AN728: Over-the-Air Bootload Server and Client Setup

This application note describes the process users should follow to perform a Zigbee® OTA (over-the-air) bootloading session between a ZCL OTA Upgrade cluster client device and server device. The hardware used is a Mighty Gecko (EFR32MG12)-based development kit. (See the WSTK6001 User Manual for details about how to configure the WSTK development board for use with the EFR32MG12.) You can also refer to procedures in this document when setting up or testing OTA bootload downloads in your own development environments with your own hardware.

Before you begin implementing the procedures in this document, you should be familiar with the basics of building and loading application images on Silicon Labs platforms. See *QSG106: Getting Started with EmberZNet PRO* for details.

KEY POINTS

- Configure and build two Zigbee OTA client images.
- Configure and build the OTA server.
- Load the original client image.
- Load the NCP server software.
- Load the updated OTA client image on the server.
- Run the client / server setup.

1. Introduction

This application note describes the steps necessary to demonstrate a Zigbee OTA bootloading session between a ZCL OTA Upgrade cluster client device and server device. The procedures use EFR32MG12-based Mighty Gecko development boards. (See the WSTK6001 User Manual for details about how to configure the WSTK development board for use with the EFR32MG12.)

Please refer to the document *UG103.6: Bootloading Fundamentals* for more details about this bootloader. For details about the ZCL OTA Upgrade cluster, consult the latest published revision of the Zigbee cluster library specification.

Note: In the Silicon Labs AppBuilder ZCL tab, the ZCL OTA Upgrade cluster is referred to as OTA Bootloading. For consistency that terminology is used in this document as well. If you are searching Zigbee documents, be sure to search for 'OTA Upgrade.'

Refer to this application note when setting up and or testing the Zigbee OTA bootload cluster on your own. The Advanced section describes how this example can be expanded to use your own hardware configuration, and how to change the software configuration to support your own manufacturer-specific information.

For further reference please see:

- Zigbee Document #07-5123, *Zigbee Cluster Library Specification* - "Over-the-air Upgrading" chapter; available from http://www.zigbee.org.
- Silicon Labs document *QSG106: Getting Started with EmberZNet PRO*
- Silicon Labs document *UG391: Zigbee Application Framework Developer's Guide*
- Silicon Labs document *UG266: Silicon Labs Gecko Bootloader User's Guide*
- Silicon Labs document *AN1084: Using the Gecko Bootloader with EmberZNet and Silicon Labs Thread*

Dotdot, which is developed and certified by the Zigbee Alliance, is a generalization of all of the application layer functionality developed both in Zigbee PRO and the Zigbee Cluster Library (ZCL) for use over a variety of transports. See *AN716: Instructions for Using ImageBuilder* for information about OTA with Dotdot.

2. Hardware and Software Requirements

Many of the steps in this document are performed using Application Builder (AppBuilder), a component of the Simplicity Studio IDE. For more information on configuring an application, refer to *QSG106: Getting Started with EmberZNet PRO*, provided with your release.

A number of the procedures use IAR-EWARM as the compiler. The IAR-EWARM version must be compatible with the EmberZNet SDK version. See the release notes for the SDK to determine the compatible compiler version number.

These procedures use two EFR32MG12-based WSTKs (Wireless Starter Kits), one for the client device running in System-on-Chip (SoC) mode, and one as the Network Co-Processor (NCP) component of the OTA Server. A PC Host running Linux or Windows with Cygwin is also required as part of the OTA Server setup.

2.1 Client Hardware

The client device used in these procedures is an EFR32MG12-based WSTK running in SoC mode. You have two choices for the download space. The first is to use an external storage device, such as a serial dataflash or serial EEPROM, connected to the device. The second option, only available with devices with more than 512 kB flash space, is to use a portion of the main flash as the download space. Bootloaders using the second option are often referred to as either internal or local storage bootloaders. EFR32MG12 parts can use local storage; EFR32MG1 parts cannot. For a full list of supported external memory parts for EFR32MGx parts, see *UG266: Silicon Labs Gecko Bootloader User Guide*.

For Mighty Gecko-based devices using the EFR32MG1, only the serial dataflash option is available; no local storage application bootloader is presently offered. However, Mighty Gecko ICs whose part numbers begin with EFR32MG1x6 or EFR32MG1x7 contain an integrated serial flash that can be utilized just like off-chip serial dataflash but without any additional components.

EFR32MG12- or MG13-based devices with at least 512 kB of internal flash also support internal storage for use with a local storage application bootloader design, as well as external storage for use with an SPI flash-based application bootloader design. In the procedures described in this document, internal storage is used on an EFR32MG12 with 1024 kB of flash. Contact Silicon Labs technical support for assistance with other memory layouts for other devices.

The Zigbee OTA client cluster can be used with an EmberZNet PRO-based device running in either SoC or EZSP NCP mode. This document does not show how to configure an EZSP host application for an NCP-based client, but the server hardware and setup described in section 2.2 Server Hardware is the same when the client is using EZSP. This application note shows one example of how the OTA client cluster can be used with a specific hardware configuration.

The client software is based upon the ZCL Application Framework V2 contained within the latest version of the EmberZNet PRO stack.

2.2 Server Hardware

The NCP device used in these procedures is an EFR32 Mighty Gecko-based WSTK with a UART-connected host application communicating via EmberZNet Serial Protocol (EZSP). In this application note, either a Linux system or Windows PC with Cygwin is required to run the host software that connects to the NCP. The storage for the OTA software images is the host's local file system. To avoid linking issues, do not use a virtual machine to run the host.

It is possible for an OTA server to run on an EFR32 Mighty Gecko-based SoC, or to use an NCP with a different (non-POSIX-based) host system. A different configuration than the one presented here requires an alternative mechanism for pushing images into the OTA server so they can be then served up to clients through the Zigbee OTA bootload cluster protocol. For example, software images could be pushed through a utility backhaul, an Ethernet connection to a local network, or some other proprietary mechanism.

A Linux system connected to a development board acting as a UART NCP is only one possible option to serve up OTA files. It was the friendliest option for an OTA server because there are many mechanisms to push OTA files into the server's file system.

The following figure shows a diagram of the hardware configuration used for the Zigbee OTA application bootload procedures. Note that only one client is required.

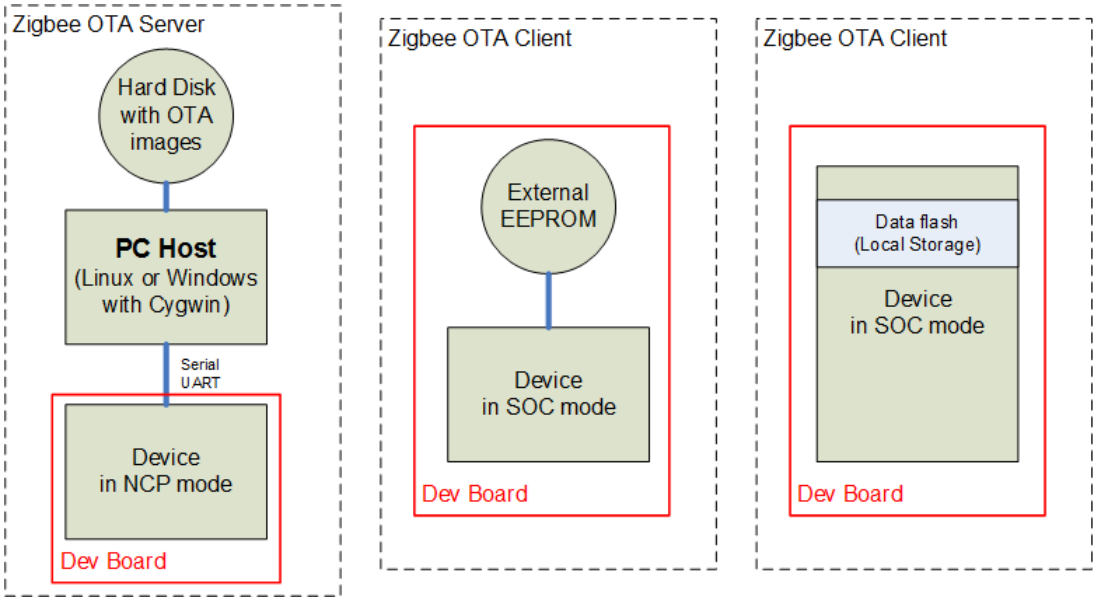


Figure 2.1. Zigbee OTA Application Bootload Hardware Diagram

3. OTA File Storage

3.1 About the Zigbee OTA File Format

The file format is defined in Zigbee document 07-5123, *Zigbee Cluster Specification*, in the “OTA File Format” section of the “Over-the-air Upgrade” chapter. Images are composed of a Zigbee OTA header, followed by one or more blobs of proprietary bootloader data, and then an *optional* set of cryptographic signature data appended to the end. The OTA server only needs to read the OTA header to serve up the file, and thus can serve up files for different manufacturers and or different products. The following figure shows an example of the layout of the file format.

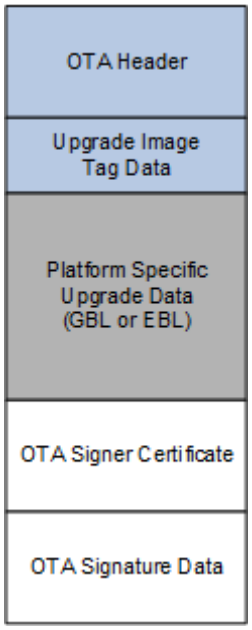


Figure 3.1. File Format Sent Over-the-Air

Silicon Labs has created a tool for generating OTA images called **Image-builder**. The non-ECC version of **Image-builder** is included with the EmberZNet PRO release in the **tool\image-builder** sub-directory. The post-build scripts in AppBuilder run Image-builder as part of a batch file to make the OTA file, provided that the OTA Bootload Cluster Client Policy plugin has been enabled. For more information on Image-builder, see *AN716: Instructions for Using Image Builder*.

OTA signing refers to signing the OTA image file, and is independent of any signature or encryption that may be present on the update image file inside the OTA wrapper file. Currently only the Smart Energy Profile has support for signatures on images. This application note assumes that OTA images do not need to be signed.

3.2 About OTA File Storage

The new image is saved in external flash or internal flash, depending on which bootloader is used.

- If using a “SPI Storage Bootloader”, the new image will be saved into SPI flash.
- If using an “Internal Storage Bootloader”, the new image will be saved into internal flash.

The offset of the new image is set in the properties of plugin “OTA Simple Storage EEPROM Driver”. There are two typical settings.

1. Using slot-manager: The slot offset is defined in the bootloader. In the application, set “Gecko Bootloader Storage Support” to any selection except “Do not use slot”. In this case, the “OTA Storage Start Offset” and “OTA Storage End Offset” will not be used.
2. Using absolute offset: In the application, you must set “Gecko Bootloader Storage Support” to “Do not use slot”. In this case the “Storage Slot To Save Images To” will not be used. The “OTA Storage Start Offset” and “OTA Storage End Offset” must be set correctly. The “OTA Storage Start Offset” must equal the start offset of a slot defined in the bootloader. The “OTA Storage End Offset” must equal the start offset plus the slot size of that slot.

4. General Procedure

The following outlines the steps to configure the OTA bootload cluster client and server. Details are presented in subsequent sections.

1. Configure and build the original OTA client image.
2. Configure and build the updated OTA client image.
3. Create the Zigbee OTA server files.
4. Build the PC host.
5. Load the original client image.
6. Load the EZSP NCP server software.
7. Load the updated OTA client image on the server.
8. Run the Zigbee OT client / server.

4.1 Configure and Build the Original Zigbee OTA Client Image

This procedure configures a Zigbee LO On/Off Light as an example, but it can be adapted for any sample application. Other Zigbee device types can be configured similarly. The example setup includes support for the Zigbee OTA bootload cluster client.

The procedures in this application note build two client images, the original version and the updated version. This client will be the original version. The image generated will run before the Zigbee OTA process upgrades the client device.

Here are the basic steps to configure the OTA client:

1. In Application Builder, create and name a new project. This can be blank or based on one of the provided examples. For the purposes of this exercise, you may want to name the file **ZNet_OTA_Client_631** (for the 6.3.1.0 stack).
2. In the **ZCL Clusters** tab:
   a. In the **Cluster List** pane on the bottom left, expand the **General** group.
   b. Check the **Over-the-Air Bootloading Cluster** client checkbox.

   Note: Some Zigbee device types do not allow you to enable this cluster by default. In that case, you can select **Zigbee Custom** from the ZCL Device Type picklist and then choose your same desired device type from that list. You can then add any other clusters to your endpoint configuration.

3. In the **Printing and CLI** tab:
   a. In the **Debug Printing** section, check the “Enable Debug Printing” checkbox.
   b. Expand the **Application specific debug printing** group.
   c. Check the **Compiled-in** and **Enabled at startup** checkboxes for the **Ota Bootload** cluster.
4. In the **HAL** tab, click [**Open Hardware Configurator**] to launch the Hardware Configurator:
   a. In the resulting Configurator window, click the **DefaultMode Peripherals** tab.
   b. Scroll down to the **Serial** section of peripherals and enable **Serial** and **Virtual COM Port** peripherals, if they are not already enabled. Normally, the 'Port for application serial' property of the Serial port should be defined to USART0.
   c. Click the AppBuilder tab for your ISC file to resume editing your application configuration.
   d. Select **File > Save all**.
5. In the **HAL** tab, click the **Bootloader** drop-down menu and specify **Application**.

   Note: A Local Storage bootloader can also be used with the OTA Client setup, but this is a more advanced use case and is not discussed here. See *UG266: Silicon Labs Gecko Bootloader User's Guide* and *AN1084: Using the Gecko Bootloader with EmberZNet and Silicon Labs Thread* for more information about bootloader configuration options.

6. In the **Plugins** Tab, make sure the following plugins are checked (other plugins may be checked by default as well):
   - OTA Bootload Cluster Client
   - OTA Bootload Cluster Client Policy
     a. In the **Firmware Version** text entry box, set the value to 1 as your original version number.

     Note: This is a critical value, as later steps assume you have the version set to 1.

   - OTA Bootload Cluster Common Code
   - OTA Bootload Cluster Storage Common Code
   - OTA Simple Storage Module
   - OTA Simple Storage EEPROM Driver
     a. Check the **SOC Bootloading Support** checkbox.
     b. Set EEPROM Device Read-modify-write support to False.

     Note: The OTA storage start offset and end offset values in this plugin may need to be changed for other storage layouts, including layouts involving Storage Slots. Contact Silicon Labs technical support for guidance on choosing other values based on your use case.

   - EEPROM
   - OTA Cluster Platform Bootloader
7. Click [**Generate**].

