



# Bringing Up Custom Devices

## For the EM35x SoC Platform

Ember EM35x chips are delivered to customers with only the Fixed Information Block (FIB) programmed. The FIB contains a serial-link-only monitor, chip identifiers, Ember EUI64, and calibration values. The FIB cannot be modified. Before the EM35x chips can be used to run EmberZNet PRO applications, the customer or a contract manufacturer/test house must prepare them. Preparation includes programming the proper application and bootloader, if necessary, into the Main Flash Block (MFB), as well as programming customer manufacturing tokens in the Customer Information Block (CIB).

This application note describes how to initialize a piece of custom hardware (a “device”) based on the EM35x so that it interfaces correctly with the EmberZNet PRO network stack. The same procedures can be used to restore Ember Development Kit devices whose settings have been corrupted or erased.

Even though the EM35x SoC flash is fully tested during production test, the flash contents in the MFB are not set to a known state prior to shipment. The CIB will be left erased with read protection disabled (read protection is an option byte and part of the CIB manufacturing tokens).

### New in this Revision

Terminology correction made in document summary.

### Contents

Setting CIB Manufacturing Tokens .....	2
Bootloaders .....	5
Running the Nodetest Application .....	6
Uploading and running nodetest .....	6
nodetest commands.....	7
Performing Functional Testing .....	7
Setting Stack Tokens.....	8



## Setting CIB Manufacturing Tokens

CIB manufacturing tokens are values programmed into a special, non-volatile storage area of flash known as the CIB. The CIB contains data that manufacturers of EM35x-based devices can program. The Fixed Information Block (FIB) also contains manufacturing tokens, but these tokens are fixed values that cannot be modified.

**Note:** Applications and the stack can read any manufacturing tokens at any time.

Table 1 identifies the CIB manufacturing tokens for EmberZNet PRO that OEMs and CMs may want to program at manufacturing time. Refer to `\hal\micro\cortexm3\token-manufacturing.h` for the token definition, because it may differ from Table 1 below depending on the stack release version.

Use the `em3xx_load.exe` utility to set manufacturing tokens. `em3xx_load.exe` can be found in the ISA3 Utilities install directory. The default is `C:\Program Files\Ember\ISA3 Utilities\bin\em3xx_load.exe`. `em3xx_load.exe` has many commands, but four commands are specifically designed for manipulating CIB manufacturing tokens.

Command line entry	Description
<code>--cibtokenspatch &lt;file&gt;</code>	Patch the CIB tokens with the token set defined in the specified file. See <code>--cibtokenspatch-help</code> for additional help on this command.
<code>--cibtokensprint</code>	Print the contents of the CIB tokens from the target in a very easy to read form.
<code>--cibtokensdump</code>	Print the contents of the CIB tokens from the target in a format that can easily be imported with using <code>--cibtokenspatch</code> .
<code>--cibtokenspatch-help</code>	Print out additional help about how to set the tokens using <code>--cibtokenspatch</code> , including detailed file syntax and a list of all supported tokens.

Executing `em3xx_load.exe --help` causes `em3xx_load.exe` to print its online help menu detailing all commands, modifiers, and arguments. In addition, Ember document 120-4032-000, *EM3xx Utilities Guide*, provides many examples detailing how the `em3xx_load.exe` utility can be used. This document also includes examples showing how to print and patch CIB manufacturing tokens.

Ember recommends that CIB manufacturing tokens be written with an external utility (`em3xx_load.exe`) at the same time as programming the MFB. Using an external utility allows for patching and reprogramming the CIB as many times as necessary. Some situations though, may require that a CIB manufacturing token be programmed at runtime from code running on the chip itself. The manufacturing token module of the HAL provides a token API to write CIB manufacturing tokens. However, this API only writes CIB tokens that are in a completely erased state. If a CIB token must be reprogrammed, you must use an external utility. The API is `halCommonSetMfgToken( token, data )`. The parameters for this API are the same as the API `halCommonSetToken( token, data )`.

**Note:** CIB manufacturing tokens written with `halCommonSetMfgToken()` must be located on a 16-bit address boundary and the byte length must be a multiple of 16 bits.

Table 1. CIB manufacturing tokens for the EM35x

Address	Size (Bytes)	Name	Description
0x08040800	16	TOKEN_MFG_CIB_OBS	<p>Dedicated special flash storage called option bytes. Option bytes are special storage because they are directly linked to hardware functionality of the chip. There are 8 option bytes, each occupying 16 bits of flash as follows:</p> <ul style="list-style-type: none"> <li>Option Byte 0: Configures flash read protection</li> <li>Option Byte 1: Reserved</li> <li>Option Byte 2: Available for customer use</li> <li>Option Byte 3: Available for customer use</li> <li>Option Byte 4: Configures flash write protection</li> <li>Option Byte 5: Configures flash write protection</li> <li>Option Byte 6: Configures flash write protection</li> <li>Option Byte 7: Reserved</li> </ul> <p>Refer to the <i>EM35x Datasheet</i> (120-035X-000) for a detailed description of option bytes, what they mean, how they work, and what values should be used.</p> <p><i>Usage:</i> All option bytes must be set to a valid state.</p>
0x08040810	2	TOKEN_MFG_CUSTOM_VERSION	<p>Version number to signify which revision of CIB manufacturing tokens you are using. This value should match <code>CURRENT_MFG_CUSTOM_VERSION</code> which is currently set to <code>0x01FE</code>. <code>CURRENT_MFG_CUSTOM_VERSION</code> is defined in <code>\hal\micro\cortexm3\token-manufacturing.h</code>.</p> <p><i>Usage:</i> Recommended for all devices using CIB manufacturing tokens.</p>
0x08040812	8	TOKEN_MFG_CUSTOM_EUI_64	<p>IEEE 64-bit address, unique for each radio. Entered and stored in little-endian. Setting a value here overrides the pre-programmed Ember EUI64 stored in the FIB (<code>TOKEN_MFG_EMBER_EUI_64</code>). This is for customers who have purchased their own address block from IEEE.</p> <p><i>Usage:</i> Optionally set by device manufacturer if using custom EUI64 address block.</p>
0x0804081A	16	TOKEN_MFG_STRING	<p>Optional device-specific string, for example, the serial number.</p> <p><i>Usage:</i> Optionally set by device manufacturer to identify device.</p>
0x0804082A	16	TOKEN_MFG_BOARD_NAME	<p>Optional string identifying the board name or hardware model.</p> <p><i>Usage:</i> Optionally set by device manufacturer to identify device.</p>
0x0804083A	2	TOKEN_MFG_MANUF_ID	<p>16-bit ID denoting the manufacturer of this device. Ember recommends setting this value to match your ZigBee-assigned manufacturer code, such as in the stack's <code>emberSetManufacturerCode()</code> API call.</p> <p><i>Usage:</i> Recommended for devices utilizing the Ember standalone bootloader.</p>

