



Bringing Up Custom Devices

For the EM260 Co-Processor

Ember EM260 chips are delivered to customers with only very basic 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 EM260 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 EM260 embedded flash is fully tested during production test, the flash contents are not set to a known state prior to shipment.

New in this Revision

Table 1 updated to include MFG_SYNTH_FREQ_OFFSET token.

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

Contents

Overview	2
Hardware and Software Setup	2
Loading EM260-nodetest Firmware	3
Loading firmware via the em2xx_load utility	3
Loading firmware via InSight Desktop	4
Setting Manufacturing Tokens	6
Running the EM260-nodetest Application	10
EM260-nodetest commands	11
Performing Functional Testing	12
EM260 Bootloader	13
Stand-alone bootloader	13
Loading the EM260 EmberZNet Firmware	13



Overview

To bring up a new hardware design (useful for complete characterization) and to bring up new devices based on an already-characterized design, follow these steps:

1. Set manufacturing tokens.
2. Upload and run the EM260-nodetest application.
3. Perform functional testing.
4. Load the EM260 EmberZNet firmware.

This document describes how to perform these steps.

Hardware and Software Setup

Before you can bring up custom EM260-based devices, a PC must be configured with the appropriate hardware and software, and you must become familiar with the process of loading firmware onto the EM260. The following steps explain how to set up a PC for communication with an EM260-based device and for loading firmware onto that device.

Note: The procedures in this document require a SIF programming device, such as the InSight Adapter or InSight USB Link, and access to the EM260's SIF interface. The SIF 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 120-2007-000, *EM260 Radio Communications Module*, and Ember's latest EM260 reference design. For details about creating a SIF programming device or purchasing InSight Adapters, contact Ember Customer Support at http://www.ember.com/support_index.html. The instructions that follow assume that Ember's InSight Adapter hardware is being used as the SIF programming and communication device.

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 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 the EM260 software, the toolkit DLLs are part of the EmberZNet PRO stack installer for this platform.
3. Attach the InSight Adapter to the EM260 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 EM260 device's InSight Port header.

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 the Ember Customer Support portal at http://www.ember.com/support_index.html.)
5. Be sure that you have the latest Ember EM260 firmware available. This firmware can be found as XPV and XDV files in the `build` directory in EmberZNet PRO stack releases for the EM260 platform.
6. Verify that the SIF Toolkit is properly installed on the machine where 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 “Loading firmware via the em2xx_load utility” or the “Loading firmware via InSight Desktop” section below depending on whether you prefer to load the firmware via the `em2xx_load` command line tool or the InSight Desktop graphical interface.

Loading EM260-nodetest Firmware

Firmware loading can be done through a command line tool (`em2xx_load.exe`) or a graphical user interface (InSight Desktop). Both methods are described in this section.

Note: Be sure that your host microcontroller does not disrupt the nRESET line of the EM260 during the firmware loading process, as this can cause the loading operation to terminate prematurely and leave the EM260 without any valid firmware. If possible, you may want to disconnect your host microcontroller from the EM260 interface to prevent this kind of disruption.

