Introduction

The Daintree Dimmable Light Demo application note describes the features of the Daintree Dimmable Light Demo software and how to execute it. The Daintree Dimmable Light Demo software is written for CEL’s EM357 Module, a module based on Silicon Labs’ EM357 System on Chip (SoC), to be compliant with Daintree Network’s solution for lighting control. The architecture behind the Daintree solution is a ZigBee® Home Automation Network, supporting devices that are a part of the ZigBee Home Automation Profile.

The Dimmable Light device participates in a Home Automation Light Network. The network could consist of a gateway, an on/off dimmable switch and dimmable lights. The gateway, also known as the Wireless Access Controller (WAC) and the on/off dimmable switch are provided by Daintree Networks. The dimmable lights interact with the gateway and the on/off dimmable switch.

The dimmable light powers up and looks for a network to join if it is not already part of a Private Area Network (PAN). Once the WAC allows the Dimmable Light device to join its network, the dimmable lights can then be controlled by either the dimmable switch or the WAC via the ControlScope™ software.

Features

- Based on the ZigBee Home Automation Profile with Dimmable Light as the device type.
- Supports the Commissioning Cluster in addition to the ZigBee Home Automation Profile.
- A push button to restore device to factory defaults after holding down for a minimum of three seconds.
- Supports the ZigBee Pro specification (stack supplied by Ember).
- Supports the optional User Descriptor ZigBee Device Profile command in addition to all mandatory ZigBee Device Profile commands.
- Supports Frequency Agility through the Ember ZigBee Stack. No application level code needed.
- Supports responses to Many to One route requests through the Ember ZigBee Stack. No application level code needed.
- Mains powered device – device does not sleep.
- Fragmentation.
- Attribute reporting per ZigBee specification.
- A-mode commissioning (automatically searches for a network to join without user intervention).
- Provisioning is not needed as the device is a server and only responds to commands sent to it.
- Over-the-Air (OTA) firmware upgrade.
- Routing capabilities (supported within the Ember ZigBee Stack).
- Uses Pulse Width Modulation (PWM) to dim an LED.

Hardware Required
- CEL Universal Evaluation Board with LED Add-on Board or Ember Evaluation Board
- EM357 Module that plugs into CEL Universal Evaluation Board or EM357 Module that plugs into Ember Evaluation Board
- Ember ISA3 (for downloading)

Software Required
- Ember Insight Desktop Software version 2.5
- IAR ARM Workbench version 6.21
- Ember Stack version 4.6.2 or later
Table of Contents

Introduction 1
Hardware Required 1
Software Required 1
Features 1
Creating the Demo 3
   Ember Appbuilder 3
   ZigBee Cluster Configuration 4
   Stack Configuration 8
   HAL Configuration 9
   Plugins 10
   Callback Configuration 13
IAR Project and Modifying Code 15
   Modifying Plugin Code 16
      On/Off Plugin Code 16
      Level Control Plugin Code 16
   Modifying Appbuilder Generated Code 16
      Daintree_Ember_board_OTA_Dimmable_light.h 16
      Daintree_Ember_board_OTA_Dimmable_light_endpoint_config.h 17
      Daintree_Ember_board_OTA_Dimmable_light_callback.c 17
Ember Initialization Callback 17
Ember Application Tick Callback 17
Ember Stack Callbacks 17
Ember Stack Statuses 18
   emberAfStackStatusCallback 18
   emberAfPluginNetworkFindFinishedCallback 18
Incoming Message 18
   emberAfPreZDOMessageReceivedCallback 18
   emberAfCommissioningClusterResetStartupParametersCallback 18
   emberAfCommissioningClusterRestartDeviceCallback 18
Ember Cluster Specific Callbacks 19
   emberAfOnOffClusterServerInitCallback() 19
   emberAfCommissioningClusterServerTickCallback() 19
   emberAfLevelControlClusterServerAttributeChangedCallback() 19
Daintree Cluster Callbacks 19
   emberAfPluginDaintreeOtaInitialiseOTAttributesCallback() 19
   emberAfPluginDaintreeOtaWriteEepromCallback() 19
   emberAfPluginDaintreeOtaReadEepromCallback() 20
Hardware Abstraction Layer 20
   halInitDimmer() 20
   halDimLED0() 20
   halTimer1Isr() 20
References 21
Revision History 21
Creating the Demo