Address	Size (Bytes)	Name	Description												
0x0804083C	2	TOKEN_MFG_PHY_CONFIG	<p>Default configuration of the radio for power mode (boost/normal), transmit path selection (bi-directional/alternate), and boost transmit power level:</p> <ul style="list-style-type: none"> <li>Bit 0 (lsb): Set to 0 for boost mode; set to 1 for non-boost (normal) mode. Boost mode mainly increases rx sensitivity but also increases tx output power by a small amount. Larger increases in tx output power are available in bits 3-7 of this token.</li> <li>Bit 1: Set to 0 if an external PA is connected to the alternate TX path (RF_TX_ALT_P and RF_TX_ALT_N pins); set to 1 otherwise.</li> <li>Bit 2: Set to 0 if an external PA is connected to the bi-directional RF path (RF_P and RF_N pins); set to 1 otherwise.</li> <li>Bits 3-7: Set to the boost tx power offset from the first integer value above nominal maximum tx power (+3dBm).</li> </ul> <table border="1"> <thead> <tr> <th>For tx power of...</th> <th>Set this subfield to...</th> </tr> </thead> <tbody> <tr> <td>+4dBm</td> <td>0</td> </tr> <tr> <td>+5dBm</td> <td>1</td> </tr> <tr> <td>+6dBm</td> <td>2</td> </tr> <tr> <td>+7dBm</td> <td>3</td> </tr> <tr> <td>+8dBm</td> <td>4</td> </tr> </tbody> </table> <p>Bit 7 is most significant; Bit 3 is least significant. All bits must be set to 1 if tx boost power level is not desired.</p> <ul style="list-style-type: none"> <li>Bits 8–15: Reserved. Must be set to 1 (the erased state).</li> </ul> <p><i>Usage:</i> Required for devices that utilize boost mode or an external PA circuit.</p>	For tx power of...	Set this subfield to...	+4dBm	0	+5dBm	1	+6dBm	2	+7dBm	3	+8dBm	4
For tx power of...	Set this subfield to...														
+4dBm	0														
+5dBm	1														
+6dBm	2														
+7dBm	3														
+8dBm	4														
0x0804083E	16	TOKEN_MFG_BOOTLOAD_AES_KEY	<p>Sets the AES key used by the Ember bootloader utility to authenticate bootloader launch requests.</p> <p><i>Usage:</i> Required for devices that utilize the standalone bootloader.</p>												
0x0804084E	8	TOKEN_MFG_EZSP_STORAGE	<p>An 8-byte, general-purpose token that can be set at manufacturing time and read by the host microcontroller via EZSP's <code>getMfgToken</code> command frame.</p> <p><i>Usage:</i> Not required. Device manufacturer may populate or leave empty as desired.</p>												
0x0804087E	92	TOKEN_MFG_CBKE_DATA	<p>Defines the security data necessary for Smart Energy devices. It is used for Certificate Based Key Exchange to authenticate a device on a Smart Energy network. The first 48 bytes are the device's implicit certificate, the next 22 bytes are the Root Certificate Authority's Public Key, the next 21 bytes are the device's private key (the other half of the public/private key pair stored in the certificate), and the last byte is a flags field. The flags field should be set to 0x00 to indicate that the security data is initialized.</p> <p><i>Usage:</i> Required by Smart Energy Profile certified devices.</p>												

Address	Size (Bytes)	Name	Description
0x080408DA	20	TOKEN_MFG_INSTALLATION_CODE	<p>Defines the installation code for Smart Energy devices. The installation code is used to create a pre-configured link key for initially joining a Smart Energy network. The first 2 bytes are a flags field, the next 16 bytes are the installation code, and the last 2 bytes are a CRC.</p> <p>Valid installation code sizes are 6, 8, 12, or 16 bytes in length. All unused bytes should be 0xFF. The flags field should be set as follows depending on the size of the install code:</p> <ul style="list-style-type: none"> <li>6 bytes = 0x0000</li> <li>8 bytes = 0x0002</li> <li>12 bytes = 0x0004</li> <li>16 bytes = 0x0006</li> </ul> <p><i>Usage:</i> Required by Smart Energy Profile certified devices.</p>
0x080408EE	2	TOKEN_MFG_OSC24M_BIAS_TRIM	<p>A relative amount (between 0x0000 and 0x000F) that trims the 24 MHz crystal bias drive. A lower value reduces drive and current consumption, but increases instability. Although a value can be set here manually, Ember recommends leaving this value unpopulated (0xFFFF) so that the stack can perform auto-calibration at runtime to determine the optimal value for lowest current consumption while maintaining stability.</p> <p><i>Usage:</i> Not recommended (leave erased, 0xFFFF).</p>
0x080408F0	2	TOKEN_MFG_SYNTH_FREQ_OFFSET	<p>An offset applied to the radio synthesizer frequency. This offset can be used to compensate for variations in 24 MHz crystals to accurately center IEEE-802.15.4 channels.</p> <ul style="list-style-type: none"> <li>Bits 0-7: Set to the signed offset to be applied to the synthesizer frequency. Each step provides approximately 11 kHz of adjustment.</li> <li>Bit 8: Set to 0 if the offset is valid and should be used. Set to 1 (the erased state) if the token is invalid or has not been set.</li> <li>Bits 9-15: Reserved. Must be set to 1 (the erased state).</li> </ul>
0x080408F2	2	TOKEN_MFG_OSC24M_SETTLE_DELAY	<p>The delay in microseconds to allow the 24 MHz crystal to settle after a bias change when waking up from deep sleep. Ember recommends leaving this value unpopulated (0xFFFF) so that the stack uses its default conservative delay value.</p>

For more information on the Smart Energy tokens, see Ember document 120-5058-000, *Setting Manufacturing Certificates and Installation Codes*.

## Bootloaders

For the EM35x, Ember provides two bootloader options:

- Application bootloader: Supports multi-hop bootloading from a gateway device. Enables an application to continue running and sending normal application packets while the new firmware image is being received.
- Stand-alone bootloader: Supports only one-hop bootloading from a gateway device. If some nodes are multiple hops away from a gateway, they must either be brought closer

to the gateway for bootloading or you must create a portable device that is taken around the installation to bootload all of the nodes.

For a discussion on what the bootloaders are, their differences, their uses, and how to use the bootloaders, refer to Ember document 120-3029-000, *Ember Application Development Fundamentals*.

For a discussion on how to program applications and bootloaders to chips as well as convert applications to the bootloader compatible .ebi file format, refer to Ember document 120-4032-000, *EM3xx Utilities Guide*.

## Running the Nodetest Application

The nodetest application is included in the EmberZNet PRO stack installation directory, in the /app/nodetest subdirectory. This directory contains only .s37 file images that can only be installed onto a chip via the normal em3xx\_load.exe procedure over an InSight Adapter (ISA3).

### Uploading and running nodetest

nodetest can be installed using either InSight Desktop (ISD) to upload the application or with em3xx\_load.exe from a command line. The following is an example of loading nodetest from a command line:

```
$ em3xx_load.exe --ip myisahostname ./app/nodetest/nodetest-em357.s37
```

For a discussion on how to program chips using em3xx\_load.exe from a command line, refer to Ember document 120-4032-000, *EM3xx Utilities Guide*.

Once the upload completes, you can interact with nodetest via the hardware UART or the Virtual UART. Using the Virtual UART requires telnetting to an ISA3, port 4900, with the InSight Port debug cable connected to the chip. You can use the hardware UART in either of these methods:

- Using a direct UART connection (the Breakout Board instantiates an RS-232 connector)
- Using the passthrough UART mode and telnetting to the ISA3, port 4901, connected to a Breakout Board using the DEI Port.

Press **Enter** in the telnet window or com port window to start the nodetest application.

**Note:** If using the hardware UART without the ISA3, you must configure your comm port program to 115,200 bps, no parity bits, 1 stop bit. If using the hardware UART via the passthrough mode of the ISA3, port 4901, configure the ISA3 to 115,200 bps by executing the command `ember> port 1 115200`. Execute the ISA3 configuration command by telnetting to the ISA3 administration interface at TCP port 4902.

**Note:** Every time nodetest resets, it will not automatically print any characters. Instead, nodetest will listen on both the hardware UART and the Virtual UART looking for a Return key. Whichever port nodetest finds a Return key on will become the port that nodetest uses for all serial interaction until nodetest resets again.

### nodetest commands

A Return key initiates the nodetest application on reset. This displays the power-up prompt, ending in the > (greater than) symbol. Pressing the Return key at the prompt will always cause another prompt to be displayed. Executing the **help** command causes nodetest to print a list of all available commands with a brief description of the commands. The commands are listed by functional modules.

## Performing Functional Testing

At this point, you may want to use the nodetest application to perform a simple send/receive test on the device to determine its range and generally test its radio functionality.

**Note:** When programming a device for test or retest, use the --erase option to ensure that all previous calibration data is erased. Ember recommends erasing the flash contents of a device prior to testing to ensure the calibration is executed at the time the device is tested.

1. Connect a device known to be in good operating order either to your computer or to a different computer via a serial port.
2. Upload nodetest to the known good device and then run it.
3. Make sure that nodetest is still running on the new device (you should see the > prompt).
4. Set both devices to a channel by typing `setchannel X`, where *X* is the channel in hex.
5. Optional: Set a power level on the test device by typing `settxpower X`, where *X* is the power level in hex.
6. On the known good device, type `rx`, which sets the device to receive and display statistics for each packet received. Type `e` to exit.
7. On the test device, type `tx X` to transmit *X* packets where *X* is in hex. (Note that `tx 0` sends infinite packets.) Type `e` to exit.
8. Reverse this procedure to test receiving on the test device.

**Note:** The fourth column in the display output, labeled “per”, shows the packet error rate. For this value to be accurate, the two devices being used must be configured per “Uploading and running nodetest” and the receiver should not hear any other devices. Exiting the test and restarting clears the values and reset the values being displayed.

**Note:** nodetest attempts to print packet data as fast as it can, but it is possible to receive packets faster than nodetest can print. Therefore, there may be gaps in the printed packets.

If the new device fails to successfully transmit or receive packets with the known good device, you may want to attach the new device to a signal generator or network analyzer to verify that generated packets on the target frequency can be received and that the new device can transmit accurately at the center frequency of the selected channel. Other tests may be required for FCC or CE compliance testing.

## Setting Stack Tokens

The EmberZNet PRO stack maintains several non-volatile settings (tokens) used for network operation. Additionally, certain applications may require the use of their own tokens for non-volatile data storage.

**Note:** nodetest cannot be rebuilt with application tokens.

To use the nodetest application for programming stack token values:

1. Install the nodetest application on the device.
2. Using a serial or telnet connection with port speed set to 115200 bps, press **Enter** to initiate the nodetest application.
3. Use the `tokmap` command to output the map of the token storage area.
4. Use the `loadtoks` command to initialize all stack tokens to their default values.
5. Use the `tokdump` command to confirm the values the tokens.
6. Use the `tokread` and `tokwrite` commands to view and manipulate individual tokens.

**Note:** For more information about tokens and the Simulated EEPROM, refer to the *token.h File Reference* in the *EmberZNet API Reference* for your Ember chip and Ember document 120-5037-000, *Using the Simulated EEPROM*.

## After reading this document

If you have questions or require assistance with the procedures described in this document, contact Ember Customer Support. The Ember Customer Support portal provides a wide array of hardware and software documentation such as FAQs, reference designs, user guides, application notes, and the latest software available to download. To obtain support on all Ember products and to gain access to the Ember Customer Support portal, visit [http://www.ember.com/support\\_index.html](http://www.ember.com/support_index.html).

Copyright © 2006-2011 Ember Corporation

All rights reserved. Neither this publication nor any part thereof can be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without prior written approval of Ember Corporation. This documentation is furnished under license and can be used or copied only in accordance with the terms of such license.

The content of this documentation is furnished for informational use only, is subject to change without notice, and does not represent a commitment or guaranty by Ember Corporation. The statements, configurations, technical data, and recommendations in this document are believed to be accurate and reliable as of the time of publication, but Ember Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this documentation. DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT, ARE DISCLAIMED. Users are responsible for their applications and for the use of any products specified in this document.

Title, ownership, and all rights in copyrights, patents, trademarks, and other intellectual property rights embodied in Ember Corporation's Products and any copy, portion, or modification thereof, shall not transfer to Purchaser or its customers and shall remain in Ember Corporation and its licensors.

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

Ember is a trademark of Ember Corporation. All other trademarks are the property of their respective holders.

