Smart Energy
Sample Application User's Guide

Document Number: SWRU215

Texas Instruments, Inc.
San Diego, California USA

Revision	Description	Date
1.0	Initial release.	04/06/2009

Table of Contents

Table of Figures

1. Introduction

ZigBee Smart Energy (SE) is one of the public application profiles released for the ZigBee 2007 specification. It enables utility companies and their customers to directly communicate with thermostats and other smart appliances; see www.zigbee.org for more information.

The Smart Energy Sample application (part of the Z-Stack installer for ZigBee 2007; covering ZigBee and ZigBee PRO) is the optimal starting point to build your own SE application on top of Texas Instruments' Z-Stack (www.ti.com/z-stack).

1.1. Scope

This document describes how to use the Smart Energy Sample Application and discusses its theory of operation. For a more general description of Smart Energy, the reader is referred to the Zigbee Smart Energy specification available from www.zigbee.org. The reader should also review the Z-Stack Smart Energy Developer's Guide prior to using this document.

There are seven defined application instances within the IAR project:

a. Energy Service Portal (ESP) as a Coordinator
b. Metering Device as a Router and also as an End Device
c. In Premise Display as an End Device
d. Programmable Communicating Thermostat (PCT) as an End Device
e. Load Control Device as a Router
f. Range Extender as a Router

Figure 1 shows the usage model of how these sample application instances interact with the ESP.

1.2. Definitions, Abbreviations, Acronyms

Term	Definition
CBKE	Certificate-based Key Establishment
DUT	Device Under Test
ECC	Elliptic Curve Cryptography
HAN	Home Area Network
SE	Smart Energy
ZCL	ZigBee Cluster Library
ESP	Electronic Service Portal
PCT	Programmable Communicating Thermostat

IPD	In Premise Display
AF	Application Framework
ZDO	Zigbee Device Object
OSAL	Operating System Abstraction Layer

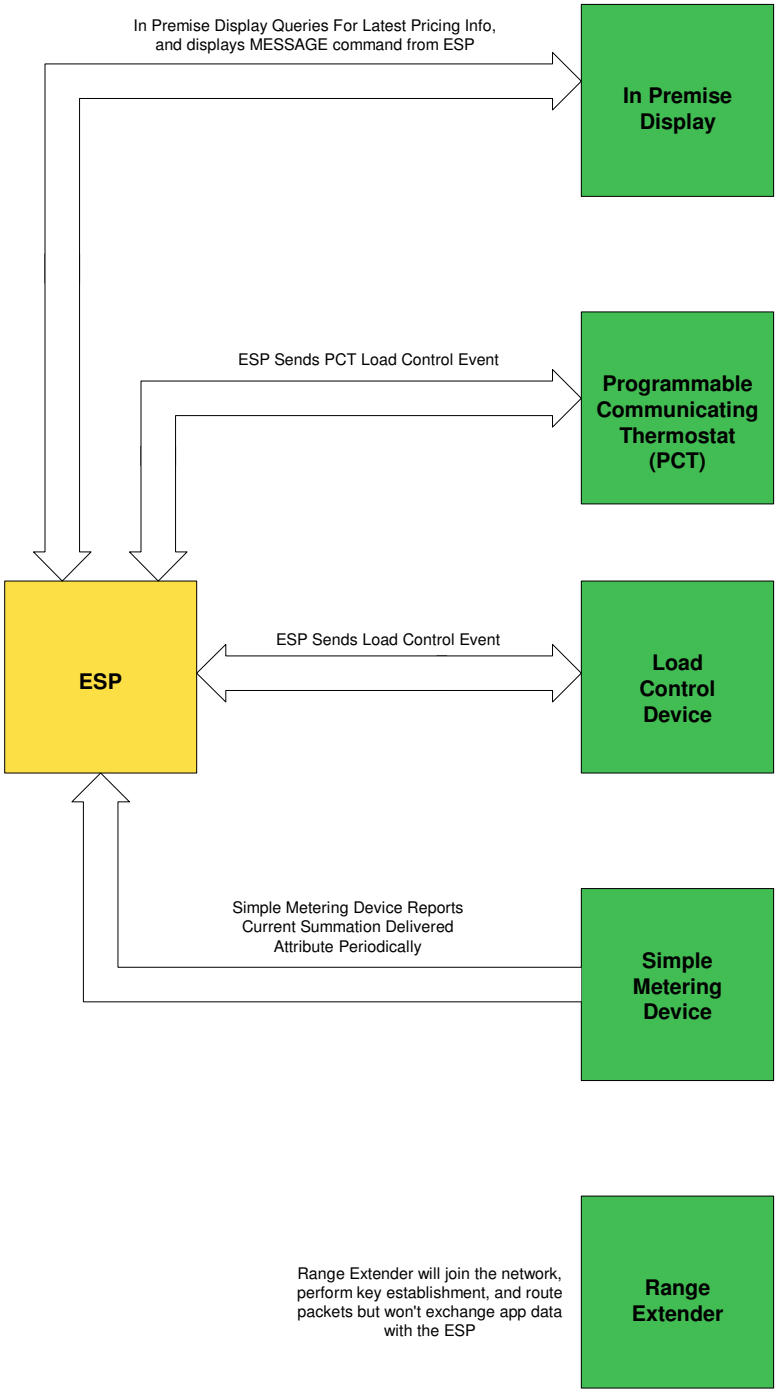


Figure 1: System Context Diagram

2. Setup

2.1. Required Tools

The tools that will be needed to evaluate this sample app and build your own application based on it are the following:

a. IAR Embedded Workbench EW8051 7.51 (2530) or EW430 4.20 (MSP430)

b. Texas Instruments Flash Programmer Tool (to obtain drivers for the SmartRF05EB board)

c. Daintree Networks Sensor Network Analyzer (Standard version is the minimum requirement)

d. Z-Tool 2.0 (provided as part of the Z-Stack install)

e. Z-Converter – A tool used to transform Certicom certificates data into arrays that can easily be imported into a Z-Tool script (Provided as part of the Z-Stack install)

f. Certicom ECC library (please contact Certicom directly to obtain a license) IF using security

2.2. Using Z-Converter to Transform Certicom Certificates

Z-Converter takes Certicom certificate data as input in the following format (the actual input requires no carriage returns in order for Z-Converter to process the data correctly):

IEEE Address: 00124b0000000001

CA Pub Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8

Device Implicit Cert:
0204ac2c2656f1eea4ff5dac4edda176bfe4fa70d95600124b0000000001544553545345434101090001000001091003

Device Private Key: 00f035a9f731f265530ad5c1202562d56d1b822543

Device Public Key: 0202f71c27abfd28eb39e0b4a718ace4cf374559a6f6

This data must be entered line by line as shown above with no carriage returns into a text file, and the user inputs this data using the "Load" button. Z-Converter then transforms this data into an array output as follows:

IEEE Address: 0x01,0x00,0x00,0x00,0x00,0x4b,0x12,0x00

CA Pub Key: 0x02,0x00,0xfd,0xe8,0xa7,0xf3,0xd1,0x08,0x42,0x24,0x96,0x2a,0x4e,0x7c,0x54,0xe6,0x9a, 0xc3,0xf0,0x4d,0xa6,0xb8

Device Implicit Cert: 0x02,0x04,0xac,0x2c,0x26,0x56,0xf1,0xee,0xa4,0xff,0x5d,0xac,0x4e,0xdd,0xa1,0x76,0xbf, 0xe4,0xfa,0x70,0xd9,0x56,0x00,0x12,0x4b,0x00,0x00,0x00,0x00,0x01,0x54,0x45,0x53,0x54, 0x53,0x45,0x43,0x41,0x01,0x09,0x00,0x01,0x00,0x00,0x01,0x09,0x10,0x03

Device Private Key: 0x00,0xf0,0x35,0xa9,0xf7,0x31,0xf2,0x65,0x53,0x0a,0xd5,0xc1,0x20,0x25,0x62,0xd5,0x6d, 0x1b,0x82,0x25,0x43

Device Public Key:
0x02,0x02,0xf7,0x1c,0x27,0xab,0xfd,0x28,0xeb,0x39,0xe0,0xb4,0xa7,0x18,0xac,0xe4,0xcf,0x37,0x45,0x59,0xa6,0xf6

Note that the Device Public Key is not used as part of the input into the Certicom library but is provided for completeness.

The user then copies these values into a Z-Tool script that uses arrays such as the following:

var ieee = [**0x01,0x00,0x00,0x00,0x00,0x4b,0x12,0x00**];

var data0x69 = [**0x02,0x04,0xac,0x2c,0x26,0x56,0xf1,0xee,0xa4,0xff,0x5d,0xac,0x4e,0xdd,0xa1,0x76,0xbf, 0xe4,0xfa,0x70,0xd9,0x56,0x00,0x12,0x4b,0x00,0x00,0x00,0x00,0x01,0x54,0x45,0x53,0x54, 0x53,0x45,0x43,0x41,0x01,0x09,0x00,0x01,0x00,0x00,0x01,0x09,0x10,0x03**];

var data0x6a = [**0x00,0xf0,0x35,0xa9,0xf7,0x31,0xf2,0x65,0x53,0x0a,0xd5,0xc1,0x20,0x25,0x62,0xd5,0x6d, 0x1b,0x82,0x25,0x43**];

var data0x6b = [**0x02,0x00,0xfd,0xe8,0xa7,0xf3,0xd1,0x08,0x42,0x24,0x96,0x2a,0x4e,0x7c,0x54,0xe6,0x9a, 0xc3,0xf0,0x4d,0xa6,0xb8**];

The mapping of the labels from the Certicom certificate to the Z-Tool script is:

IEEE -> var ieee
Device Implicit Cert -> var data 0x69
Device Private Key -> var data 0x6a
CA Pub Key -> var data0x6b

Figure 2 provides a screenshot of the Z-Converter graphical interface. The user can also save the output into a text file for later use using the "Save As" button.

Figure 2. Z-Converter Graphical Interface

Z-Tool is used to program the Certicom Certificate information into Z-stack Non-Volatile memory. A serial cable is required to allow Z-Tool to communicate with the stack via the DB-9 connector on the SmartRF05EB board. The MSP5438 Experimenter's board uses the mini-USB connector with the supported virtual COM port driver available from the TI website.

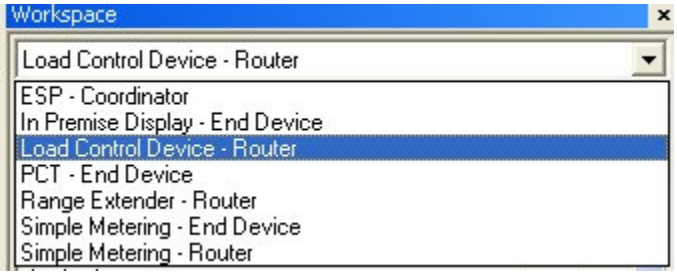
2.3. Supported Hardware Platforms

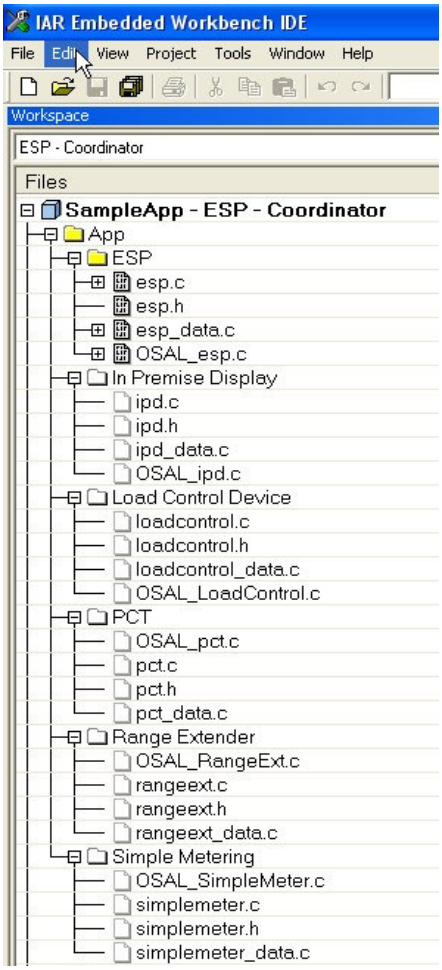
a. SmartRF05EB/BB boards with CC2530EM
b. SmartRF05EB boards with CCMSP2618 + CC2520EM
c. MSP5438 Experimenter boards with CC2520EM

3. Getting Started

3.1. Building the ESP, PCT, and IPD Application Instances

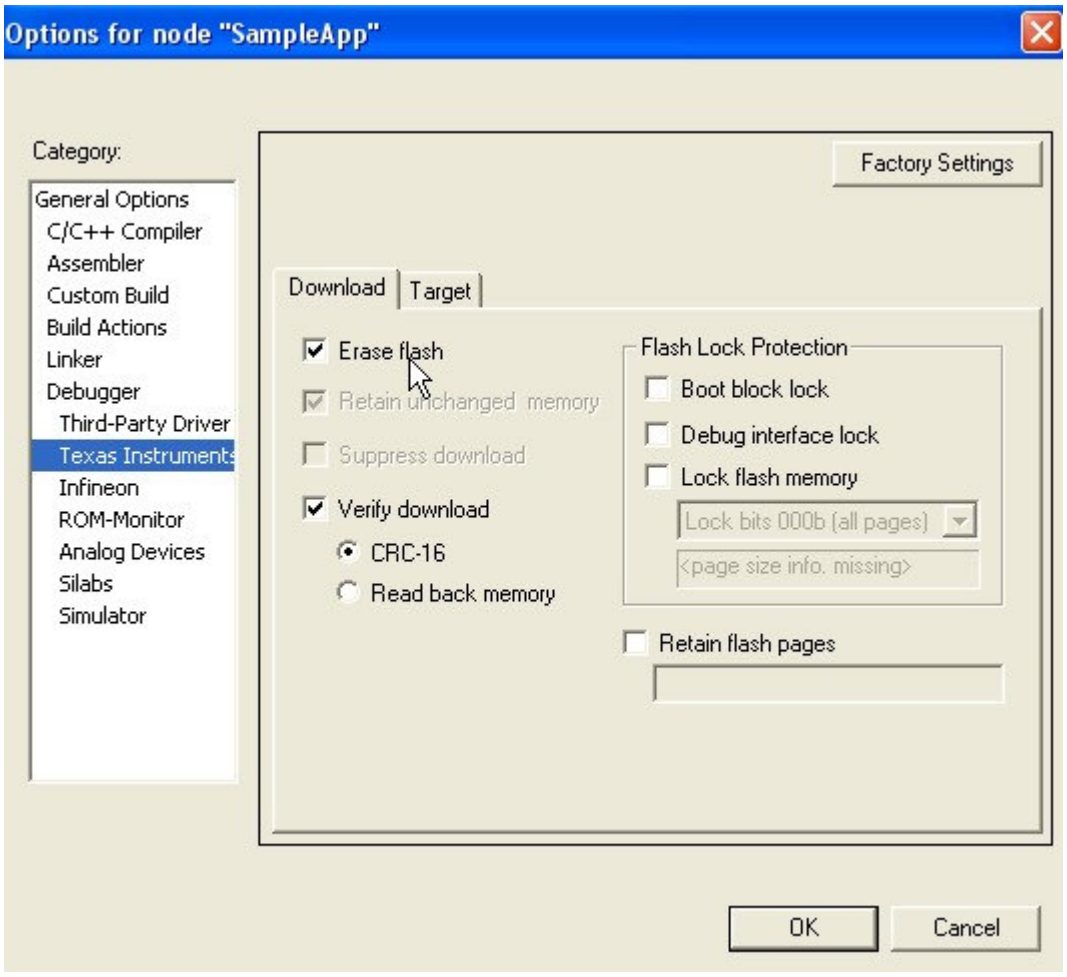
The Smart Energy sample application project is located in C:\Texas Instruments\ZStack-2.2.0-1.3.0\Projects\zstack\SE\SampleApp\<Target> (e.g. <Target> = CC2530DB). Open the SampleApp.eww project file. The figures below show how each configuration is organized. For each device configuration, only the relevant application files are brought in for each device configuration.





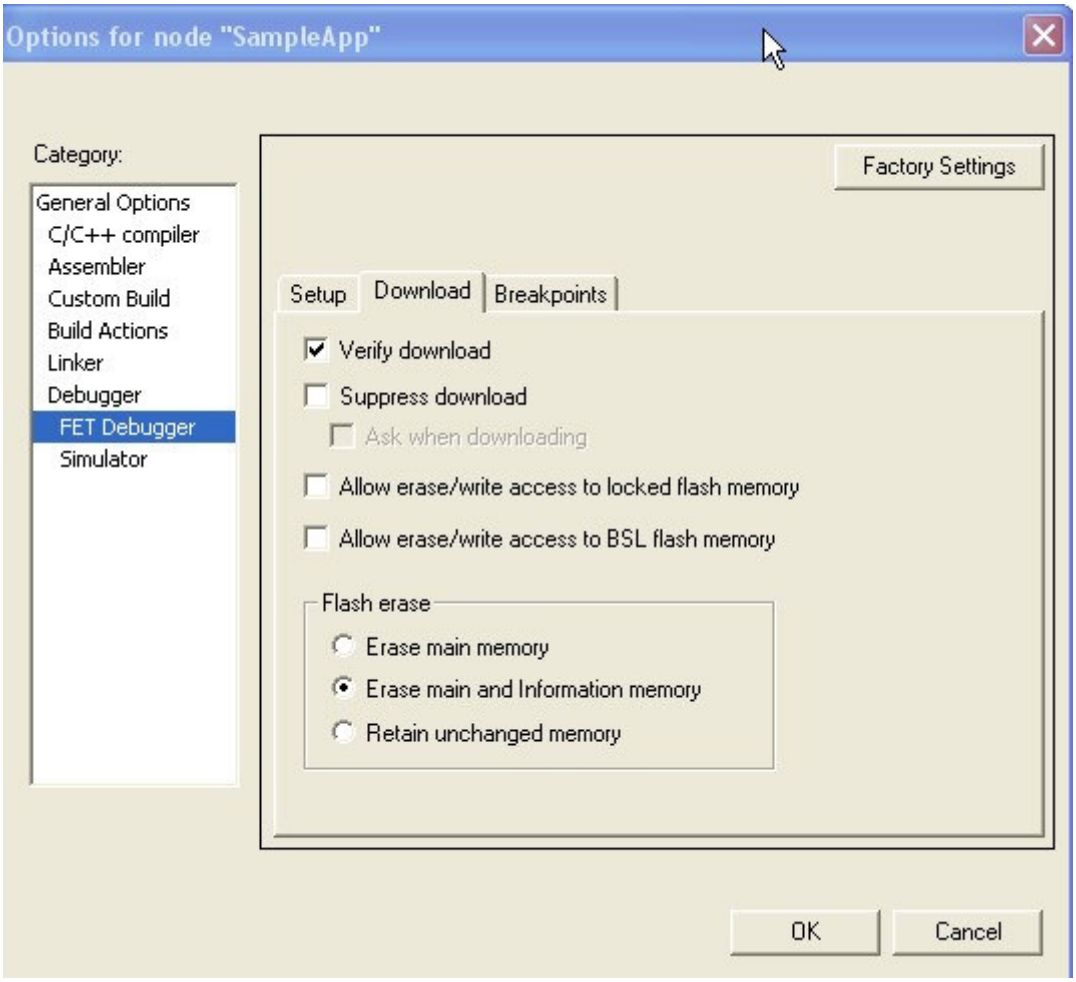
Build the "ESP – Coordinator", "PCT – End Device", and "In Premise Display – End Device" configurations.

1. If the SECURE=1 compile option is used, before rebuilding the project, take the Certicom library and rename it to ecc.r51 and drop it into C:\Texas Instruments\ZStack-2.2.0-1.3.0\Projects\zstack\Libraries\TI2530DB\bin
2. The NWK_INDIRECT_MSG_TIMEOUT parameter in the f8wConfig.cfg file must be increased to a suggested value of 10 in order to buffer the message long enough to accommodate the 8 second poll period by the end device. The MAX_POLL_FAILURE_RETRIES parameter should also be set to 4 as during the CBKE procedure the ESP will be somewhat unresponsive to data requests during this time.
3. Select Project -> Rebuild All.
4. Highlight the configuration desired and go to Project -> Options and select the "Erase flash" option as shown for the CC2530 project.