The software is based on the sample software created by the Ember Appbuilder application that is a part of the Ember Insight Desktop; some understanding of the Ember Insight Desktop environment, the ZigBee Home Automation Profile, ZigBee Pro specification and the ZigBee Cluster Library is assumed. Daintree supplies a proprietary cluster for their OTA firmware upgrade which must be installed prior to continuing. Contact Daintree for installation instructions.

**Ember Appbuilder**

The Daintree Dimmable Light Demo software is created from the Ember Appbuilder application. Using the appropriate installed stack version, the Appbuilder creates an IAR Project with framework code that supports the ZigBee Home Automation Profile for a dimmable light device. The steps to create the project are as follows:

- **Step 1** – Make sure the appropriate version of the Ember Stack you want to use is installed along with the latest version of the Ember Insight Desktop. The Ember Insight Desktop contains the Appbuilder application.

- **Step 2** – Configure the Appbuilder software to point to the appropriate Ember Stack version. Click “Add” to select/enter the directory where the stack version is installed. Once completed, you will see the stack version appear in the list. See screenshot below of the list of available stacks:
• **Step 3** – Select the stack version to use. The Daintree Dimmable Light Demo uses “EmberZNet 4.6 GA EM35X” stack selection.

![Installed stack selection](image)

• **Step 4** – Start configuring the Daintree Dimmable Light Demo using the Appbuilder. The following screenshots are from the Appbuilder software for the Daintree Dimmable Light Demo.

**ZigBee Cluster Configuration**

The ZigBee device type for a dimmable light should be a router so that the light can route within the mesh network.

There is only one endpoint: the Home Automation (Profile ID = 0x0104) Dimmable Light (Device ID = x0101).

The ZCL device type is normally a HA Dimmable Light. Daintree has some special cluster requirements in addition to those required for a HA Dimmable Light that forces a ZigBee Custom ZCL device type. A typical HA Dimmable Light requires the following clusters:

1. Basic Server Cluster
2. Identify Server Cluster
3. Groups Server Cluster
4. On/Off Server Cluster
5. Level Control Server Cluster

The Daintree Dimmable Light Demo requires the following additional clusters:

1. Commissioning Server Cluster
2. Daintree OTA Server Cluster
Each cluster has mandatory and optional attributes. Appbuilder highlights optional attributes with boxes in the attribute table. Column F shows the attribute settings that will be saved to the internal flash. Column S shows the attributes applied to all endpoints (this saves on storage space). Column B includes min/max boundaries. Daintree requires the following additional optional attributes to be supported (see screen-shots that follow).
Basic Cluster:

1. Application Version
2. Stack Version
3. Hardware Version
4. Manufacturer Name
5. Model Identifier
6. Date Code
7. Power Source
Level Control Cluster:

1. On level
Commissioning Cluster:

1. Short Address
2. Pan ID

Stack Configuration

Home Automation Security is selected, since the device is part of a Home Automation Network. The specification versions are all the latest versions by default.

The Home Automation profile specifies the radio channels which are automatically selected by Appbuilder. See the ZICM357xxx datasheet for maximum radio power levels vs. channel to meet agency certification requirements.

Set the Extended PAN ID to "Fixed" with all 0's (per Daintree requirement) and enable ZigBee Device Object messages for debugging ZDO messages.

Other settings include enable bindings, enable end device bind, enable receive statistics and concentrator support with Low RAM concentrator.