8. Build the application according to your preferred method. Make sure that the .GBL (Gecko Bootloader) file is generated.

   Note: An EEPROM/dataflash driver is not specifically included in the application's project file. The application will call into the bootloader's storage driver code to perform the low-level calls to the serial dataflash. This is why it is important to enable a bootloader in the client configuration and later to load a bootloader into the device.

   For a Zigbee OTA Bootload Cluster Client, AppBuilder automatically creates a Windows .BAT file that executes Image-builder and wraps the GBL file in the Zigbee OTA file format. The parameters passed to Image-builder are based upon those specified in the application configuration, specifically the values declared in the "OTA Bootload Cluster Client Policy" plugin. The OTA image will be located in the compiler Debug folder rather than in the main project folder.

4.2 Configure and Build the Updated Zigbee OTA Client Image

This step builds the updated version image. The images are identical except for the version number embedded in them.

4.2.1 Copy the Original OTA Client Image

Copy the original ZNet_OTA_Client.gbl file (the one built with the OTA Bootload Cluster Client Policy: Firmware Version set to **1**) to another folder to avoid it being overwritten when the "updated" version is built next to enable an OTA update. Rename this file appropriately, for example: ZNet_OTA_Client_631_v1.gbl.

4.2.2 Modify, Generate, and Build the Updated OTA Client Configuration

To generate a new OTA Client configuration, perform the following steps.

1. Open the previously saved OTA Client ISC file.
   a. Open the **File** menu and select **Open**.
   b. In the dialog box, navigate to the location of the saved ISC file and select **Open**.
2. Select the **Plugins** tab.
   a. Select the **OTA Bootload Cluster Client Policy** plugin.
   b. Change the value in the **Firmware version number** entry box to 22. This value is an arbitrary choice but it must be greater in value than the previous version (1).
3. Click [**Generate**].
4. Build the project. Locate the .ota file created in the compiler output directory; for example, \IAR ARM - Default\ZNet_OTA_Client_631.ota.
5. Copy this file to the directory where you saved your ZNet_OTA_Client_631_v1.gbl and rename the .ota file to track the version number, for example ZNet_OTA_Client_631_v22.ota

4.3 Create the Zigbee OTA Server Files

This procedure creates the source for a Zigbee 3.0 gateway host device. The device includes support for the Zigbee OTA bootload cluster server. The host server software can support both signed and unsigned OTA images since it will not do any verification on these files. To configure the OTA server:

1. In Application Builder, start a new AppBuilder project.
2. Select **Silicon Labs Zigbee** and click [**Next**].
3. Select **EmberZNet <version> GA Host** (not SoC) and click [**Next**].
4. Select the **Z3Gateway** host sample application and click [**Next**].
5. Rename your project and click [**Next**].
6. Do not select a Board. Under Part, select **None (Compatible)**, and leave **No Toolchain** selected. Click [**Finish**].
7. Click [**Generate**].

4.4 Build the PC Host

Building the OTA Server can be done either on a Linux box using GCC, or a Windows box running Cygwin and GCC. Both processes are described below.

4.4.1 Building on a Linux Box

Building the OTA server can be done on a Linux system. This application note assumes that you are NOT using a cross-compiler and that the target system is the same system as the one where the development tools exist. It is possible to cross-compile for a different Linux system (for example, on an x86 PC targeting an embedded ARM Linux system). See section 5.4 Using a Cross-compiler for the OTA Server for details on how to cross-compile for other systems.

Building the OTA Server for Linux requires a number of development tools. We assume you have access to these tools. They are as follows:

- Make
- GCC
- sed
- awk
- xargs
- The standard C Library and its development headers
- The Readline Library and its development headers
- The Ncurses Library and its development headers

The steps to build the OTA server under Linux follow. It is assumed that these steps are run after configuring the server in Simplicity Studio.

1. Copy the EmberZNet PRO stack files from their installed location on Windows to a directory on the Linux PC. This should include the files that were generated in a previous step.
2. Launch a Bash Shell.
3. Change to the EmberZNet project directory on your Linux PC or Raspberry Pi.
4. Run **Make** on the generated Makefile from that directory, for example `make –f OtaServer.mak` or simply `make` if the generated Makefile is in the current working directory and is specifically named “Makefile”.

The compilation should complete successfully.

4.4.2 Building on a Windows Box Under Cygwin

Building the OTA Server under Windows requires Cygwin. Cygwin is a set of native Windows tools that emulate a Linux environment. Cygwin can be downloaded from the Internet from http://www.cygwin.com.

Cygwin is actually a collection of hundreds of small programs that make up a Linux distribution. Cygwin can be configured in a myriad of different ways for a particular Windows PC and so by default may not include the required tools for building an EmberZNet PRO host application. Therefore proper installation of the required tools is necessary before actually building the EmberZNet PRO code.

The following is a general list of the packages necessary to build an EmberZNet PRO host application under Cygwin.