Loading firmware via the em2xx_load utility

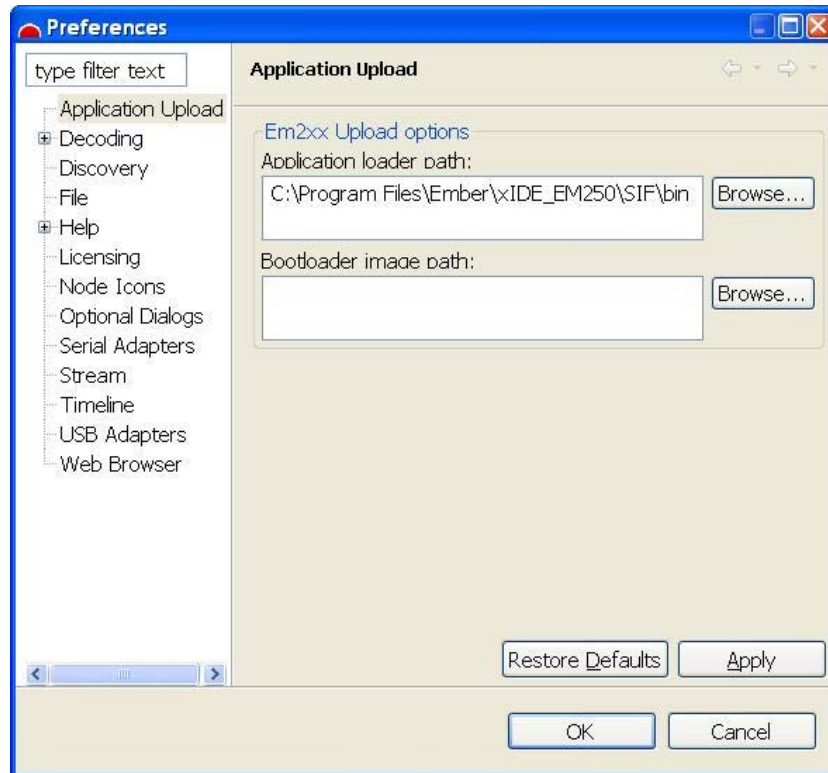
1. Open a new command prompt (choose **Start | Programs | Accessories | Command Prompt** from the Windows **Start** menu).
2. Determine the IP address or hostname of the InSight Adapter attached to the EM260 device being programmed. (If your adapters appear in InSight Desktop’s “Adapters” view, you can 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 EM260 device prefaced by the `-sid` argument, and supply the firmware filename (either the XPV or the XDV filename is sufficient). For example, to load an EM260 with the firmware located at `C:\EmberZNet\build\EM260-nodetest.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\build\EM260-nodetest.xpv"
```
4. As the programming process progresses, you should see messages indicating the following steps of a successful firmware loading:
 - Which XPV is being loaded, for example: `Loading code file C:\EmberZNet\build\EM260-nodetest.xpv...`
 - Which XDV file is being loaded, for example: `Loading data file C:\EmberZNet\build\EM260-nodetest.xdv...`
 - 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 EM260 device begin executing the firmware by issuing the `em2xx_load` command with the `-Run` argument. For example, for an InSight Adapter with the IP address 192.168.0.1, execute the following command from the command prompt:

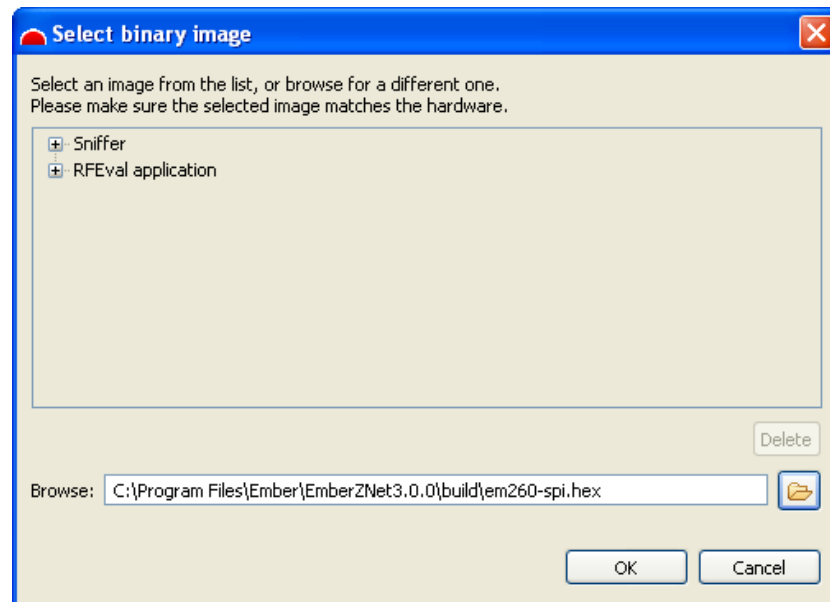
```
em2xx_load -sid 192.168.0.1 -Run
```

Loading firmware via InSight Desktop

1. Start the InSight Desktop software and verify that the InSight Adapter used to program the target EM260 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. Verify that the properties for the adapter, when expanded, describe the "Node type" as EM260. If this is not the case, be sure your EM260 device is attached to that InSight Adapter, remove power from the adapter, and then re-apply power to the adapter to cause it to detect the attached device again. Click the **Discover adapters** button in the Adapters view to refresh the list of available adapters, and check the "Node type" description for your adapter again to make sure that it reads **EM260**.
3. Right-click on the target device's connected adapter and select **Connect** from its context menu.
4. 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 loader 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 a valid XPV filename corresponding to the bootloader image (if you use InSight Desktop for loading EM250 devices as well) or else be left blank. Note that although this setting will not be used for this process (because an EM260 bootloader is not yet available), the setting must still contain a valid value. The **Always upload bootloader image** checkbox setting should be unchecked. The following figure shows an example of these settings.



5. Right-click on the target device's connected adapter in the Adapters view and select **Upload Network Coprocessor**. In the resulting dialog box, browse to the location of your EM260-nodetest firmware file (one of the XPV or XDV files from the `build` subfolder of your current EmberZNet PRO stack) and select it, then click **OK** to proceed with the upload of the firmware. A sample dialog is shown next.



When using InSight Desktop to program the EM260, the firmware will begin executing on the device as soon as the programming and verification of the firmware has completed. You can begin interacting with the firmware as soon as the circular icon beside that device's adapter has turned solid green again (indicating a successful state). Refer to the section "Running the EM260-nodetest Application" later in this document for details on how to interact with Ember's EM260-nodetest utility; details on the EM260 EmberZNet firmware can be found in Ember document 120-2060-000, *EM260 Datasheet*.

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 EM260 chip itself, as well as data that can be programmed by manufacturers of EM260-based devices. This information can be used by the host application or the EmberZNet PRO stack itself when device-specific properties are needed.

Warning: Unlike stack tokens, which reside in the simulated EEPROM section of the chip and can be set at runtime by the EM260 EmberZNet firmware, 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 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 `dev0470` on an EM260 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
```