(Note: Not all routers are concentrators. It is recommended to select a few devices to be concentrators, as concentrators sends out many to one route requests, which increases traffic to the network. Low RAM concentrators do not store the route table which saves on RAM.)
Hardware Abstraction Layer Configuration

The HAL Configuration is for the Ember Evaluation Board. This same configuration works for the CEL Evaluation Board.

Take note that the bootloader option must be an Application bootloader to indicate to the application that there is an application bootloader loaded. Make sure the appropriate application bootloader is downloaded into the EM357 for the hardware used. For the Daintree Dimmable Light Demo, the demo can run on the CEL Universal Evaluation Board or the Ember Evaluation Board. Each board has a specific bootloader which is provided.

The application serial port is used for connecting a terminal emulator and is by default on Port1. Virtual UART is through the ISA3 box for Insight Desktop to open a serial connection.

The GPIO register configuration is per hardware settings; the only GPIO pins of concern are PA6 connected to the dimmable LED and PB6 for the push button. The defaults are per Ember Evaluation Board settings.
Plugins

The Ember supplied code can be used “as is” or with minor changes for certain features. The “as is” code is production ready. There is some code that requires “extending” (the user needs to add to the code for their specific use). These code samples are called “plugins”. By using a production ready plugin, you do not need to code in support for the feature. The Daintree Dimmable Light uses the following plugins:

1. Daintree OTA Server Cluster (“as is” code with no changes)
2. Fragmentation (“as is” code with no changes)
3. General Response Commands (“as is” code with no changes)
4. Groups Server Cluster (“as is” code with no changes)
5. Identify Cluster (“as is” code with no changes)
6. Level Control Server Cluster (needs “extending”)
7. Network Find (“as is” code with no changes)
8. On/Off Server Cluster (needs “extending”)
9. Reporting Cluster (“as is” code with no changes)
Some plugins require support in the form of “Callbacks”, where the developer can supply code to support hardware- or application-specific functions. These are in the form of “defined Callbacks”. Sometimes there is no need to fill in the “defined Callback”; by leaving the field empty the function will do nothing.

The plugins used that have callbacks needing code are as follows:

1. Daintree OTA Server Cluster
2. Network Find
3. Reporting Cluster

Callback Configuration

In addition to the “defined Callbacks” from the plugins, certain callbacks can be enabled to the application for special handling. Examples are “main init” for special hardware initialization, or “main tick” for checking for a button press. The Callback Configuration tab gives a list of available callbacks that can be exposed to the application level for special handling. Some of the callbacks are already used in the plugin and are not exposed. The “checked” boxes in the images on the following two pages are used in the Daintree Dimmable Light Demo.
• **Step 5** – Generating the IAR Project and base source code for modification.

Click “Generate” to create a list of files and an IAR Project under the directory `{app\builder\Daintree_Ember_board_OTA_Dimmable_light}` for this particular demo code. The directory name comes from the “device name” in the Appbuilder GUI.

Under the `{app\builder\Daintree_Ember_board_OTA_Dimmable_light}` directory, the generated application level source code and IAR Project start with the device name, `Daintree_Ember_board_OTA_Dimmable_light`.

**IAR Project and Modifying Code**

Once the IAR Project has been created, open the IAR Project `{.eww file}` to start modifying code. Normally, all file changes are limited to the `Daintree_Ember_board_OTA_Dimmable_light_callback.c` file. Since the Demo uses plugins that require enhancing, changes will need to be made to those files.

*Note: Refer to the Ember Application Framework V2 Developer Guide for more detailed information on the structure of the software and how the callbacks interact with the software generated.*
The files that will need to be changed for the Daintree Dimmable Light Demo are as follows:

1. app\framework\plugin\on-off\on-off.c
2. app\framework\plugin\level-control\level-control.c
3. app\builder\Daintree_Ember_board_OTA_Dimmable_light\Daintree_Ember_board_OTA_Dimmable_light.h
4. app\builder\Daintree_Ember_board_OTA_Dimmable_light\Daintree_Ember_board_OTA_Dimmable_light_callback.c
5. app\builder\Daintree_Ember_board_OTA_Dimmable_light\Daintree_Ember_board_OTA_Dimmable_light_endpoint_config.h

Modifying Plugin Code

The two plugins that require enhancing are the On/Off and Level Control plugins. All plugin code is found in the app\framework\plugin directory.