This allows one to start with a clean slate before programming Certicom certs. After the device has been configured, unselect this box so you can retain the IEEE address and key information when downloading new code each time.

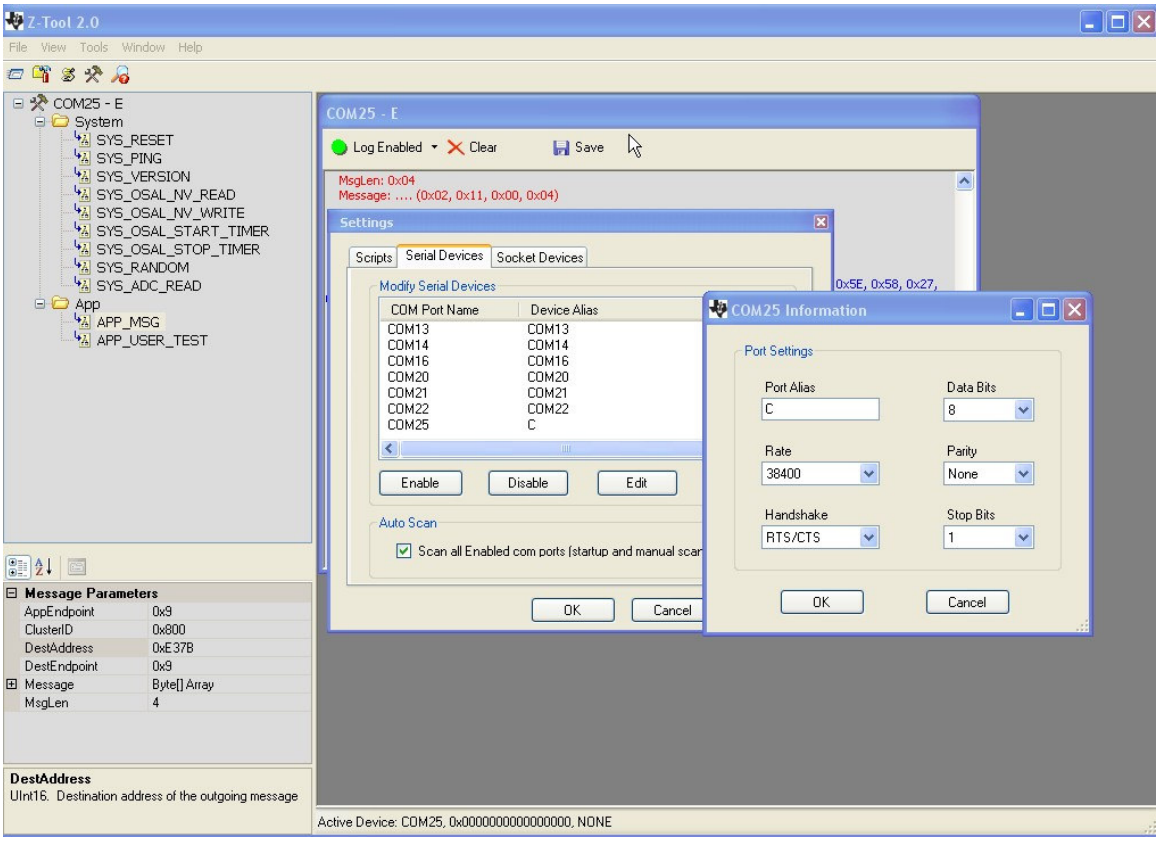
For MSP projects, select “Erase main and information memory” as shown:



5. Download the program by clicking on the debug icon or by going to Project -> Debug. Hit the Run button, and watch the coordinator start up, or if it's a device, it will blink its LED waiting for you to press SW1 to join an existing network.

3.2. Configuring Certicom Keys Using Z-Tool

In order to configure the IEEE address and Certicom keys for each device, use the provided Z-Tools scripts and Z-Tool as shown in the following screen captures:



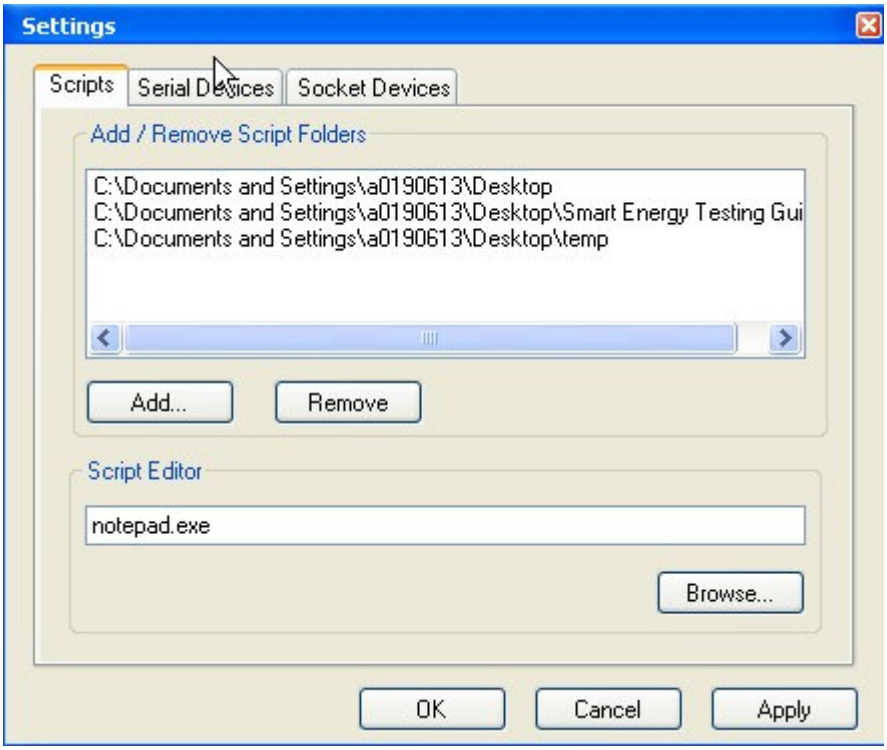
Open Z-Tool from the Texas Instruments start menu links and select Tools -> Settings to configure the baud rate, Port Alias, and Handshake protocol as shown above.

Then as shown in the figure below, go to the Settings -> Scripts tab to import the scripts. Point it to the folder where the scripts are located by clicking on add.

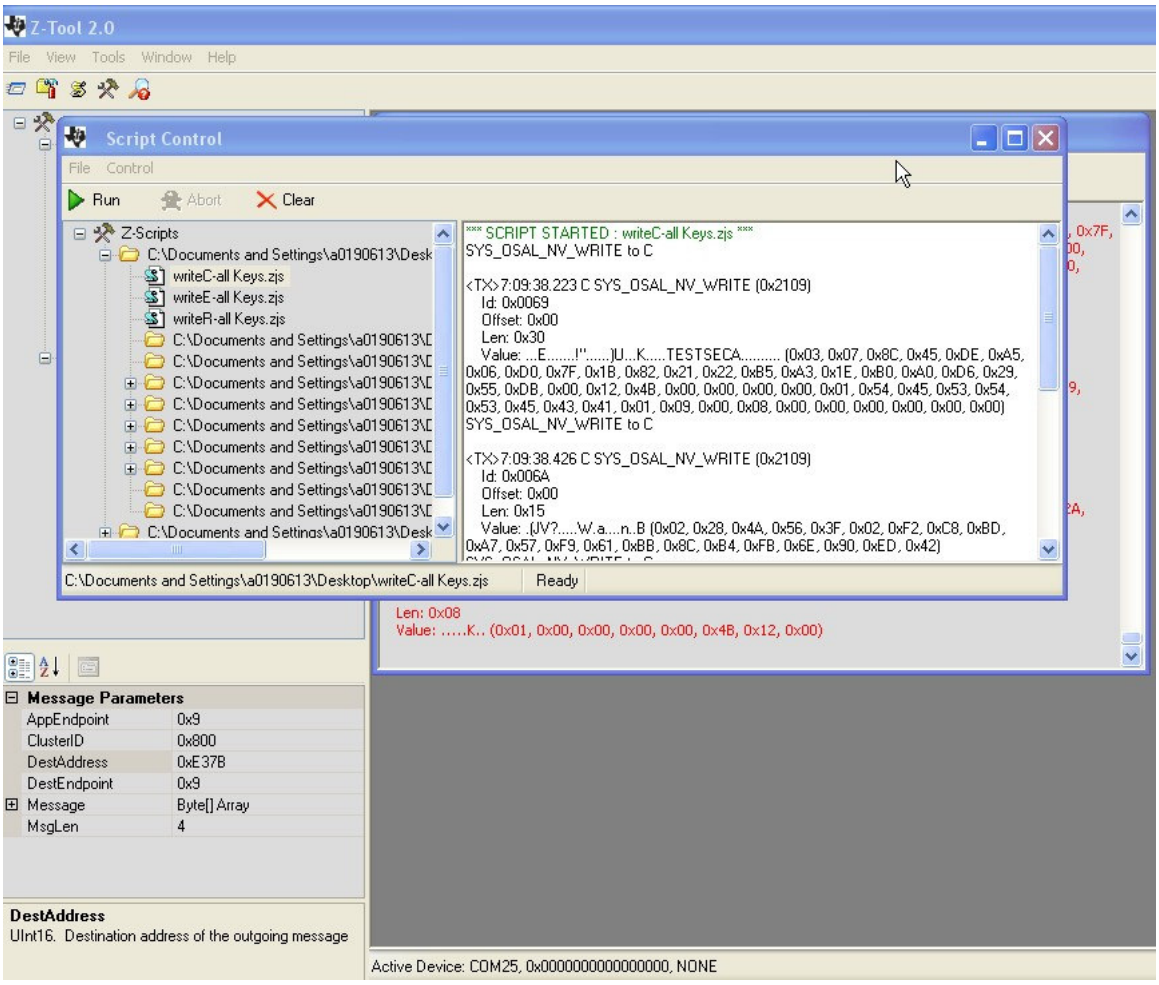
There are already pre-made Z-Tool scripts which you can use in the folder "Z-Tool scripts" in ZStack-2.2.0-1.3.0\Tools\Z-Converter. There is one for each type of device, coordinator, router, and end device.

Coordinator = IEEE address ending with 01
Router = IEEE address ending with 02
End Device = IEEE address ending with 03

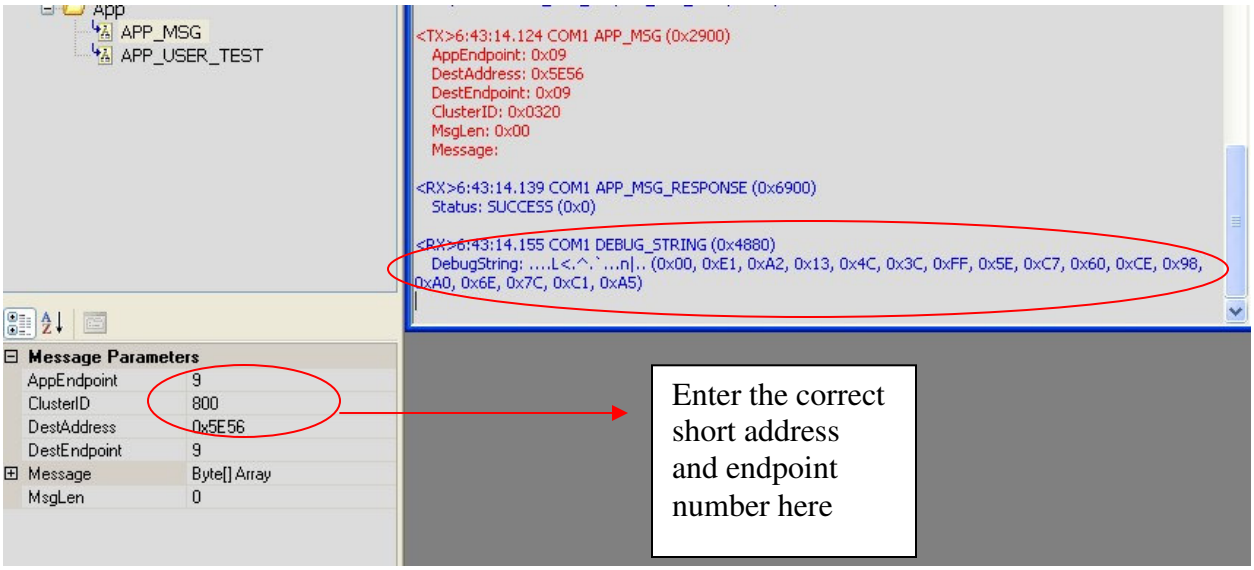
Of course, there is no reason that you couldn't interchangeably use the scripts, but using it in this order makes it easier to keep track of configuration of certs.



Then, go to Tools -> Script Control and execute the script called “writeC-all Keys.zjs” by highlighting it and clicking on the RUN button.

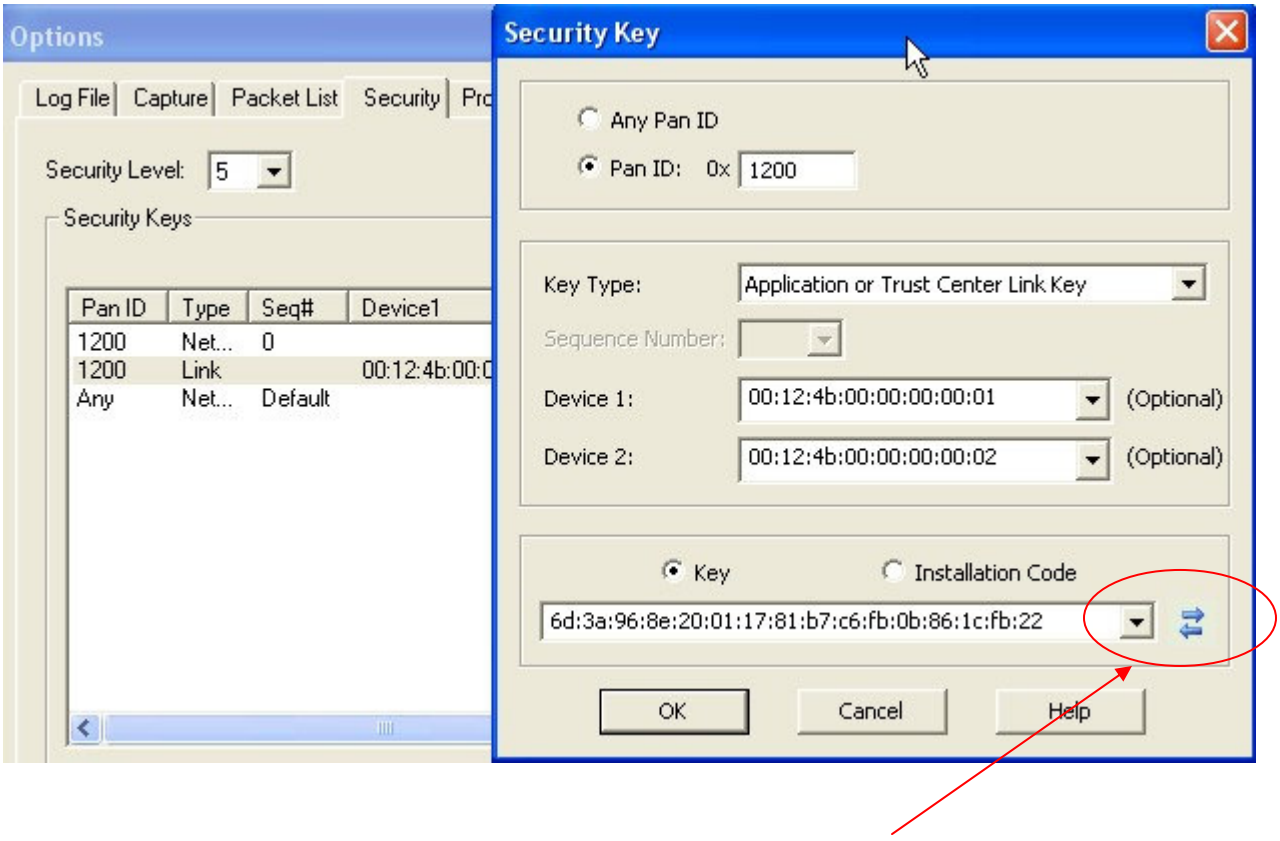


Repeat this procedure for each device you wish to configure with the IEEE address and certicom keys. Once key establishment is successful, use the APP_MSG command to extract the link key established for the device. This must be run on the ESP device in order to get the key used by the pair of devices (needed for the packet sniffer as described below). Configure the APP_MSG command as shown below. The only value that changes is the Dest Address, which will depend on what short address your device is assigned (this can be verified using the packet sniffer).



The first byte returned in the DEBUG_STRING is the status (0x00) which means it was a success. The bytes that follow are the key, which must be entered into Daintree in reverse order. The best way to format this key is to copy and paste it into notepad, and just replace "0x" with blank space, then "," with blank space. If the link key extraction is not successful, the string of bytes (16 total) will not be displayed.

Copy and paste the key into Daintree then reverse it using the "arrows button"



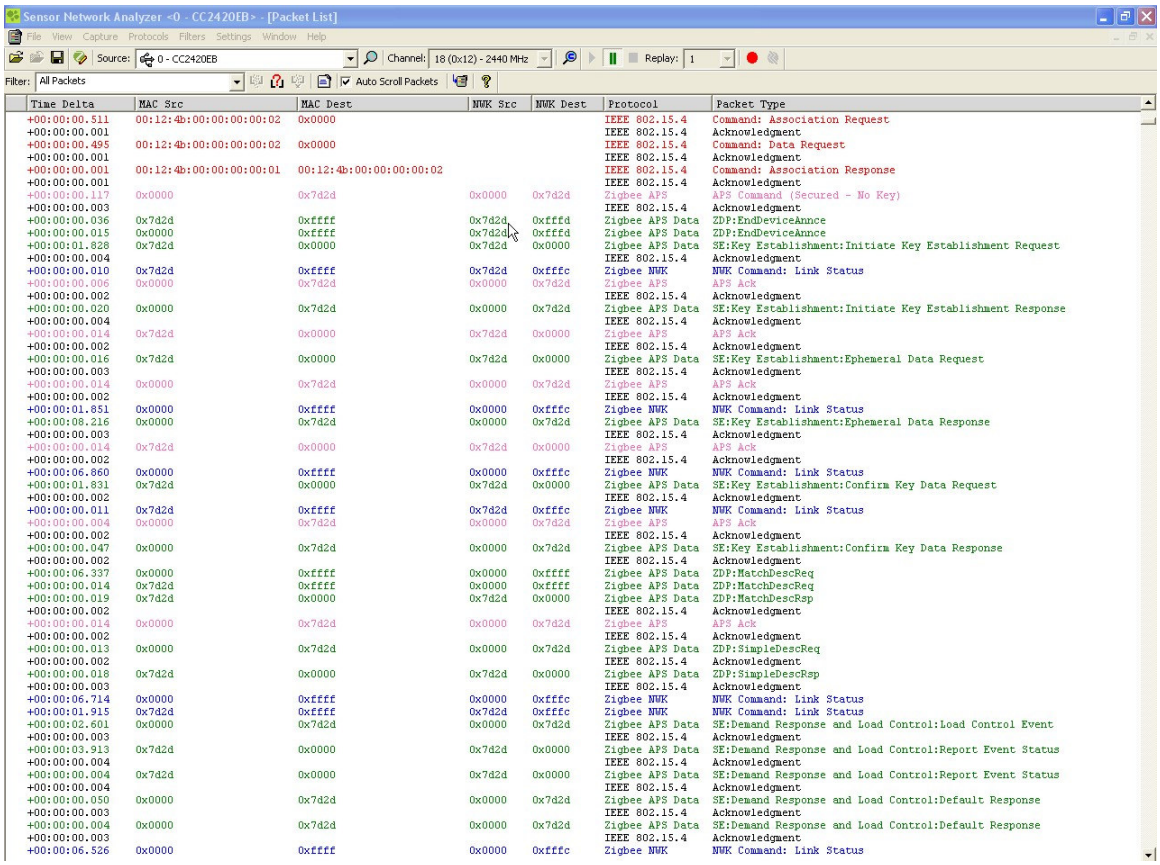
3.3. Running the ESP, PCT, and IPD Applications

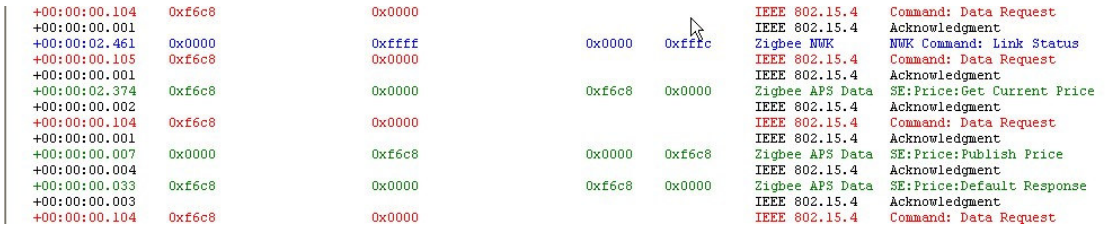
After compiling these sample app instances and programming the devices one can start these applications as follows:

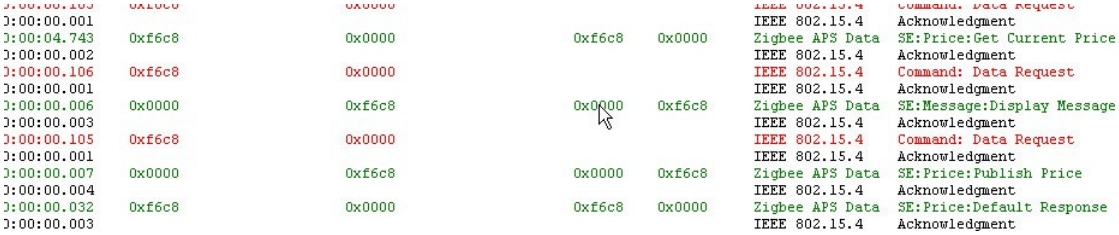
1. Power the coordinator, wait for "Zigbee Coord" to show up on the LCD display. The 2nd line of the LCD will display the PAN ID.
2. Then power the PCT device. The LED will flash, waiting for you to press SW1 which is joystick UP. This will cause the device to join the network. Verify with the sniffer that key establishment is complete. Because it's polling rate is set to 8secs, the key establishment procedure take a lot longer. Wait about 20 seconds for this to complete. Verify with the sniffer that the end device received the confirm key response. Then, hit SW4 (joystick to the left) on the coordinator to discover the PCT device. Verify on the SNA that the simple descriptor response is received by the coordinator. Then, press SW2 (joystick to the right) to send a PCT event to the PCT device. The PCT device will flash it's LED and display "PCT Evt Started". When it completes the PCT event (which is for 1 min), then it will stop flashing its LED and display "PCT Evt Complete".
3. Power up the IPD end device. Hit SW1 to let it join the network. Because it's polling rate is set to 8secs, the key establishment procedure take a lot longer. Wait about 20 seconds for this to complete. Verify with the sniffer that the end device received the confirm key response. Then, every 5 seconds it will get the pricing info and display the provider id. On the coord, hit SW3 (joystick down) to send a display message to the IPD. "Rcvd MESSAGE Cmd" should be displayed.

Note: NV_RESTORE is **not** turned on by default, so if you power cycle any of the devices, it needs to go through the key establishment procedure again, and a different link key would be used. NV_RESTORE is turned off by default to allow rapid development. Otherwise you would have to erash flash each time you downloaded new code to the device. Also, any device joining the ESP wouldn't be able to join more than once since the new link key is used for authenticating the device instead of the default TC link key.

The following show what happens in the Daintree SNA for these application instances.







4. Theory of Operation

This section explains how each application instance behaves with respect to the ESP device and also discusses the design of each application instance.

4.1. SE Secure Joining

The default trust center link key (TC link key) is used to commission each device. Each device uses the SE secure joining process. See the Z-stack Smart Energy Developer's Guide for more details.

4.2. Key Establishment

Upon successfully joining the network using security, each device will initiate key establishment with the ESP. Certificate information required to perform the key establishment is entered in ahead of time using Z-Tool. Figure 3 shows a flowchart of the device startup logic. The ZDO_STATE_CHANGE event is an OSAL system message that is provided to the application to indicate that the device has successfully started or joined the network.

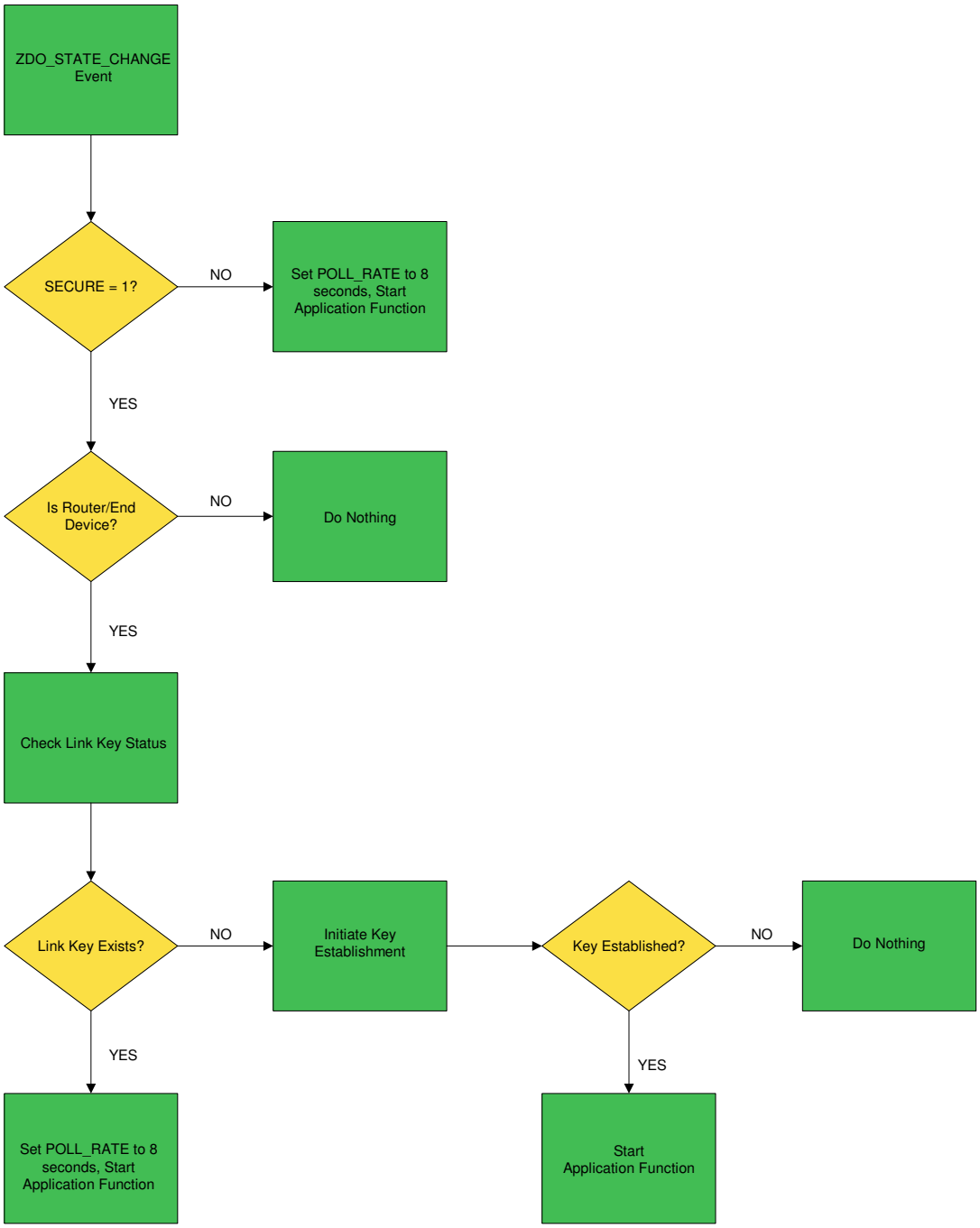


Figure 3. Flowchart of Device Startup Logic

4.3. Device and Service Discovery

Each device communicates with the ESP. Since the ESP is the coordinator and will always have a short address of 0x0000, devices that join the network can assume this and communicate with the ESP directly. Therefore, no device discovery or service discovery is required for joining devices. The ESP however, demonstrates the ability to discover Load Control devices and PCTs.

Once the device joins the network and performs a successful key establishment, it starts its communications with the ESP based on the application behavior.

The following section defines the application behavior for each device after it has successfully joined the network using SE secure join, and has an application link key established with the ESP.

4.4. ESP

The ESP is assumed to be the coordinator, trust center, and network manager of the network. Every other device communicates with the ESP.

The ESP application instance consists of the following modules:

OSAL_esp.c - functions and tables for task initialization
esp.c – main application function that has init and event loop function
esp.h – header file for application module
esp_data.c – container for declaration of attributes, clusters, simple descriptor

The ESP application makes function calls to the following ZCL SE functions and ZDO API functions:

esp_GetCurrentPriceCB – callback executed when get current price message is received

zclSE_Pricing_Send_PublishPrice – send publish price command to IPD

esp_ProcessInReportCmd - print out value of current summation delivered when attribute report is received

zclSE_LoadControl_Send_LoadControlEvent – send a load control event message to a load control device

zclSE_Message_Send_DisplayMessage – send a message command to an IPD

esp_ProcessAppMsg – MT_SYS_APP_MSG is used to extract the link key from the ESP using Z-Tool. This function in turn calls esp_KeyEstablish_ReturnLinkKey.

esp_ProcessZDOMsg – responses for match descriptor and simple descriptor requests are processed here. Only two match descriptor responses are handled by default, but this implementation could be extended to handle more than two responses. For the first response, the SIMPLE_DESC_QUERY_EVT_1 event is set, and this causes the simple descriptor request to be triggered for this source address. If another response is received, the SIMPLE_DESC_QUERY_EVT_2 event is set, and this causes the simple descriptor request to be triggered for the second source address.

esp_HandleKeys – user switch events are processed here. SW1 sends out a load control event to the PCT, SW2 sends out a load control event to the load control device, SW3 sends out a display message command to the IPD, and SW4 issues a match descriptor request to discover the PCT and load control device.

The ESP interacts with other application instances in different ways. At first, the ESP is configured with Certicom key information using MT OSAL_NV_READ and OSAL_NV_WRITE commands. A Z-Tool script is used to configure each device. Endpoints, command structures, ZDO callbacks, and SE callbacks are initialized by calling esp_Init. System and user events are handled in esp_event_loop.

When a Simple Metering device joins the network, once the key establishment procedure is successful (the ESP application doesn't have to do anything here since key establishment is handled via the zcl_key_establishment.c module), it will receive attribute reports from the Simple Metering device every 5 seconds. Therefore, every 5 seconds the function esp_ProcessInReportCmd is called to display the received CurrentSummationDelivered attribute on the ESP's LCD screen.

When an IPD device joins the network, once the key establishment procedure is successful (the ESP application doesn't have to do anything here since key establishment is handled via the zcl_key_establishment.c module), it will send a get pricing info message. On the ESP, the esp_GetCurrentPriceCB is called when the get pricing info message is received. This then calls zclSE_Pricing_Send_PublishPrice() to send out the publish price command to the IPD. If SW4 is pressed, the function zclSE_Message_Send_DisplayMessage() is called to send a display MESSAGE to the IPD device.

When a PCT or Load Control device joins the network and establishes a link key with the ESP, the ESP will discover these devices by using the match descriptor and simple descriptor ZDO requests. SW4 sends a broadcast match descriptor request that solicits the response from the PCT and Load Control device. Each response is collected and triggers a user event called SIMPLE_DESC_QUERY_EVT_1 and SIMPLE_DESC_QUERY_EVT_2. Only two responses are collected and processed, and therefore only two load control type devices (PCT or Load Control Device) are supported at any one time. It is easy to extend the functionality to support more than two though. For the first match descriptor response, the user event SIMPLE_DESC_QUERY_EVT_1 is initiated, and for the second match descriptor response, the user event SIMPLE_DESC_QUERY_EVT_2 is initiated. Each simple descriptor response is parsed in the ZDO callback handler esp_ProcessZDOMsg, and the device ID field is checked to determine whether this response came from the PCT or Load Control Device. Depending on the device ID, the destination address for the PCT or Load Control Device is then populated. This is how the ESP then knows which type of load control event to send to which device. SW1 is used to send a PCT event to the PCT, and SW2 is used to send an event to the load control device.

4.5. Simple Metering Device

The Simple Metering Device will periodically send reports for the CurrentSummationDelivered Simple Metering attribute to the ESP. The ESP will display the CurrentSummationDelivered value on the LCD. See Figure 4 for a sequence diagram.

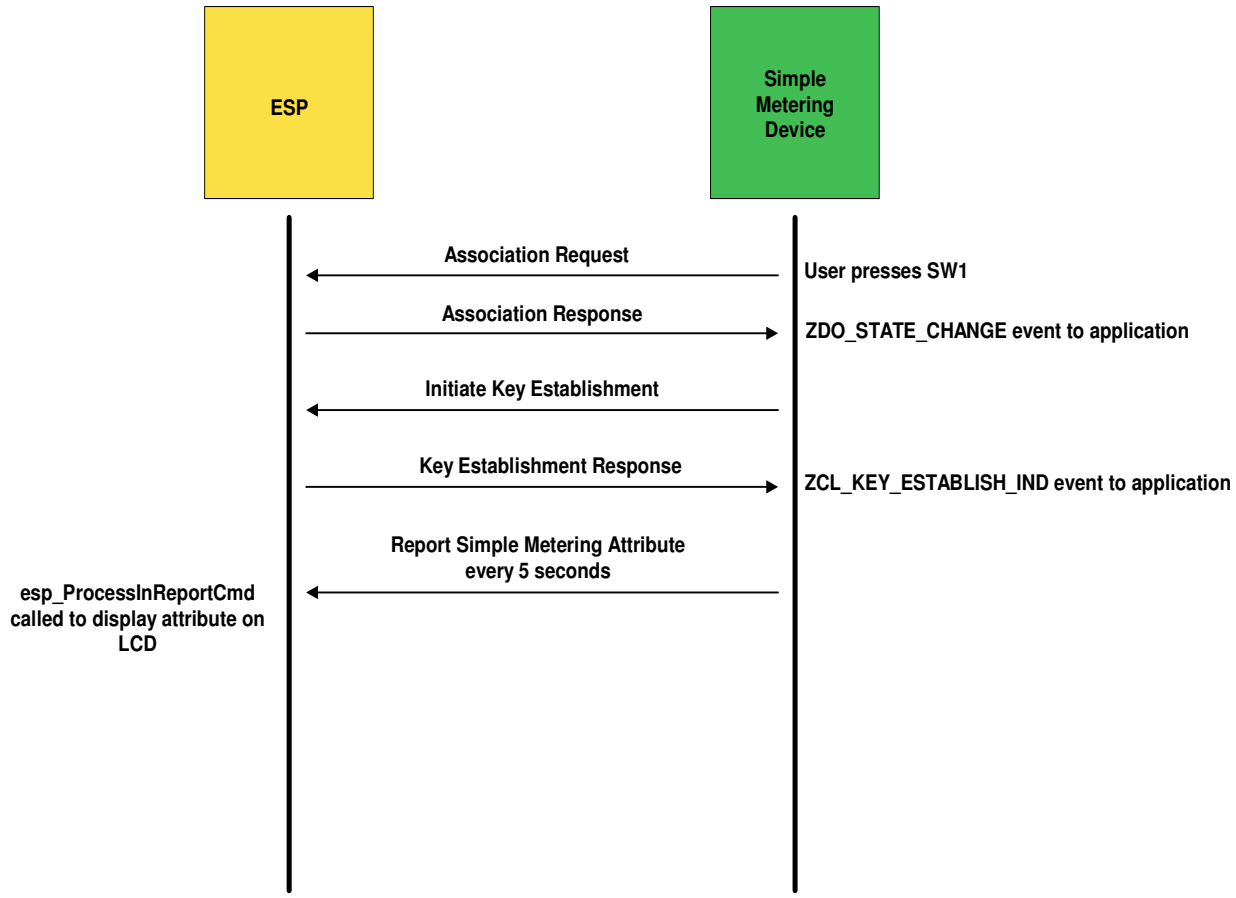


Figure 4. Sequence Diagram for a Simple Meter Device

The Simple Meter application instance consists of the following modules:

OSAL_simplemeter.c - functions and tables for task initialization
simplemeter.c – main application function that has init and event loop function
simplemeter.h – header file for application module
simplemeter_data.c – container for declaration of attributes, clusters, simple descriptor

The Simple Meter application makes the following function calls:

zcl_SendReportCmd – this function sends the CurrentSummationDelivered report attribute to the ESP.

simplemeter_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the simple meter device joins the network, it goes through the state machine explained in Figure 2. simplemeter_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If it hasn't, it will set the osal event SIMPLEMETER_KEY_ESTABLISHMENT_REQUEST_EVT. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. An osal timer event called SIMPLEMETER_REPORT_ATTRIBUTE_EVT is then started to send attribute reports every 5 seconds of the CurrentSummationDelivered attribute. The structure for the report command is created and initialization of the attribute is done in the simplemeter_Init() function.

4.6. Load Control Device

The ESP will send a load control event to the Load Control Device via a user switch button press. In the Load Control Event payload, the Device Class Field Bitmap will indicate that Bit 7 is set (representing Residential ON/OFF Load). When the Load Control Device receives the load control event, it sends a ReportEventStatus command that it received it, and sends another one when it starts it. The Start Time field will indicate to "Start Now". The event duration will last 1 minute. While the load control event is in process, the Load Control Device will flash its LED. When the load control event is finished, another ReportEventStatus command will be sent to the ESP to indicate the completion of the load control event. See Figure 5 for a sequence diagram.

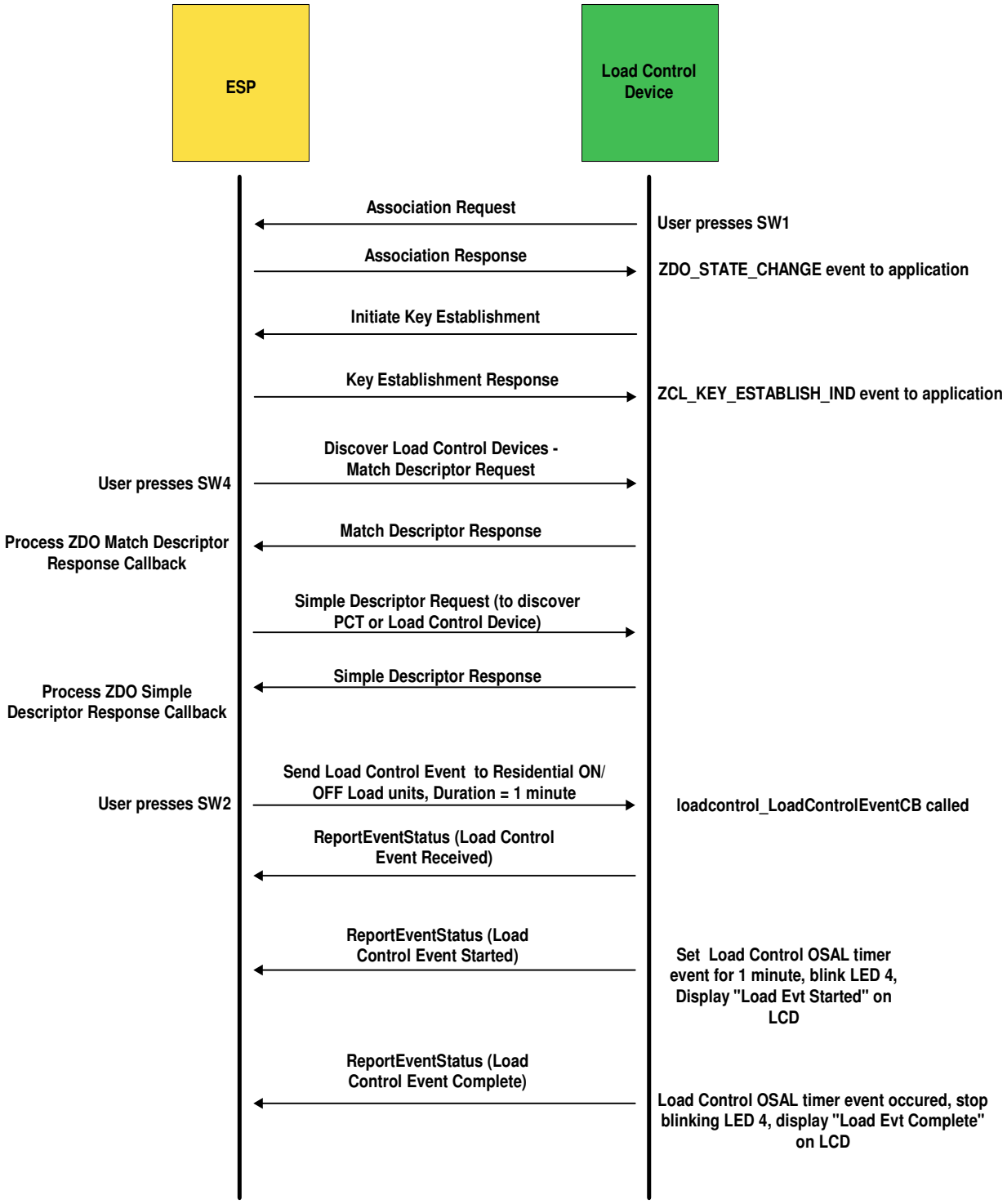


Figure 5. Sequence Diagram for a Load Control Device

OSAL_loadcontrol.c - functions and tables for task initialization
loadcontrol.c – main application function that has init and event loop function
loadcontrol.h – header file for application module

loadcontrol_data.c – container for declaration of attributes, clusters, simple descriptor

The Load Control application makes function calls to the following ZCL SE functions and ZDO API functions:

loadcontrol_LoadControlEventCB – this function is called when the load control device received a load control command.

zclSE_LoadControl_Send_ReportEventStatus – When a load control command is received, the device sends a report event status command back to the ESP. Possible values for the event type for this particular sample app could be event received, event started, and event completed. The response structure (rsp) is made a global variable so that the process event loop can manipulate the eventStatus field once the load control event is complete.

loadcontrol_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the Load Control device joins the network, it goes through the state machine explained in Figure 2. loadcontrol_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If it hasn't, it will set the osal event LOADCONTROL_KEY_ESTABLISHMENT_REQUEST_EVT. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. Nothing is done at this point, and the load control device is ready to accept load control messages from the ESP. When a load control command is received, the callback loadcontrol_LoadControlEventCB is executed. This callback function checks the issuer event id (0x12345678), the start time (0x00000000 = NOW) in order to make sure that it is not just blindly responding to a random load control event. Furthermore, it checks the deviceGroupClass to determine whether this load control event was for the PCT or load control device. The values within this load control command originate from the ESP. The correct string on the LCD is then displayed to indicate whether this is a load control device event or PCT event. It is assumed that the load control device belongs to the residential on/off load device class, and that the PCT belongs to the HVAC compressor/furnace device class.
An osal timer event called LOADCONTROL_LOAD_CTRL_EVT is then started to commence the load control event and flash the LED for the duration specified, which is 1 minute. When this timer event expires, the status response of event completed is sent back to the ESP, and the LED stops flashing. The user also sees a display on the LCD indicating that the load control event is complete.

4.7. PCT

The PCT will have a very similar behavior to the Load Control Device. However, in the Load Control Event payload, the Device Class Field BitMap will indicate that Bit 0 is set (HVAC compressor or furnace). The Load Control Event for the PCT will be regarded as a “PCT Event”, and the message “PCT Event Started” will be displayed if the device has LCD support. See Figure 6 for a sequence diagram.

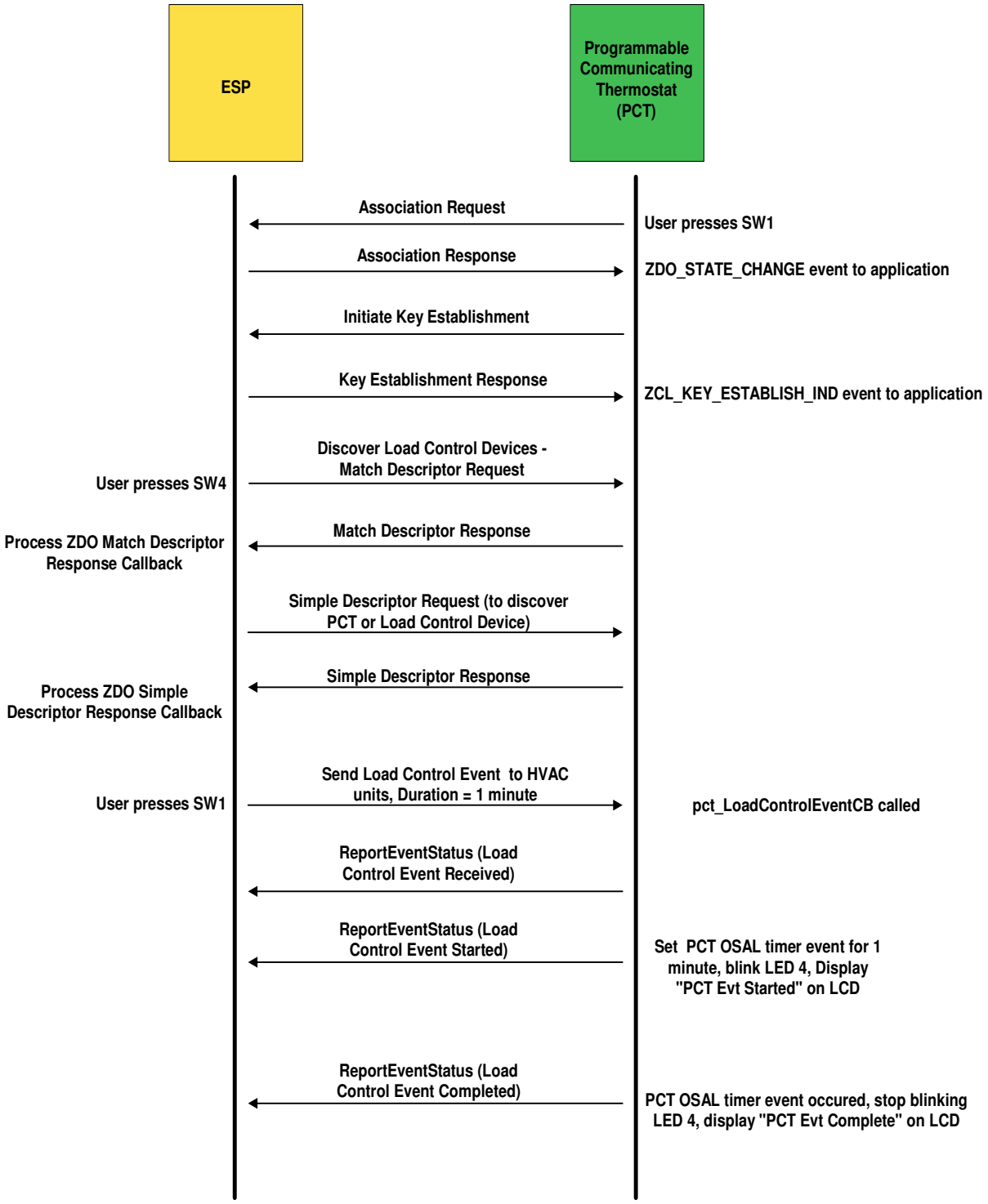


Figure 6. Sequence Diagram for a PCT

OSAL_pct.c - functions and tables for task initialization
pct.c – main application function that has init and event loop function
pct.h – header file for application module
pct_data.c – container for declaration of attributes, clusters, simple descriptor

The PCT application makes function calls to the following ZCL SE functions and ZDO API functions:

pct_LoadControlEventCB – this function is called when the PCT device received a load control command.

zclSE_LoadControl_Send_ReportEventStatus – When a load control command is received, the device sends a report event status command back to the ESP. Possible values for the event type for this particular sample app could be event received, event started, and event completed. The response structure (rsp) is made a global variable so that the process event loop can manipulate the eventStatus field once the PCT event is complete.

pct_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the PCT device joins the network, it goes through the state machine explained in Figure 2. pct_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If it hasn't, it will set the osal event PCT_KEY_ESTABLISHMENT_REQUEST_EVT. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. Nothing is done at this point, and the PCT device is ready to accept load control messages from the ESP. When a load control command is received, the callback pct_LoadControlEventCB is executed. This callback function checks the issuer event id (0x12345678), the start time (0x00000000 = NOW) in order to make sure that it is not just blindly responding to a random load control event. Furthermore, it checks the deviceGroupClass to determine whether this load control event was for the PCT or load control device. The values within this load control command originate from the ESP. The correct string on the LCD is then displayed to indicate whether this is a load control device event or PCT event. It is assumed that the load control device belongs to the residential on/off load device class, and that the PCT belongs to the HVAC compressor/furnace device class.
An osal timer event called PCT_LOAD_CTRL_EVT is then started to commence the load control event and flash the LED for the duration specified, which is 1 minute. When this timer event expires, the status response of event completed is sent back to the ESP, and the LED stops flashing. The user also sees a display on the LCD indicating that the PCT event is complete.

Note: The mechanics of the PCT and Load Control Device are the same. There are just slight differences in variable names and the type of message displayed on the LCD.

4.8. In Premise Display

The In Premise Display will use the GetCurrentPrice command to obtain the current pricing info from the ESP. The pricing information will be displayed on its LCD. The ESP will be able to use the SW3 button press to send a display MESSAGE command to the In Premise Display. This message will then be displayed on the LCD. See Figure 7 for a sequence diagram.