- Bash shell
- Make
- GCC4
- sed
- awk
- xargs
- The standard C Library and its development headers
- libreadline-devel - The Readline Library and its development headers
- libncurses-devel - The Ncurses Library and its development headers

Note: Although it is possible to use the Cygwin tools to build an application that does NOT require the Cygwin DLL, it is not possible to do this with an EmberZNet PRO host application. The host application requires access to the Cygwin DLL to correctly run certain parts of the Ember application framework code.

The steps to build the OTA server under Windows using Cygwin are as follows.

1. Launch a Cygwin Shell.
2. Change to the project directory, for example /cygdrive/c/SiliconLabs/SimplicityStudio/v4/developer/sdks/gecko_sdk_suite/v2.2/app/builder/Z3GatewayHost_631
3. Enter `make`.
4. A \build\exe\ folder will be created with your app Z3GatewayHost_631.exe inside it. This is your Host app.

The compilation should complete successfully.

4.5 Load the Original Client Image

Flash an appropriate Application bootloader for your board, such as \developer\sdks\gecko_sdk_suite\v2.3\platform\bootloader\sample-apps\bootloader-storage-spiflash-single\efr32mg12p432f1024gl125-brd4161a\bootloader-storage-spiflash-single-combined.s37 for a BRD4161A. For simplicity, in this case, you can use the same bootloader for both the OTA Client and Server if you are using the same board (for example, BRD4161A).

You can use Simplicity Studio to do this (see *QSG106: Getting Started with EmberZNet PRO* for instructions on loading application and bootloader firmware) or use Simplicity Commander commands (see *UG162: Simplicity Commander Reference Guide* for additional detail):

```
commander flash -d EFR32 [path to bootloader-storage-spiflash-single-combined.s37]
commander flash -d EFR32 [path to ZNet_OTA_Client_632.gbl]
```

Tip: You can drag the file image into the CMD window to include the file reference easily.

4.6 Load the EZSP NCP Server Software

In order to use a Mighty Gecko device connected to a PC host through USB, it is necessary to load the EZSP NCP software onto the target chip. The following describes the process using an EFR32MG12 as an example for the NCP target chip, and precompiled images. You can also build your own images as described in *AN1010: Building a Customized NCP Application*.

1. Right-click the NCP target device in the Devices pane.
2. Select **Upload Application**.
3. In the resulting dialog box:
   a. Click the folder icon to the right of the **Application image path** text box.
   b. Navigate to the location of your EmberZNet PRO release, browse to the “ncp-images” subdirectory, and select the pre-built NCP image. Use the ncp-uart-hw.s37 file, for example: `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v2.3\protocol\zigbee\ncp-images\efr32mg12p432f1024gl125-brd4161a\ncp-uart-hw.s37`.
   c. Check Bootloader image, then browse to the bootloader image folder, for example :

      `\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\platform\bootloader\sample-apps\bootloader-storage-spiflash-single\`

      Note that some SDKs have variations on this path.

      Open the folder that corresponds to your board and part number and select the .s37 file, for example:

      `\efr32mg12p432f1024gl125-brd4161a\bootloader-storage-spiflash-single-combined.s37`
   d. Click [**OK**].
4. Wait for the device to successfully finish a flash load of the NCP.

4.7 Load the Updated OTA Client Image on the Server

The OTA Server application looks for OTA files in a directory called **ota-files** that is a sub-directory in the current working directory from which the server application is launched.

1. Launch a Cygwin or Linux shell on the OTA Server PC.
2. Navigate to the directory where the OTA Server application will be run, make a subdirectory, and copy the OTA file to it.

   For example, in Linux:

```
cd /home/rob/
mkdir ota-files
cp -i OtaClient-v22.ota ota-files/
```

   For example, on Cygwin:

```
cd /cygdrive/c/SiliconLabs/SimplicityStudio/v4/developer/sdks/gecko_sdk_suite/<v
ersion>/app/builder/Z3GatewayHost_OTA/build/exe
mkdir ota-files
cp –i OtaClient-v22.ota ota-files/
```

You can also copy the file using the Windows Explorer or Linux/RasPi GUI.

4.8 Run the Zigbee OTA Client / Server

1. Make sure the WSTK for the NCP device is connected to the PC with a USB cable.
2. Launch the OTA Server by executing the OTA server application from the command line, and pass it the location of the communications port. The communications port will either be a USB or serial port device. The method for determining which port the EFR32MG12 is connected to depends on the operating system of the host PC.