Note: 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 EM260

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.
0x0041	8	MFG_CUSTOM_EUI_64	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. <i>Usage:</i> Optionally set by device manufacturer if using custom EUI64 address block.
0x0045	16	MFG_STRING	An optional device-specific string, for example, the serial number. <i>Usage:</i> Optionally set by device manufacturer to identify device.
0x004D	16	MFG_BOARD_NAME	An optional string identifying the board name or hardware model. <i>Usage:</i> Optionally set by device manufacturer to identify device.
0x0055	2	MFG_OSC24M_BIAS_TRIM	A relative amount (between 0x0 and 0xF) that the 24MHz crystal bias should be trimmed. As this value increases, the drive level to the crystal increases, changing the stability of the 24MHz 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. <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.

Word Offset	Size (Bytes)	Name	Description
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> Not required, as this is reserved for future compatibility with Ember's EM260 bootloader when available; however, a device manufacturer or software designer may want to set this value (if possible) for future compatibility with Ember's EM260 bootloader firmware.</p>
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 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>

Word Offset	Size (Bytes)	Name	Description
0x0080	40	MFG_ASH_CONFIG	Configuration token for EZSP UART only. It contains several parameters used by the ASH protocol, such as baud rate, various timings, and so on. <i>Usage:</i> Required for EZSP UART only.
0x0094	92	MFG_CBKE_DATA	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. <i>Usage:</i> Required by Smart Energy Profile certified devices.
0x00C2	20	MFG_INSTALLATION_CODE	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. 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: <ul style="list-style-type: none"> • 6 bytes = 0x0000 • 8 bytes = 0x0002 • 12 bytes = 0x0004 • 16 bytes = 0x0006. <i>Usage:</i> Required by Smart Energy Profile certified devices.

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 24MHz 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 11kHz 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*.

Running the EM260-nodetest Application

To begin running EM260-nodetest, first be sure that you have completed the steps in the earlier section “Hardware and Software Setup.” Once the upload in InSight Desktop completes, you can interact with EM260-nodetest via Ethernet through Ember’s InSight Adapter. To do this, perform the following steps:

1. Make sure that your EM260 device is connected to an InSight Adapter via the 10-pin InSight Port and that your InSight Adapter is receiving power and is accessible via your Local Area Network (LAN).
2. Supply power to the EM260 device, either through the InSight Adapter’s “INT” switch position or through an external power supply and the “EXT” setting on the InSight Adapter.
3. Open a telnet connection to TCP port 4900 on the InSight Adapter where your device is connected. For example, for an EM260 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 4900
4. Press the Enter key while the cursor is in the telnet window to start the EM260-HALtest application.

When the process is complete, you should see some diagnostic information printed in the telnet window, then a version string, and finally a > (greater than) prompt where commands can be entered. An example of this output is shown below:

```
> 32 samples cumCalib=0008A27E CLK1K_CAL was 5000 now 4514 (orig 5000)
Stack LWM used at reboot was 00E8/0200 bytes (~43%)

RESET:PWR

Ember Test Application v2.0
Oct 1 2006:20:34:00
Set to channel 0x0F
```

>

EM260-nodetest commands

The EM260-nodetest application uses the SIF port as a virtual UART for communicating with the EM260, so the port speed (baud rate) for communication is fixed (by the InSight Adapter) at 500,000 bps.

