# Z-Stack
# Smart Energy
# Developer's Guide

Document Number: SWRA216

**Texas Instruments, Inc.**
San Diego, California USA

| Version | Description | Date |
|---|---|---|
| 1.0 | Initial release. | 6/06/2008 |
| 1.1 | Add Key Establishment Task registration section, application link key NV management section, and updates all commands details. | 6/16/2008 |
| 1.2 | Add SE secure join section | 6/20/2008 |
| 1.3 | Update for Z-Stack 2.2.0 | 4/02/2009 |
| 1.4 | Update `zclSE_AppCallbacks_t` table with new callbacks<br>Add SE 1.1 features:<br>- Fast Poll command and response<br>- Price Acknowledgment command<br>- New fields to Publish Price structure<br>- Get Block Period and Publish Block Period commands<br>- Select Available Emergency Credit Command<br>- Change Supply Command<br>- Supply Status Response Command<br>Add Communicating With Non Trust Center Devices section<br>Add Multiple ESI section | 07/17/2011 |

## TABLE OF CONTENTS



 

## LIST OF FIGURES



 

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to define the Smart Energy (SE) APIs. These APIs allows the higher layers (Profile and Application) to access the SE functionality. The following ZigBee Cluster Libraries (ZCL) are added in the SE functional domain:

- Demand Response and Load Control
- Simple Metering
- Price
- Messaging

The following new cluster is added to the existing General functional domain:

- Key Establishment

This document covers only the APIs for the above-listed SE and General clusters. Please refer to [2] for the ZCL Foundation and existing General functional domain APIs.

## 1.2 Scope

This document enumerates all the function calls provided by the SE clusters defined in the SE and General functional domains. It also enumerates the callback functions that need to be provided by the higher layers. Furthermore, this document doesn't explain the SE concepts which are explained in detail in reference [1].

## 1.3 Acronyms

| | |
|---|---|
| API | Application Programming Interface |
| APS | Application Support Sub-Layer |
| CA | Certificate Authority |
| CBKE | Certificate-Based Key Establishment |
| ECC | Elliptic Curve Cryptography |
| MT | