a syntax problem. Whenever a syntax error is detected, the utility will terminate with an error message and print out the Help instructions as a reminder of the expected command syntax.

## Example

In this example, the command line has been typed incorrectly. The `sid` value has been specified as "foo" a hostname that does not resolve.

## Command line input

```
$ ./em2xx_read.exe -sid foo -outfile boo.hex
```

## Command line output

```
em2xx_read.exe Version 2.0 Build 5
Sif.dll Version 4.12 Build 0
Hostname "foo" lookup failed: Unknown error
Invalid -sid<slaveid> option
Usage: em2xx_read.exe [-options] {startAddr[+size|-endAddr]}*
Note: All -options are case sensitive.
[-sid<slaveid>] Specify SIF Slave id # or IP address/hostname
[-id<podid>] Specify SIF Pod id # or IP address/hostname
[-Mfg] Subsequent addresses refer to manufacturing token
area word offsets
[-excludeMfg] Ignores manufacturing tokens when
writing unloaded data to a .hex file. By default
customer area manufacturing tokens are included in any
.hex file that this application creates.
[-includeSIMEE] Includes SIMEE portion of data
(addresses 0xF000 - 0xFFFF) when
writing to a .hex file.
[-Run] Start em2xx node via SW_RESET afterward
[-Reset] SW_RESET then stop em2xx node afterward
{startAddr[+size|-endAddr]}* Flash/Manuf range(s) to unload
[-outfile <filename>] Writes entire address space
out to a file of type provided.
Currently only .hex is supported.
```

## Instructional Error

Some common errors result in an instructional message enclosed by a marquee. The instructional information is specific to the committed error.

### Example 1

In this example, the user has forgotten to specify the *-Override* argument when programming manufacturing tokens.

#### Command line input

```
$ ./em2xx_patch.exe -id 7 -Run -Mfg @40=AAAA
```

#### Command line output

```
$ ./em2xx_patch.exe -sid 192.168.223.206 -Run -Mfg @40=AAAA
em2xx_patch.exe Version 2.0 Build 5
Sif.dll Version 4.12 Build 0
SIF Slave id already set to 1 (pod 1)
  XAP2= DbgEna performed
  XAP2= Stop performed
  XAP2= DbgEna performed
  XAP2= Reset performed
Programming FLASH...
  XAP2= Run performed
FlashLoader addr=0x0040 len=0x2 error 0x2
  XAP2= Stop performed
Verifying FLASH...
!0040: 6666 != 4141
!0041: ffff != 4141
Failed to verify manufacturing flash area.

****Application load: FAILURE****
****Verification:      FAILURE****

*****
  Problem encountered programming the chip. Remember
  to use the -Override flag to program the
  manufacturing area of flash more than once.
*****

  XAP2= Stop performed
  XAP2= Reset performed
  XAP2= Issuing SW_RESET
```

### Example 2

In this example, `em2xx_convert` is being used to create a hex file. However, the `.xdv` file does not start at index 0.

#### Command line input

```
$ ./em2xx_convert.exe -outfile boo.hex ./test/training.xdv
```

## Command line output

```
em2xx_convert.exe Version 2.0 Build 5 Loading code file ./test/training.xpv...
// ROM START ADDRESS 0
// WORDSIZE 16
Loading data file ./test/training.xdv...
// ROM START ADDRESS 0
// WORDSIZE 8
RAM Usage:
  Available:      2314
  Stack requires: 1660
  Remaining:      654
*****
WARNING: Generated Hex image does not start at address 0
        The file: boo.hex
        may not be usable. Use -btl <filename> to
        include a bootloader with this file.
*****
Successfully created file boo.hex in the Intel Hex format.
```

---

## Index

EM2USBLoad.....	2
EM2USBPatch.....	2
EM2USBRead .....	2
em2xx_convert .....	2, 11
em2xx_load .....	5
EmberZNet .....	1
Error Messages .....	17
EUI64 .....	15
file formats.....	
summary .....	4
hex file.....	
creating .....	18
Hex File format .....	2
Instructional Error Messages.....	18
Intel Hex format .....	14
patch EUI64 .....	15
Scripting .....	7, 10, 12, 16
SIF interface .....	5, 13
syntax error .....	17
technical support.....	i
token .....	14
EUI64 .....	15
utilities.....	
em2xx_convert .....	11
em2xx_load .....	5
em2xx_patch.....	13
em2xx_read .....	8
xdv.....	2
xpv.....	2
ZigBee.....	1