   Under Linux

   The USB port is registered as /dev/ttyUSBX, where X is a number as assigned by the operating system. For example the following would use `ttyUSB0`.

   a. Launch a Shell.

   b. Change to the directory where the OTA Server application is located. This should be the same directory where a sub-directory called “ota-files” exists, for example: `cd /home/rob.`

   c. Enter `./OtaServer –p /dev/ttyUSB0`

   Under Windows

   The USB port is either COM1, COM2, COM3, and so on. The COM port can be determined by watching the Ports (COM & LPT) section of Windows Device Manager while unplugging and replugging the WSTK's USB connector. The following example uses COM1.

   a. Launch a Cygwin Shell.

   b. Navigate to the host app you built, for example:

   ```
   cd /cygdrive/c/SiliconLabs/SimplicityStudio/v4/developer/sdks/gecko_sdk_suite/v2.2/app
   /builder/Z3GatewayHost_631
   ```

   c. To run your host app with the COM port as a parameter, type the command: `./Z3GatewayHost_631.exe -p com1`
3. Connect to the OTA Client device in your Simplicity Studio Serial 1 window or by launching a Windows command prompt and connecting via telnet to its IP address on port 4901.
4. Make sure both the client and server devices have Over-the-Air Bootloading Cluster printing enabled.

   a. Execute the following command on the OTA Client: `plugin ota-storage-common printImages`

   b. Execute the following command on the OTA Server: `plugin ota-storage printImages`

   In both cases output should print to the CLI. If no output is visible, go back and verify the configuration of the application and its Debug Printing configuration.
5. Make sure that the client is running the older version of the client software. From the OTA client’s CLI type the following command. You should see the corresponding output:

```
OtaClient>plugin ota-client info
Client image query info
Manuf ID:         0x1002
Image Type ID:    0x0000
Current Version:  0x00000001
Hardware Version: NA
Query Delay ms:            300000
Server Discovery Delay ms: 600000
Download Delay ms:         0
Run Upgrade Delay ms:      600000
Verify Delay ms:           10
Download Error Threshold:  10
Upgrade Wait Threshold:    10
```

Note that the current version field says 0x00000001.

6. Verify that the OTA Server has a newer image loaded in it. From the OTA server's CLI type the following command. You should see the corresponding output:

```
OtaServer>plugin ota-storage printImages
Image 0
  Header Version: 0x0100
  Header Length:  56 bytes
  Field Control:  0x0000
  Manuf ID:       0x1002
  Image Type:     0x0000
  Version:        0x00000022
  Zigbee Version: 0x0002
  Header String:  Our Test Image
  Image Size:     146954 bytes
  Total Tags: 1
    Tag: 0x0000
      Length: 146880
1 images in OTA storage.
```

   Note that the version number is 0x00000022. The exact size of the image may vary.

7. Erase any previous version file that has been saved in the client. This is not normally necessary in a production system, but in this example a previous version may cause problems. Enter the following command on the client:

```
plugin ota-storage delete 0.
```

8. Turn off receive message printing to limit console output during the download. Skipping this step will not negatively impact the functionality.
   a. On the server: `option print-rx-msgs disable`
   b. On the client: `option print-rx-msgs disable`
9. Join the devices into the same network.
   a. On the server: `net leave`
   b. On the server: `plugin network-creator start 1`
   c. On the server: `plugin network-creator-security open-network`
   d. On the client: `plugin network-steering start 0`

      If the client joins successful, then you should see a dialog about trust center interactions in the output.
10. To verify that the client is connected to the server, type `info` on both the client and the server. The PAN IDs should be the same.
11. Start the OTA Client's state machine on the client: `plugin ota-client start`

    Note: The download may take up to 10 minutes.

12. If all goes well you should see the following output:

```
Download: 100% complete
Bootload state: Verifying Image
Last offset downloaded: 0x00023E6C
Starting EBL verification
EBL passed verification.
Custom verification passed: 0x00
Bootload state: Waiting for Upgrade message
Sending Upgrade End request.
OTA Cluster: wait for 0 s
RXed timeOut 0x00000000 s, MAX timeOut 0x00000DBB s
Adding 2000 ms. delay for immediate upgrade.
Countdown to upgrade: 2000 ms
Bootload state: Countdown to Upgrade
Applying upgrade
Executing bootload callback.
Reset info: 0x02 (BTL)
```

4.9 Repeating the Procedure

Once a Zigbee OTA client is upgraded through the Zigbee over-the-air bootload cluster, it can only be upgraded again if the server has a different version of software.

In order to perform a Zigbee OTA firmware download again it is necessary to do one of the following:

1. Downgrade the running image of the client to the original version

   or
2. Create another client image with a newer software version.

Downgrade the Running Image to the Original Version

Re-run the steps in the section 4.5 Load the Original Client Image. Load version 1 of the software, the original image.

Create Another Image with a Newer Software Version

Run the following steps:

1. Re-run the steps in section 4.1 Configure and Build the Original Zigbee OTA Client Image but specify a different Firmware Version that is greater than 22.

   Make sure that you click [**Generate**] to generate the build files.
2. Re-run the steps in sections and 4.4 Build the PC Host.

   Make sure that the **--version** argument passed to Image-builder reflects the newest version number.

5. Advanced Topics

This section presents a number of advanced topics that allow the developer to customize the bootload to their specific hardware and software.

5.1 Specifying Your Own Manufacturer ID

Follow these steps in order to specify a manufacturer ID other than 0x1002, which is the Silicon Labs-specific manufacturing ID.