The EM260-nodetest application automatically displays a prompt on reset or power-up, ending in the > (greater than) symbol. Table 2 provides the EM260-nodetest command set.

Table 2. EM260-nodetest commands

Command	Description
? , help	Prints the Help menu.
calchannel	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.)
setChannel	Sets the channel (11 by default). For valid values, see Ember document 120-2007-000, <i>EM260 Radio Communication Module</i> .
getChannel	Gets the channel. For valid values, see Ember document 120-2007-000, <i>EM260 Radio Communication Module</i> .
rx	Puts the device into receive (RX) mode on the current channel. The following commands are valid while in receive mode: c Clear statistics. q Query statistics. e Exit receive mode.
spiptest	This SPI protocol test, when used with a breakout board or custom board with host that has host-haltest installed, allows for testing of the SPI between EM260 and host. Issue <code>spiptest</code> from the EM260 to configure the EM260 for the test, and issue <code>spiptest 260</code> from the host to trigger EZSP SPI transactions to occur. Host will reset the EM260 if communication is valid.
tokdump	Dumps all known tokens and their values.
tokread	<code>tokread <key></code> shows the contents of the token indexed by <code><key></code> as the stack will read it when run.
tokscrub	Erases and reinitializes simulated EEPROM, deleting all tokens stored there.
tokwrite	<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.
tx	Transmits the specified number of packets on the current channel (infinite if 0).
setTxPow	Sets power to specified dBm. For valid values, see Ember Document 120-2007-000, <i>EM260 Radio Communication Module</i> .
setTxPowMode	Use <code>settxpowmode x y</code> for <code>x=0</code> or <code>1</code> and <code>y=0</code> or <code>1</code> to engage Boost mode (<code>x=1</code>) for the chip or switch to using the external PA (RF_TX_ALT_P/N) signal path (<code>y=1</code>).
txstream	Performs a modulated carrier wave transmission on the current channel.
txtone	Performs an unmodulated carrier wave (“tone”) transmission on the current channel.

Command	Description
<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 simulated EEPROM, refer to the *token.h File Reference* in Ember document 120-3020-000, *EmberZNet API Reference for the EM260 Co-Processor*, and Ember document 120-5037-000, *Using the Simulated EEPROM*.

Performing Functional Testing

At this point, you may want to use the EM260-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. Follow the procedure in the section “Running the EM260-nodetest Application” to upload EM260-rangetest to the known good device and then run it.
3. Make sure that EM260-rangetest 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 120-2007-000, *EM260 Radio Communications Module* for the device for 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 displays 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 Loading EM260-nodetest Firmware 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 EM260-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*.

EM260 Bootloader

For the EM260, Ember provides a “stand-alone” bootloader for either the SPI or UART based design. This bootloader is intended to update the firmware on the EM260 itself, not the host firmware. It is left to the user to develop a host firmware upgrade mechanism.

Stand-alone bootloader

Standalone Bootloading is a single-stage process that allows the application image to be placed into Flash Memory, overwriting the existing application image, without the participation of the Application itself. There is very little interaction between the standalone bootloader and the application running on the host. In general, the only time that the application interacts with the bootloader is when it wants to run the bootloader in which it will issue the `EZSP_LAUNCH_STANDALONE_BOOTLOADER` command. Once the bootloader is running, it receives bootload packets containing (new) firmware image either via physical connections (ex. serial, spi) or via the radio (over-the-air).

Note that for an EM260 connected to a host via a UART, the 260 may not be the target of an over-the-air upgrade. This is due to flash code space limitations.

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

Loading the EM260 EmberZNet Firmware

After you are satisfied that the tokens are in order and the radio is functioning properly, you can load the final application to the device. Note that the EM260 image comes in several variants depending on the functionality that is desired. The reason for the variants is that there is not enough flash available to create a single, fully functional image.

- Em260-spi-debug-only-no-bootloader — this version is for SPI connected hosts and contains full debug information. This does not contain a bootloader.
- Em260-spi-with-bootloader-no-debug — this version is for SPI connected hosts and contains a bootloader, but not full debug.

- Em260-uart-with-bootloader – this version is for UART connected hosts and contains a bootloader, but not full debug.

To load the final EM260 firmware, locate the appropriate hex file in the stack installer build directory and then upload it via SIF or serial connection as described in earlier in this document in the section “Loading EM260-nodetest Firmware.” 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_read.exe` utility found in the SIF Toolkit (where `em2xx_load.exe` is located). Details on the EM260 EmberZNet firmware can be found in Ember document 120-0260-000, *EM260 Datasheet*.

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.