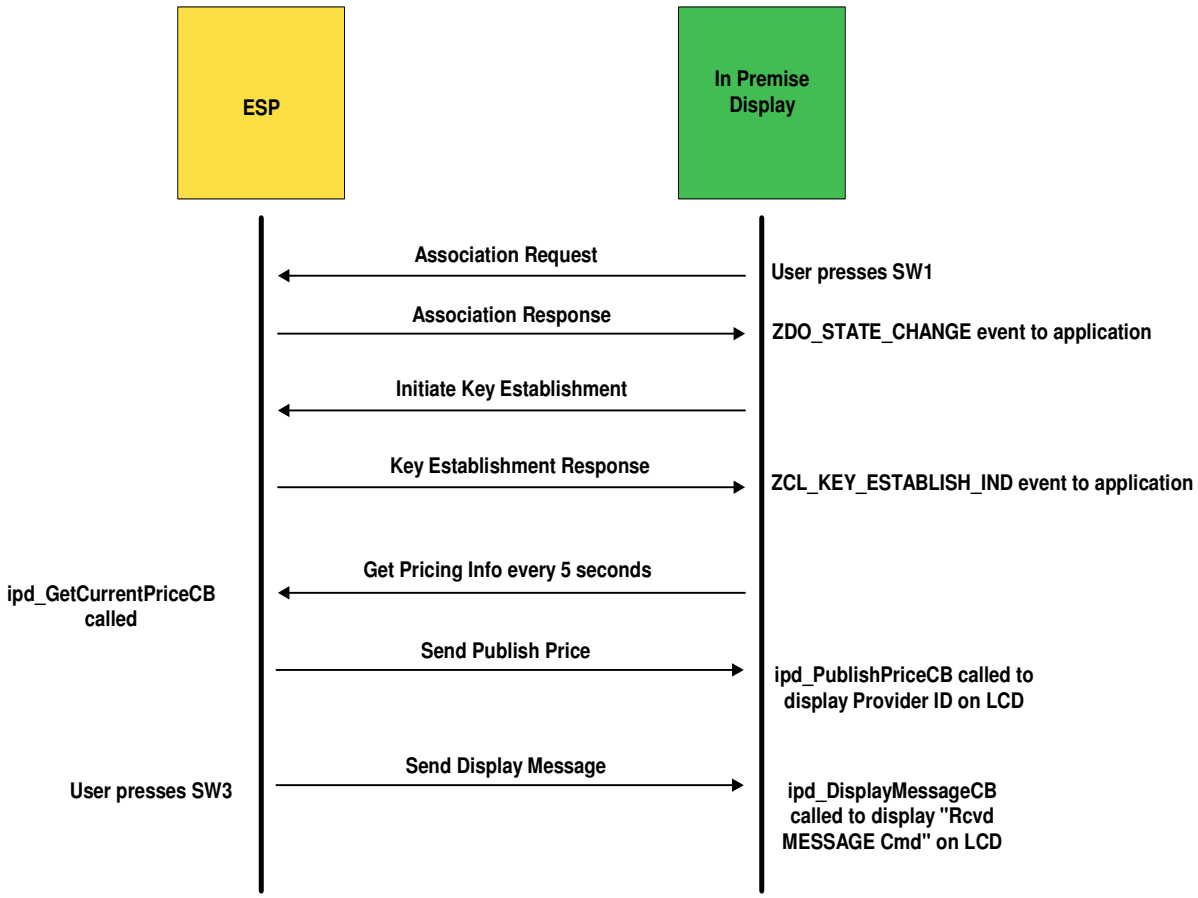


Figure 7. Sequence Diagram for an In Premise Display

OSAL_ipd.c - functions and tables for task initialization
ipd.c – main application function that has init and event loop function
ipd.h – header file for application module
ipd_data.c – container for declaration of attributes, clusters, simple descriptor

The IPD application makes function calls to the following ZCL SE functions and ZDO API functions:

zclSE_Pricing_Send_GetCurrentPrice – this function sends the get current price request to the ESP.

ipd_PublishPriceCB – this function is called when the IPD device receives the publish price command from the ESP, and displays the provider ID on the LCD.

ipd_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the IPD device joins the network, it goes through the state machine explained in Figure 2. pct_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If it hasn't, it will set the osal event IPD_KEY_ESTABLISHMENT_REQUEST_EVT. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. An osal timer event called IPD_GET_PRICING_INFO_EVT is then started to send the get pricing info request every 5 seconds. The result of the pricing request is received via ipd_PublishPriceCB and is displayed on the LCD. Only the provider ID field of the published price payload is displayed.

4.9. Range Extender

The Range Extender device will not exchange application data with the ESP. It will be able to join the SE network, perform key establishment, and route packets. Figure 8 shows a sequence diagram for the Range Extender.

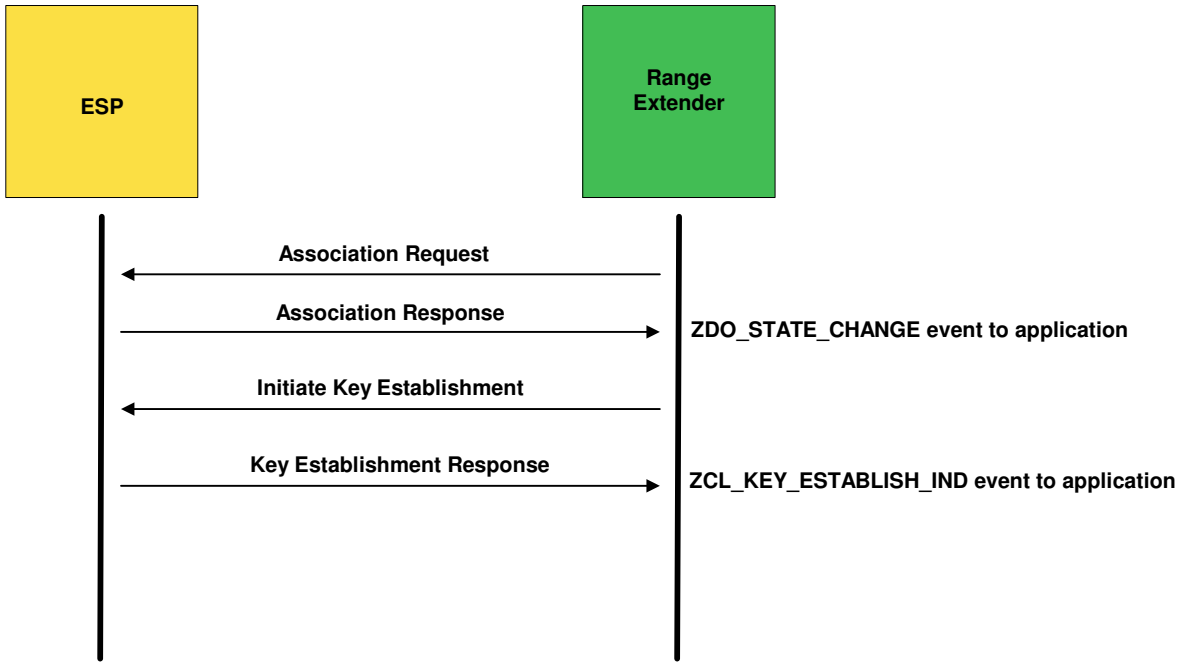


Figure 8. Sequence Diagram for a Range Extender

OSAL_rangeext.c - functions and tables for task initialization
rangeext.c – main application function that has init and event loop function
rangeext.h – header file for application module
rangeext_data.c – container for declaration of attributes, clusters, simple descriptor

The Range Extender application makes function calls to the following functions:

rangeext_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

The Range Extender application does not send any application level SE messages, but it does join the network and perform key establishment. It functions as a standard Zigbee Pro router device.

5. Limitations

5.1. Trust Center Operation

- The trust center does not demonstrate how to refresh network keys.
- The trust center does not demonstrate how to manage an access control list
- The trust center assumes that joining devices have the same pre-configured trust center link key already established at compile time or using Z-Tool. It is assumed that the trust center link key is already available. Out of band means of deriving the trust center link key is out of the scope of this sample application.

5.2. Network Manager Operation

- The ESP is the only device that has the Network Manager functionality.
- The ESP application instance should be compiled with the NWK_MANAGER compile option in order to enable Network Manager functionality hooks.
- The default network manager application ZDNwkMgr.c is provided “as is” and there is no application level functionality incorporated to demonstrate this Network Manager application.

5.3. Secure Joining Operation

All devices must have the pre-configured Trust Center Link Key defined at compile time, or entered via Z-Tool.

5.4. Key Establishment Operation

SE Key Establishment is always initiated by the joining device and its partner will always be the ESP.

5.5. Device Startup Behavior

The Smart Energy Profile spec mentions application best practices of controlling join/rejoin duty cycles. This sample app does not make any attempt at changing the join/rejoin mechanism as implemented in ZDApp.c. The Smart Energy Profile spec says that end devices should not be more than 7.5 seconds. This sample application therefore supports a nominal poll rate of 8 seconds. The NWK_INDIRECT_MSG_TIMEOUT parameter in the f8wConfig.cfg file must be increased to a suggested value of 10 in order to buffer the message long enough to accommodate the 8 second poll period by the end device. The MAX_POLL_FAILURE_RETRIES parameter should also be set to 4 as during the CBKE procedure the ESP will be somewhat unresponsive to data requests during this time.

5.6. Load Control Device Behavior

There is no example of how to cancel a load control event from the ESP or how to supersede an ongoing load control event.

5.7. ESP Behavior

The ESP service discovery implementation is limited to supporting up to two load control devices at any given time, but could be easily modified to support more.

6. Applicable Documents

6.1. Z-Stack Documents (part of the Z-Stack installer)

1. OSAL API, Texas Instruments Document SWRA194
2. Z-Stack API, Texas Instruments Document SWRA195
3. Z-Stack ZCL API, Texas Instruments Document SWRA197
4. Z-Stack Monitor and Test API, Texas Instruments Document SWRA198
5. Z-Stack Smart Energy Developer's Guide, Texas Instruments Document SWRA216

6.2. Other Documents (www.zigbee.org)

1. ZigBee Alliance – Smart Energy Profile Specification
2. ZigBee Alliance –ZigBee Smart Energy Test Specification