Each member of the Zigbee Alliance has its own manufacturer ID. The list of manufacturer IDs is maintained in Zigbee Document 05-3874 Manufacturer Code Database. If your company does not have a manufacturer ID then it must request one from the Zigbee help desk (help@zigbee.org).

1. When configuring the OTA Client in AppBuilder, do the following:
   a. Select the **ZCL Clusters** Tab.
   b. In the **Manufacturer Code** text entry box on the right, enter your own manufacturer code.
   c. Click [**Generate**].
   d. Re-compile the application using IAR workbench.
2. When generating the Zigbee OTA image file, do the following.
   a. Specify your own manufacturer ID by passing that value to the **--manuf-id** argument when executing the **image-builder** command.

Note: You do NOT need to modify the manufacturer ID of the OTA server in this example. The server may have a different manufacturer ID than the OTA Client. You may choose to modify the manufacturer ID to match the OTA server device that your company is building.

5.2 Specifying Your Own Image Type ID

Follow these steps in order to specify an image type ID other than 0x0000.

1. When configuring the OTA Client in AppBuilder, do the following:
   a. Select the **Plugins** tab.
   b. Select the **OTA Bootload Cluster Client Policy** plugin.
   c. In the **Image Type ID** text entry box, enter your own image type ID code.
   d. Click [**Generate**].
   e. Re-compile the application using IAR workbench.
2. AppBuilder will automatically update the Windows Batch file command being executed by IAR Workbench to generate a Zigbee OTA file. The new Image Type argument will be passed to Image-builder with **--image-type**.

5.3 Modifying the OTA Client to Use a Different EEPROM

Many different external storage parts are available, and the choice of which one is used is based on many different factors. *AN772: Using the Ember Application Bootloader* and *UG266: Silicon Labs Gecko Bootloader User Guide* detail several different parts supported by Silicon Labs and their various device parameters.

If the external storage part does not support read-modify-write, then the AppBuilder application configuration must be updated. The following describes the process:

1. In AppBuilder, open the OTA client configuration (ISC) file.
2. Select the **Plugins** tab.
3. Select the **OTA Simple Storage EEPROM Driver** plugin and uncheck the option labeled **EEPROM Read-modify-write device Support.**
4. Click [**Generate**] to re-generate the appropriate AppBuilder files.
5. Re-compile the application.

Note: A new Bootloader will be required to support the change in EEPROM. You must load a new bootloader by following the steps in section 4.5 Load the Original Client Image, and selecting one appropriate to the bootloader you are using. For the example presented in this section the appropriate bootloader would be: `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\zigbee\platform\bootloader\sample-apps\bootloader-storage-spiflash-single\{chipName}\bootloader-storage-spiflash-single-combined.s37`.

5.4 Using a Cross-compiler for the OTA Server

It is possible to use a cross-compiler for building the OTA server for another target system. For example, building on an x86 Windows PC for an ARM Linux system. This process is relatively straight forward once the cross-compiler is properly installed and configured. To do this you must run make and pass it the name/location of the compiler and linker on the command-line.

For example:

```
make CC=/opt/crosstool/gcc-4.0.1-glibc-2.3.5/arm-unknown-linux-gnu/bin/arm-unknown-linux-gnu-gcc
LD=/opt/crosstool/gcc-4.0.1-glibc-2.3.5/arm-unknown-linux-gnu/bin/arm-unknown-linux-gnu-gcc
```

The Makefile expects a GCC style command-line compiler and it uses the compiler command (rather than the linker command directly) for linking.

5.5 Using Encrypted GBL Images

Encrypted images protect the new application image from inspection or tampering by using a symmetric encryption key. For more information about the Gecko Bootloader and encrypted GBL images, see *UG266: Silicon Labs Gecko Bootloader User Guide*.

5.5.1 Changing the Bootloader Type

The following instructions apply to EM35x devices. For EFR32MG devices, follow the instructions in *UG266: Silicon Labs Gecko Bootloader User's Guide.*

1. Load the ISC file configured with the OTA Bootload Cluster Client.
2. Switch to the **HAL** tab.
3. Open the **Bootloader** drop-down menu, and select **Secure Application** or **Secure Local Storage**.
4. A Secure Application bootloader requires an encryption key. There are two ways to acquire a key:
   - AppBuilder can randomly generate a key file
   - The User can specify a key file.

   a. To have AppBuilder randomly generate a key, click **Generate Key**: 

   b. To have AppBuilder use an existing key file, click **Browse**  and navigate to a file containing the key. The format of a key file is as follows:

   ```
   # Comment lines begin with '#'
   TOKEN_MFG_SECURE_BOOTLOADER_KEY: 0123456789ABCDEF0123456789ABCDEF
   ```

5. Click [**Generate**] in AppBuilder
6. Build the application using IAR.

   Note that the encrypted EBL output will be: build/ApplicationName/ApplicationName.ebl.encrypted
7. AppBuilder will automatically create a Zigbee OTA file containing the encrypted EBL: build/ApplicationName/ApplicationName.ota
8. Load a Secure Bootloader onto the device. Follow the steps in section 4.5 Load the Original Client Image, but select a Secure Application bootloader. Secure bootloader IAR and project files for the EM3x are located in the bootloader folders for the part, for example:

   ```
   \SiliconLabs\SimplicityStudio\<version>\developer\sdks\gecko_sdk_suite\<version>\protocol\zigbee\tool\bootloader-em357
   ```

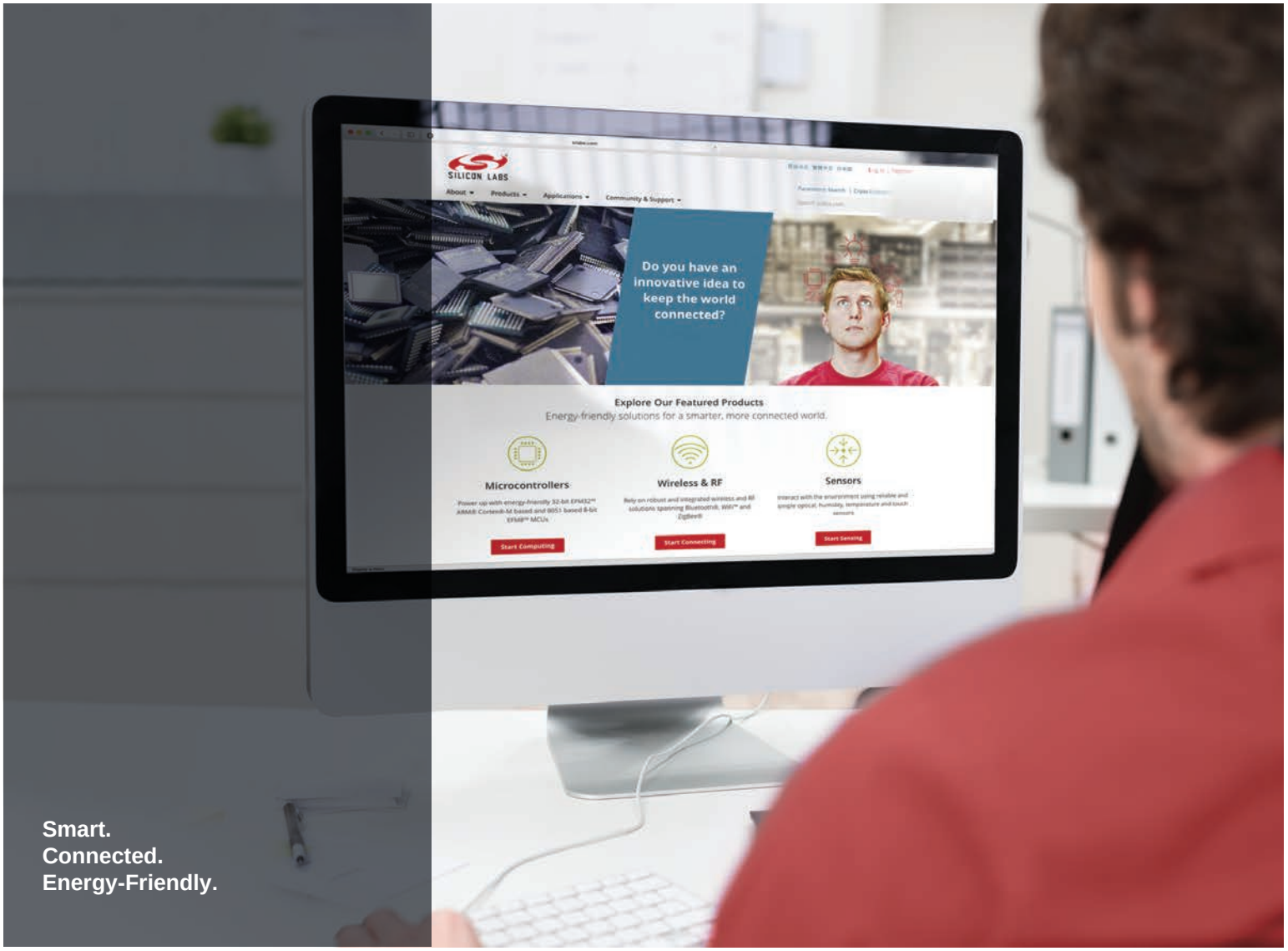
5.5.2 Load an Encryption Key onto the Device

The following instructions apply to EM35x devices and reference the em3xx_load utility. For EFR32MG devices, follow the instructions in *UG266: Silicon Labs Gecko Bootloader User's Guide.* See *UG162: Simplicity Commander Reference Guide* for more information on applicable commands.

In order for the secure bootloader to function correctly, a Secure Bootloader key must be loaded in the manufacturer tokens of the device. To load a key into the device use **em3xx_load** and pass it the key file that was specified previously in AppBuilder.

```
em3xx_load.exe --ip <ip-address-of-ISA3> --cibtokenspatch <key-file>
```

Note: The standalone bootloader encryption key is a different manufacturing token then the Secure Bootloader Encryption key.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com