On/Off Plugin Code

The On/Off plugin code is found in the app\framework\plugin\on-off subdirectory. The main file for the code change is on-off.c. This file handles the entire cluster of On/Off commands that were received and parsed. This file was enhanced to include code to physically turn on or off the LED hardware when handling the on/off commands received. CEL used an API supplied by the developer to turn the light on/off based on dimming called halDimLED0().

*Note: this code interacts with the Level Control cluster ZCL_USING_LEVEL_CONTROL_CLUSTER_SERVER and is defined to build in the Level Control Cluster Code.*

Level Control Plugin Code

The Level Control plugin code is found under the app\framework\plugin\level-control subdirectory; level-control.c is the main file for the code change. This file handles the entire cluster Level Control Cluster commands that are received and parsed.

The level-control.c code was enhanced to “write” the new level attribute setting to be used in NVRAM when turning on the light. The light will then turn on at the level set by the control next the time an ON command is received.

Modifying Appbuilder Generated Code

Appbuilder generates an application layer Callback file in the directory app\builder\Daintree_Ember_board_OTA_Dimmable_light for the developer to add in their specific code changes. The callback APIs are the ones selected from the Appbuilder Plugin and Callback tabs. The file is named Daintree_Ember_board_OTA_Dimmable_light_callback.c. It uses the generated supporting .h files Daintree_Ember_board_OTA_Dimmable_light.h and Daintree_Ember_board_OTA_Dimmable_light_endpoint_config.h.

Daintree_Ember_board_OTA_Dimmable_light.h

This file holds all #defines that enable cluster support, plugin support, the Ember Stack features and library features for the base application. The only #defines that need to be added are the following:
1. `EMBER_APPLICATION_HANDLES_UNSUPPORTED_ZDO_REQUESTS`: Handles the optional ZDO requests.
2. `EMBER_AF_ENABLE_CUSTOM_COMMANDS`: Adds new CLI commands under “custom” keyword.
3. `EMBER_APPLICATION_HAS_BUTTON_HANDLER`: Uses a push button and supplies the ISR handler.
4. `EMBER_AF_ENABLE_FRAMEWORK_EEPROM_INTERFACE`: Uses an EEPROM.

`Daintree_Ember_board_OTA_Dimmable_light_endpoint_config.h`

This file contains default values for the clusters. The change here is for startup defaults that are different from the ZigBee specification. Specifically, the commissioning cluster startup control attribute should be 3 according to Daintree requirements. The comments will guide you to a correct start up attribute.

`Daintree_Ember_board_OTA_Dimmable_light_callback.c`

This file is where all callbacks reside. Any additional support functions are also included in this file.

Types of callbacks used by the Dimmable Light:

1. Ember Initialization Callbacks – Used by the application for software structure or hardware initialization.
2. Ember Application Tick Callback – Application usage to monitor push buttons, blink LED, etc.
3. Ember Stack Callbacks – Used to handle ZigBee Application incoming messages, network status or transmit status.
4. Ember Cluster Specific Callback – Application can execute special handling if needed for the specific cluster.

**Ember Initialization Callback**

The Ember Initialization callback used is `emberAfMainInitCallback()`. This callback allows the developer to do an initialization such as initializing hardware, software structures, etc. The EEPROM and dimmer were initialized within this callback.

**Ember Application Tick Callback**

The Ember Application Tick Callback used is `emberAfMainTickCallback()`. This callback is the developer’s hook into the Ember Application Tick. The Ember Application Tick is not based on any timer tick, but is more of an application poll. This callback is used for continuous checking of a button press and to start the process of searching for a network to join by starting the A-mode commissioning state machine.

The A-mode commissioning initial state is `NOT_JOINED`. Within this callback the state changes to `SEARCHING_FOR_NETWORK`, if no network is joined. If the stack indicates that a node is part of a network, the state is changed to `JOINED`.

There are three states defined for A-mode commissioning:

1. `NOT_JOINED` – Node has not joined a PAN.
2. `SEARCHING_FOR_NETWORK` – Node searching all channels looking for a PAN to join.
3. `JOINED` – Node has joined a PAN.

**Ember Stack Callbacks**

The Ember Stack can generate three types of callbacks:
1. Send – For the developer to do application specific handling once a message has been sent and the Stack has returned the status of the send.
2. Incoming Message Received – For the developer to handle the message before it is passed up to the Ember application handling.
3. Stack Status – For the developer to handle any network status condition.

The Daintree Dimmable Light Demo is only concerned with the Incoming Message type and the Stack Status type.

**Ember Stack Statuses**

**emberAfStackStatusCallback**

The callback `emberAfStackStatusCallback()` handles the status returned by the stack. The Daintree Dimmable Light Demo is only interested in the following statuses:

- Down – Updates A-mode commissioning state machine to `NOT_JOINED`.
- Joined – Updates A-mode commissioning state machine to `JOINED`.
- Channel Changed – Prints out message to indicate the new channel of the network.

**emberAfPluginNetworkFindFinishedCallback**

The callback `emberAfPluginNetworkFindFinishedCallback()` is called as a result of a joined complete notification. This notification allows the developer to do any specific handling as a result of the node joining the network. The Daintree Dimmable Light Demo will update the A-mode commissioning state machine to the state of `JOINED` here.

**Incoming Message**

**emberAfPreZDOMessageReceivedCallback**

The callback `emberAfPreZDOMessageReceivedCallback()` passes the application an incoming ZDO message and gives the application the opportunity to handle it. The stack automatically handles mandatory ZDO commands but can pass up optional commands to the Application layer. The Daintree Dimmable Light Demo handles the optional command `USER_DESCRIPTOR_REQUEST` in this callback by creating the response message with the user descriptor info in it and sends it out via unicast.

**emberAfCommissioningClusterResetStartupParametersCallback**

The callback `emberAfCommissioningClusterResetStartupParametersCallback()` is called when a packet is received, parsed and determined to be a Commissioning Reset Startup Command. This callback allows the developer to handle the command as though there is no usable plugin provided that is spec compliant. The Daintree Dimmable Light Demo handles this callback based on the interpretation of the ZigBee Commissioning Cluster recommendations.

**emberAfCommissioningClusterRestartDeviceCallback**

The callback `emberAfCommissioningClusterRestartDeviceCallback()` is called when a packet is received, parsed and determined to be a Commissioning Restart Device Command. This callback allows the developer to handle the command as though there is no usable plugin provided that is spec compliant. The Daintree Dimmable Light Demo handles this callback based on the interpretation of the ZigBee Commissioning Cluster recommendations which is to restart the device however the developer wishes.
Ember Cluster Specific Callbacks

Cluster-specific callbacks are for the developer to do any custom handling in regard to a cluster. The Daintree Dimmable Light Demo uses the following cluster specific callbacks:

1. emberAfOnOffClusterServerInitCallback()
2. emberAfCommissioningClusterServerTickCallback()
3. emberAfLevelControlClusterServerAttributeChangedCallback()

emberAfOnOffClusterServerInitCallback()

The callback emberAfOnOffClusterServerInitCallback() gives the application a chance to do some initialization for the On/Off Cluster. The Daintree Dimmable Light Demo turns on the LED light if the attribute saved is the ON state.

emberAfCommissioningClusterServerTickCallback()

The callback emberAfCommissioningClusterServerTickCallback() is the Commissioning Cluster-specific application tick. The Daintree Dimmable Light Demo uses this callback for the following:

1. To handle delayed events for restarting the device at a later time.
2. To handle updating the A-mode Commissioning state machine. This tick callback takes over the A-mode commissioning state machine from the main tick callback once the Commissioning Cluster starts running.

To restart a device, the Daintree Dimmable Light Demo checks the Commissioning Cluster startup attribute and acts according to its setting per the ZigBee Commissioning Cluster specification.

Note: there are other ways to handle restarting a device; the Server Tick Callback is just one example.

emberAfLevelControlClusterServerAttributeChangedCallback()

The callback emberAfLevelControlClusterServerAttributeChangedCallback() is the Level Control Callback to the application for handling an attribute change. Whenever the level control attribute is written with a new value, this callback is generated. The Daintree Dimmable Light Demo sets the LED brightness as a centralized location using a developer supplied API HalDimLED0.

Daintree Cluster Callbacks

Daintree has defined callbacks that need to be supplied by the developer. They are as follows:

1. emberAfPluginDaintreeOtaInitialiseOTAattributesCallback()
2. emberAfPluginDaintreeOtaWriteEepromCallback()
3. emberAfPluginDaintreeOtaReadEepromCallback()

emberAfPluginDaintreeOtaInitialiseOTAattributesCallback()

The callback emberAfPluginDaintreeOtaInitialiseOTAattributesCallback() initializes the required attributes: Daintree HW ID, Daintree Version string and the Basic Cluster Manufacturer Name Attribute.

emberAfPluginDaintreeOtaWriteEepromCallback()

The callback emberAfPluginDaintreeOtaWriteEepromCallback() writes data to EEPROM using Ember API, emAfEepromWrite.
emberAfPluginDaintreeOtaReadEepromCallback()

The callback `emberAfPluginDaintreeOtaReadEepromCallback()` reads data from EEPROM using Ember API `emAfEepromRead`.

**Hardware Abstraction Layer**

The following three HAL APIs need to be added by the developer to support light dimming:

1. `halInitDimmer()`
2. `halDimLED0(level)`
3. `halTimer1Isr()`

`halInitDimmer()`

The API `halInitDimmer()` initializes the EM357 PWM according to the GPIO for LED0 as follows:

1. Use the 12 Mhz PCLK with a divide by 64 setting.
2. Re-init counter and generate update event to interrupt the controller.
3. PWM runs in Mode 1 (see EM357 data sheet) and counts up.
4. Enable buffer request.
5. PWM frequency = 255.
6. PWM duty cycle = 0 (will change as the lights go up or down).
7. Active Low polarity.
8. Auto reload buffer reg enable, edge-aligned, generate interrupt on counter overflow.
9. Init counter = 0.
10. Enable interrupts to timer1.
11. Start the counter to generate a PWM signal.

`halDimLED0()`

The API `halDimLED0()` sets the duty cycle value for the next PWM interrupt. The range is 0 – 254.

`halTimer1Isr()`

The developer-supplied interrupt service routine `halTimer1Isr()` is the PWM interrupt handler that updates the EM357 PWM duty cycle register with a duty cycle value (to do the actual dimming). The duty cycle is set by `halDimLED0`. 
Daintree Dimmable Light Demo

References

**Daintree**

Daintree Networks ZigBee Device Requirements Rev 0.8

**Ember**

1. 120-3028-000 Application Framework V2 Developer Guide
2. 120-3029-000 App Dev Fundamentals

**ZigBee**

1. IEEE Standard 802.15.4-2003
2. ZigBee Home Automation Profile Specification Version 1.2 Revision 29
3. ZigBee Cluster Library Specification Revision 3
4. ZigBee Specification Revision 19

Revision History

<table>
<thead>
<tr>
<th>Previous Versions</th>
<th>Changes to Current Version</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Issue A) August 27, 2012</td>
<td>Initial preliminary document</td>
<td>N/A</td>
</tr>
</tbody>
</table>