



Bringing Up Custom Devices

For the EM250 SoC Platform

Ember EM250 chips are delivered to customers with only a minimal amount of chip identification data programmed into their embedded flash contents. Before these chips can be used to run EmberZNet PRO applications, they must be prepared by the customer or by a contract manufacturer/test house.

This application note describes how to initialize a piece of custom hardware (a “device”) based on the EM250 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 EM250 SoC flash is fully tested during production test, the flash contents are not set to a known state before shipment.

New in this Revision

Table 1: Added MFG_SYNTH_FREQ_OFFSET.

All references to rangetest and corresponding command set updated to reflect nodetest.

Contents

Setting Manufacturing Tokens.....	3
Choosing a Bootloader.....	7
Application bootloader	7
Stand-alone bootloader	7
Installing the Bootloader.....	8
Hardware and software setup	8
Installing the bootloader image via the em2xx_load utility	9
Installing the bootloader image via InSight Desktop.....	10
Bootloading and Running the Nodetest Application	12
Uploading and running nodetest by Ethernet	13
Uploading and running nodetest via serial connections	13
nodetest commands.....	14
Performing Functional Testing	15
Setting Stack and Application Tokens.....	16
Bootloading the Final Application	17



Setting Manufacturing Tokens

Manufacturing tokens are values programmed into a special, non-volatile storage area of flash known as the Flash Information Area (FIA). The FIA contains data programmed during manufacturing of the EM250 chip itself, as well as data that can be programmed by manufacturers of EM250-based devices.

Warning: Unlike stack and application tokens, which reside in the simulated EEPROM section of the chip and can be set using the process detailed in the “Setting Stack and Application Tokens” section of this document, data in the FIA (including these tokens) are subject to the following restrictions:

- They cannot be written by code running on the chip itself (no writes while executing), although applications and the stack can read these tokens at runtime.
- Each value can only be written *once* over the lifetime of the chip.

Table 1 identifies the manufacturing tokens for EmberZNet PRO that OEMs and CMs may want to program at manufacturing time. Refer to `\hal\micro\xap2b\token-manufacturing.h` for the token definition, as it may differ from Table 1 below depending on the stack release version.

Use the `em2xx_patch.exe` utility (found in the same directory as `em2xx_load.exe` as part of the SIF Toolkit) to set the manufacturing tokens by issuing the `em2xx_patch` command with the target device’s connected InSight Adapter hostname or IP address. Follow this command with `-sid`, the hostname, the `-Mfg` argument followed by the @ character and a word offset (as shown in Table 1), the equal sign (=), and the value to be stored.

For example, the following command sets the `MFG_BOARD_NAME` token to the string `dev0455` on an EM250 connected to InSight Adapter with the hostname `ember00`:

```
em2xx_patch -sid ember00 -Mfg @004D="dev0470"
```

For tokens with numeric values, such as the crystal bias trim, specify the value as either a decimal number or with a `0x` prefix to indicate hexadecimal numbers. Leading zeroes can be optionally omitted. For example:

```
em2xx_patch -sid ember00 -Mfg @0055=0xF
```

Remember that `em2xx_patch` can only write a value to the manufacturing tokens once over the lifetime of that device, so be careful when programming these values.

Table 1. User-programmable EmberZNet PRO manufacturing tokens for the EM250

Word Offset	Size (Bytes)	Name	Description
0x0040	2	MFG_CUSTOM_VERSION	An optional version number to signify which revision of your hardware design the product uses. <i>Usage:</i> Optionally set by device manufacturer to identify device.

Word Offset	Size (Bytes)	Name	Description
0x0041	8	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 EUI64 from Ember's address block (MFG_EMBER_EUI_64). 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>
0x0045	16	MFG_STRING	<p>An optional device-specific string, for example, the serial number.</p> <p><i>Usage:</i> Optionally set by device manufacturer to identify device.</p>
0x004D	16	MFG_BOARD_NAME	<p>An optional string identifying the board name or hardware model.</p> <p><i>Usage:</i> Optionally set by device manufacturer to identify device.</p>
0x0055	2	MFG_OSC24M_BIAS_TRIM	<p>A relative amount (between 0x0 and 0xF) that the 24 MHz crystal bias should be trimmed. As this value increases, the drive level to the crystal increases, changing the stability of the 24 MHz crystal. Although a value can be set here manually, Ember recommends leaving this value unpopulated so that the stack can perform auto-calibration at runtime to determine the optional value for this trim.</p> <p><i>Usage:</i> Not recommended for devices using EmberZNet 2.5 or later. Optionally set by device manufacturer to override stack's automatic calibration value.</p>
0x0056	2	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 stand-alone bootloader.</p>

Word Offset	Size (Bytes)	Name	Description
0x0057	2	MFG_PHY_CONFIG	<p>The default configuration of the radio for power mode (boost/normal) and power amp (PA) selection (internal/external):</p> <p>Bit 0 (lsb): Set to 0 for Boost mode; set to 1 for non-Boost (normal) mode.</p> <p>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.</p> <p>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.</p> <p>Bits 3-7: Reserved. Must be set to 1 (the erased state).</p> <p>Bits 8-15 (msb): Leave all bits set to 1 unless instructed to do otherwise by Ember support. This byte serves as an override to the TX_STEP_TIME register when either Bit 1 or Bit 2 is set to 0.</p> <p><i>Usage:</i> Required for devices that utilize Boost mode or an external PA circuit.</p>
0x0058	16	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>
0x0060	8	MFG_EZSP_STORAGE	<p>A collection of an 8-byte, general-purpose token that can be set at manufacturing time and read by the host microcontroller through EZSP's <code>getMfgToken</code> command frame.</p> <p><i>Usage:</i> Not required. Device manufacturer may populate or leave empty as desired.</p>
0x0080	40	MFG_ASH_CONFIG	<p>Configuration token for EZSP UART only. It contains several parameters used by the ASH protocol, such as baud rate, various timings, and so on.</p> <p><i>Usage:</i> Required for EZSP UART only.</p>

Word Offset	Size (Bytes)	Name	Description
0x0094	92	MFG_CBKE_DATA	<p>This token 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>
0x00C2	20	MFG_INSTALLATION_CODE	<p>This token 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"> • 6 bytes = 0x0000 • 8 bytes = 0x0002 • 12 bytes = 0x0004 • 16 bytes = 0x0006. <p><i>Usage:</i> Required by Smart Energy Profile certified devices.</p>

Word Offset	Size (Bytes)	Name	Description
0x00CC	2	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"> • Bits 0-7: Set to the signed offset to be applied to the synthesizer frequency. Each step provides approximately 11 kHz of adjustment. • 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. • Bits 9-15: Reserved. Must be set to 1 (the erased state).

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

Choosing a Bootloader

For the EM250, Ember provides two bootloader options:

- Application bootloader
- Stand-alone bootloader

Each bootloader option is described next.

Application bootloader

The application bootloader supports multi-hop bootloading from a gateway device, and it enables an application to continue running and sending normal application packets while the new firmware image is being received. Because it stores the bootloaded image at the same time that the current image is running, this bootloader requires twice the amount of flash memory on a node. It is important to note that the packets sent to bootload a node are ZigBee compliant, just like all the packets sent from the application. The application bootloader is only 5 kB large; it only supports flash operations and cannot drive the radio. 128 kB of off-board download space is also required. An I²C serial EEPROM can be used for this purpose. Refer to Ember document 120-3029-000, *Ember Application Development Fundamentals*, for more information on the use of I²C serial EEPROM for the application bootloader.

Stand-alone bootloader

The 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.

The application does not run on the node while it is being bootloaded. The packets sent to perform the bootload are not ZigBee compliant. This bootloader requires enough flash memory to support just one application in addition to the 10kB size of the bootloader itself. This bootloader is much larger than the application bootloader because it needs to support all the flash operations as well as full radio control, including the PHY and MAC layers. This radio support is one hop only, as it cannot fit a full transport layer.

For more information, see Ember document 120-3029-000, *Ember Application Development Fundamentals*.

Installing the Bootloader

Installing the bootloader requires a SIF programming device, such as the InSight Adapter or InSight USB Link, and access to the EM250's SIF programming interface. The SIF programming interface for the chip is often exposed through the 10-pin programming/debug header known as the InSight Port. For details about the InSight Port, refer to Ember document 120-2001-000, *EM250 Radio Communications Module*, and Ember document 120-5026-000, *PCB Design with an EM250*. For details about creating a SIF programming device or purchasing InSight Adapters, contact Ember Customer Support at http://www.ember.com/support_index.html. Note that the following instructions assume that Ember's InSight Adapter hardware is being used as the SIF programming device.

Bootloading can be done through a command line tool (em2xx_load.exe) or a graphical user interface (InSight Desktop). Both methods are described below.

Hardware and software setup

1. If needed, install the device on a breakout board or attach it to a stable power supply. If a special interface connector is required to access the InSight Port header on your custom device, attach that now.
2. Install the SIF Toolkit, which allows access to the SIF programming interface via Ethernet connection to an InSight Adapter. The SIF Toolkit can be installed in either of the following ways:
 - By selecting the option to install the "SIF Toolkit" component during the installation of Ember's xIDE for EM250 software. (Ember recommends using the latest xIDE for the EM250 installer for best results.)
 - By running an EmberZNet PRO stack installer for a release on the EM260 platform. Because the EM260 requires the SIF Toolkit for programming but does not include the xIDE for EM250 software, the toolkit DLLs are part of the EmberZNet PRO stack installer for this platform.
3. Attach the InSight Adapter to the EM250 device. Plug one end of the 10-pin ribbon cable that came with the InSight Adapter into the "Debug Port" of the adapter and the other end into the InSight Port header of the EM250 device.

Note: When attaching the programming/debug cable to the InSight Port, make sure that pin 1 is nearest the red stripe on the ribbon connector.

4. If you plan to use Ember's InSight Desktop software to perform loading of firmware through a graphical interface, be sure that the latest version of this software is

installed. (Current released software versions are available through Ember Customer Support at http://www.ember.com/support_index.html.)

5. Be sure that you have the latest Ember bootloader firmware available. This firmware can be found as XPV and XDV files in the tool/bootloader/standalone-bootloader directory in EmberZNet stack releases 2.3 and later for the stand-alone bootloader for the EM250 platform, and tool/bootloader/app-bootloader directory in EmberZNet PRO stack releases 3.0 and later for the application bootloader. Note that this firmware is offered in a standard form or with the `-noLED` option. The `-noLED` file does not attempt to use the GPIOs for LED activity, which may be useful if your custom hardware uses these GPIOs for other peripherals.
6. Verify that the SIF Toolkit is properly installed on the machine on which you wish to perform the firmware loading. This can be done by entering `em2xx_load` at a command prompt with no arguments. (The directory containing this utility should already be in your path if the SIF Toolkit is properly installed, so you should not need to change to a specific directory before executing this command.) When this command is entered with no arguments, a version string similar to the one shown below is output:


```
em2xx_load.exe Rev 1.7 b4 (Jun 12 2006 09:37:59)
```
7. Follow the instructions in either the “Installing the bootloader image via the `em2xx_load` utility” or the “Installing the bootloader image via InSight Desktop” section below depending on whether you prefer to load the bootloader firmware using the `em2xx_load` command line tool or the InSight Desktop graphical interface.

Installing the bootloader image via the `em2xx_load` utility

1. Open a new command prompt (choose Start | Programs | Accessories | Command Prompt from the Windows Start menu).
2. Determine the IP address or hostname of the InSight Adapter attached to the EM250 device being programmed. (If your adapters appear in InSight Desktop’s “Adapters” view, you can expand the properties of each one to view its IP address.)
3. Execute the `em2xx_load` command, specifying the IP or hostname of the InSight Adapter attached to the target EM250 device prefaced by the `-sid` argument, and supply the bootloader firmware filename (either the XPV or the XDV filename is sufficient). For example, to load an EM250 with the stand-alone bootloader firmware located at `C:\EmberZNet\tool\bootloader\standalone-bootloader\standalone-bootloader-em250-noLED.xpv` via the InSight Adapter with IP address 192.168.0.1, execute the following command from the command prompt:

```
em2xx_load -sid 192.168.0.1 "C:\EmberZNet\tool\bootloader\standalone-
bootloader\standalone-bootloader-em250.xpv"
```

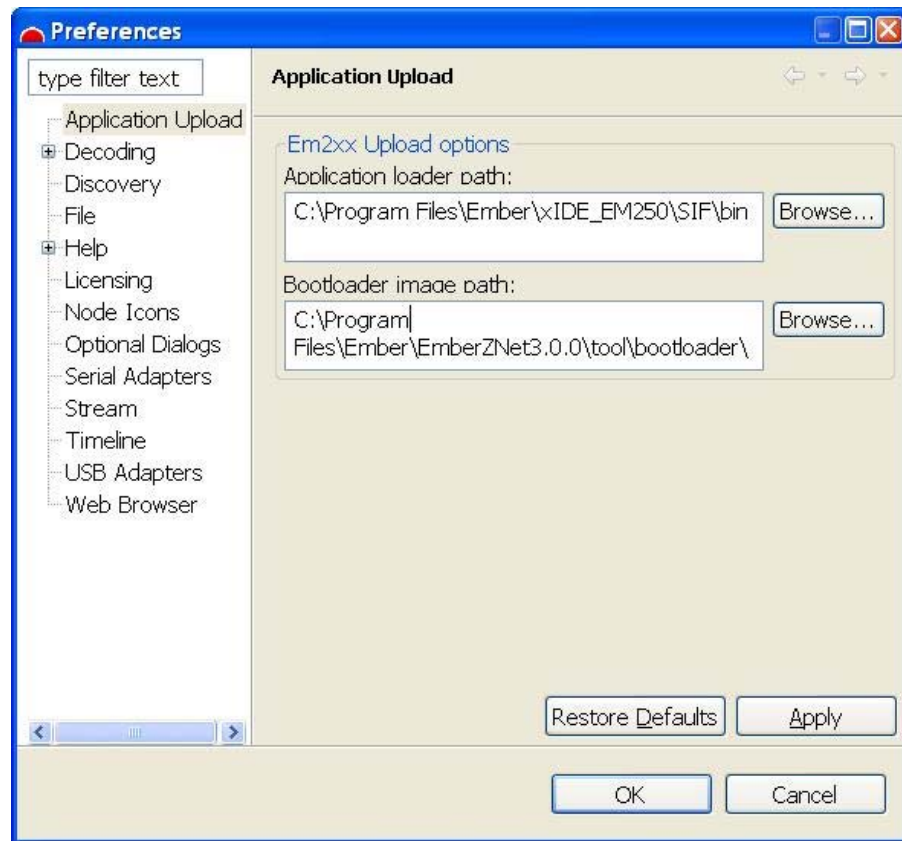
Note: You can optionally choose to load the application firmware at this time as well. To do this, supply the path to an application firmware file (in XPV or EBL file format) as an argument to the `em2xx_load` command, and add the `-bt1` argument just before specifying the bootloader firmware file. If you do so, the bootloader firmware will be loaded first, and then the target application firmware will be loaded subsequently as part of the process. For example, to load the bootloader firmware at the same time as target firmware called `myfile.ebl`, execute the following command:

```
em2xx_load -sid 192.168.0.1 myfile.ebl -bt1
"C:\EmberZNet\tool\bootloader\standalone-bootloader\standalone-
bootloader-em250.xpv"
```

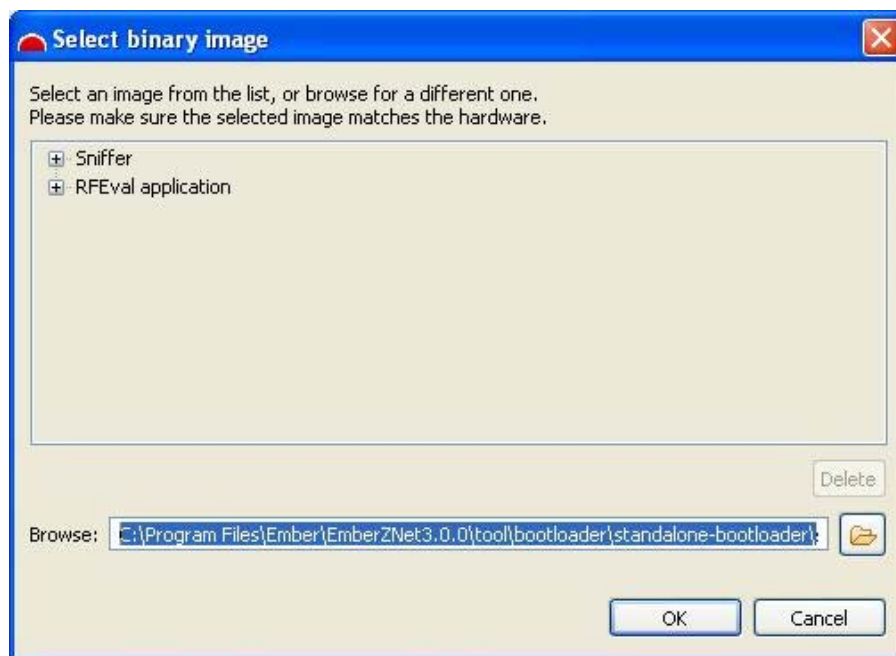
4. As the programming process progresses, you should see messages indicating the following steps of a successful firmware loading:
 - Which XPV is being bootloaded, for example: Loading code file
C:\EmberZNet\tool\bootloader\standalone-bootloader\standalone-bootloader-em250.xpv...
 - Which XDV file is being bootloaded, for example: Loading data file
C:\EmberZNet\tool\bootloader\standalone-bootloader\standalone-bootloader-em250.xpv...
 - That flash memory is being written: Programming FLASH...
 - That the written flash is being verified: Verifying FLASH...
 - That the verification succeeded: ****Verification: SUCCESS****
5. Once the flash has been successfully written and verified, you can optionally have the EM250 device begin executing the bootloader firmware (to perform additional uploads via serial or via the radio, using the default bootloader network parameters) by issuing the `em2xx_load` command with the `-Run` argument, for example, `em2xx_load -sid 192.168.0.1 -Run` for an InSight Adapter with the IP address 192.168.0.1. Refer to Ember document 120-3029-000, *Ember Application Development Fundamentals*, for details on how to interact with the Ember stand-alone bootloader through different mechanisms.

Installing the bootloader image via InSight Desktop

1. Start the InSight Desktop software and verify that the InSight Adapter used to program the target EM250 device is visible in the Adapters view. (If your target device's connected adapter is not shown in the Adapters view, you may need to reconfigure your network discovery properties in the **Discovery** section of the **File | Preferences** dialog.)
2. Right-click on the target device's connected adapter and select **Connect** from its context menu.
3. Verify that the **Application Upload** preferences are set correctly for your system configuration. You can view these properties on the **Application Upload** tab of the **File | Preferences** dialog in InSight Desktop. The **Application Uploader Path** setting should refer to the directory where `em2xx_load.exe` can be found (usually the `.../SIF/bin` directory in the folder where your SIF Toolkit files were installed). The **Bootloader Image Path** setting should refer to the XPV filename of your chosen Ember bootloader firmware (usually chosen from the `.../tool/bootloader/standalone-bootloader` subfolder of your current EmberZNet PRO stack release.) The **Always upload bootloader image** checkbox setting can remain unchecked unless you are replacing the bootloader firmware on a device, in which case it should be checked.



- Right-click on the target device's connected adapter in the Adapters view and select **Upload Application**. In the resulting dialog box, browse to the location of your bootloader firmware file (one of the XPV or XDV files from the .../tool/bootloader/standalone-bootloader subfolder of your current EmberZNet PRO stack) and select it, then click OK to proceed with the upload of the bootloader firmware.



Note: You can optionally choose to load the application firmware at this time as well by choosing an application firmware file (in either XPV or EBL file format) in this dialog. If you do so, the bootloader firmware will be loaded first, using the **Bootloader Image Path** setting from your preferences, and then the target application firmware will be loaded subsequently as part of the process.

When using InSight Desktop to program the EM250, the bootloader firmware (or the application firmware if this was uploaded during the process as well) will begin executing on the device as soon as the programming and verification of the firmware has completed. You can begin interacting with the bootloader (or the application) as soon as the circular icon beside that device's adapter has turned solid green again (indicating a successful state).

Bootloading and Running the Nodetest Application

Use the bootloader to upload the nodetest application to the device so you can set tokens and upload an application to the device.

The nodetest application is included in the EmberZNet PRO stack installation directory, in the /app/nodetest subdirectory. This directory contains both HEX- and EBL-formatted versions of the nodetest application. This enables you to upload it using either an Ethernet connection to an InSight Adapter or a serial connection to the UART of the EM250. Over-the-air upload is also possible using software on another EmberZNet PRO device to bootload the image on the bootloader's default radio channel (13). However, that procedure is beyond the scope of this document. For more information about over-the-air bootloading with the standalone bootloader, see the "standalone-bootloader-demo" sample application. For more information about over-the-air bootloading with the application bootloader, see Ember document 120-3029-000, *Ember Application Development Fundamentals*, and the "app-bootloader-demo" sample application.

Uploading and running nodetest by Ethernet

To begin uploading nodetest by Ethernet, first be sure that you have completed the steps in the earlier section “Hardware and software setup.” Once those steps are complete, follow the steps in the section “Installing the bootloader image via InSight Desktop,” substituting the EBL file in the /app/nodetest subdirectory of your EmberZNet PRO installation in place of the firmware file to be uploaded in step 4 of that procedure.

Once the upload in InSight Desktop completes, you can interact with nodetest via Ethernet if your device mates to Ember’s EM250 Breakout Board. To do this, perform the following steps:

1. Make sure that your EM250 Breakout Board is properly configured to access the UART (SC1 on the EM250) by Ethernet. This configuration is described in Ember document 120-2003-000, *EM250 Breakout Board Technical Specification*.
2. Make sure that the InSight Adapter is configured to use 115,200 bps as the serial port rate for port 1. This can be done by connecting to the adapter’s administrative interface (by USB or by telnet to TCP port 4902) and issuing the `port 1 115200` command. (Refer to the *Development Kit User’s Guide* for your Ember chip for details about using the InSight Adapter and accessing the administrative interface.)
3. Open a telnet connection to TCP port 4901 on the InSight Adapter where your device is connected. For example, for an EM250 device connected to an InSight Adapter with the IP address 192.168.0.1, you would enter the following text at a Windows command prompt:


```
telnet 192.168.0.1 4901
```
4. Press **Enter** in the telnet window to start the nodetest application.

Uploading and running nodetest via serial connections

1. Connect the device to your computer through a serial port or a serial cable, or a USB to serial adapter if you do not have a serial port.
2. Use a terminal program such as Windows HyperTerminal to establish an RS-232 serial connection to the device, at 115200 baud, 8 bits, 1 stop bit, no parity, and no flow control.
3. When the connection opens, press **Enter** and the bootloader menu is displayed.

Note: If an application is already on your device, you must ground GPIO5 and reset the device to bring it into bootloader mode, unless you have a mechanism to do this in the application currently loaded.

4. Type the number beside the **upload ebl** option from the menu, followed by the **Enter** key. The bootloader then displays `begin upload`, and a series of `C` characters are printed out to the terminal. This indicates that you can send a file.
5. In HyperTerminal, choose **Transfer | Send File**. Click **Browse** and browse to the app/nodetest/ subdirectory in your EmberZNet PRO installation.
6. Choose the `.ebl` file in that directory and click **Open**.
7. Be sure that the upload protocol is set to **Xmodem**, then click **Send**. After upload, the message `Serial upload complete` is displayed.
8. If you want to confirm that the file was successfully uploaded, you can use the **ebl info** option in the bootloader menu to display platform-specific information about the

uploaded file. The resulting string should tell you the microcontroller architecture (such as “XAP2”), microcontroller model (such as “em250”), radio model (such as “em250”), and board name (such as “dev0455”) for the uploaded file.

9. Enter the menu option number for “run” to execute the uploaded image. Pressing Enter again will start the nodetest application.

nodetest commands

The nodetest application runs the serial port at 115200 for the EM250, so be sure that your baud rate has been set accordingly.

A carriage return initiates the nodetest application on reset or power-up. This displays the power-up prompt, ending in the `>` (greater than) symbol. Table 2 provides the nodetest command set.

Table 2. nodetest Commands

Command	Description
<code>?, help</code>	Prints the Help menu.
<code>bootload</code>	Launches the bootloader application.
<code>calchannel</code>	Uses <code>calchannel x</code> to switch to channel <code>x</code> and perform calibration. (Uses current channel if <code>x</code> is not specified.)
<code>setChannel</code>	Sets the channel (11 by default). For valid values, see the document <i>EM250 Radio Communication Module (120-2001-000)</i> .
<code>getChannel</code>	Gets the channel. For valid values, see the document <i>EM250 Radio Communication Module (120-2001-000)</i> .
<code>ledoff</code>	Use <code>ledoff x</code> to turn BOARDLED <code>x</code> off
<code>ledon</code>	Use <code>ledon x</code> to turn BOARDLED <code>x</code> on.
<code>ledtest</code>	Cycles the 4 LEDs (at their default GPIOs on the EM250 Breakout Board) until told to stop.
<code>rx</code>	Puts the device into receive (RX) mode on the current channel. The following commands are valid while in receive mode: <code>c</code> Clear statistics. <code>q</code> Query statistics. <code>e</code> Exit receive mode.
<code>shutdown</code>	Places the chip in the lowest power mode for deep sleep current measurements.
<code>sleep</code>	Sends the radio to sleep (<code>x=1</code>) or wakes it up (<code>x=0</code>).
<code>tokdump</code>	Dumps all known tokens and their values.
<code>tokread</code>	<code>tokread <key></code> shows the contents of the token indexed by <code><key></code> as the stack will read it when run.
<code>tokscrub</code>	Erases and reinitializes simulated EEPROM, deleting all tokens stored there.
<code>tokwrite</code>	<code>tokwrite <key></code> writes a new value to the token indexed by <code><key></code> and prompts for each byte of data. Manufacturing tokens cannot be written with this command.
<code>tx</code>	Transmits the specified number of packets on the current channel (infinite if 0).

Command	Description
<code>settxpow</code>	Sets power to specified dBm. For valid values, see Ember document 120-2001-000, <i>EM250 Radio Communication Module</i> .
<code>settxpowmode</code>	Use <code>settxpowmode x y</code> for x=0 or 1 and y=0 or 1 to engage Boost mode (x=1) for the chip or switch to using the external PA (RF_TX_ALT_P/N) signal path (y=1).
<code>txstream</code>	Performs a modulated carrier wave transmission on the current channel.
<code>txtone</code>	Performs an unmodulated carrier wave (“tone”) transmission on the current channel.
<code>setSynOffset</code>	Use <code>setSynOffset x</code> to offset the synth frequency by x 11kHz increments, where x is a signed integer. This command allows for tuning the 24MHz crystal frequency in manufacturing (+1.397MHz/-1.408MHz range). This command applies the offset directly to the radio register and does not store data in the corresponding MFG_SYNTH_FREQ_OFFSET token. This token needs to be set separately to store and apply this offset.

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*.

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 by a serial port.
2. Follow the procedure in the section “Bootloading and Running the Nodetest Application” to 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.
5. Optional: Set a power level on the test device by typing `settxpower x`, specifying the power level to use with *x*. (See Ember document 120-2001-000, *EM250 Radio Communication Module*, for transmit power levels.)
6. Optional: Engage Boost mode or the external PA signal path using the `settxpowmode` command with the appropriate arguments. (See Table 2 for details about command usage.)

7. On the known good device, type `rx`, which sets the device to receive and display statistics for each packet received.
8. On the test device, type `tx 64` to transmit 100 (64 hex) packets. (Note that `tx 0` sends infinite packets.) Type `e` to exit.
9. 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 by Ethernet 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.

On the receive device, you can type and `e` to exit receive mode. Exit receive mode to clear all receive statistics. See Table 2 for all nodetest commands.

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 with the commands listed in Table 2 may be required for FCC or CE compliance testing.

For more detailed information on manufacturing test and Ember’s recommendations for manufacturing test, consult Ember document 120-5016-000, *Manufacturing Test Guidelines*.

Setting Stack and Application Tokens

The EmberZNet PRO stack maintains several non-volatile settings (tokens) used for network operation. Although an application linked with a debug build of the stack will automatically set these tokens if they are unset, you may wish to set these explicitly prior to runtime. Additionally, certain applications may require the use of their own tokens for non-volatile data storage. If the final application does not initialize these tokens on its own, the Token Utility application can be built with knowledge of these tokens and their initial values so that it can program and verify them.

To use the Token Utility application for programming stack and application token default values, do the following:

1. Open the EM250 Token Utility workspace for xIDE, found in the top-level EmberZNet directory.
2. If using custom application tokens, determine the header file with these token definitions. In the **Compiler Preprocessor** options for the project, define the `APPLICATION_TOKEN_HEADER` to match the path to this file.
3. Define the `CONFIGURATION_HEADER` for the project to a file matching your stack configuration parameters (such as table sizes and other static memory allocations).
4. Build the project’s target using xIDE for EM250.

5. Using a procedure similar to the one described in the earlier section “Bootloading and Running the Nodetest Application,” bootload the resulting Token Utility EBL or XPV/XDV file (found in a subdirectory in the “build” folder of the EmberZNet PRO installation) to the device, so that you can set tokens and upload an application to the device.
6. Using a serial or telnet connection with port speed set to 115200 bps, press **Enter** to initiate the Token Utility application.
7. Use the `tokmap` command to output the map of the token storage area. Verify that this matches the expected static memory allocation that the final application expects.
8. Use the `loadtoks` command to initialize all stack and application tokens to their default values as defined by the Application Token Header file.
9. Use the `tokdump` command to confirm that the values of the user-settable tokens match have been set appropriately.

More information about the Token Utility application can be found in the Sample Applications descriptions page, available as `sampleApps.htm` in the “app” subdirectory in the EmberZNet PRO stack installation.

Bootloading the Final Application

After you are satisfied that the tokens are in order and the radio is functioning properly, you can bootload the final application to the device.

To load the final application, locate the XPV/XDV or EBL file for that application and then upload it via Ethernet or serial connection as described earlier in the “Bootloading and Running the Nodetest Application” section of this document. If you want to examine the flash memory contents of the chip to verify that the image was loaded correctly, you can use the “`em2xx_unload.exe`” utility found in the SIF Toolkit (where “`em2xx_load.exe`” is located).

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 FAQ’s, 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.

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.

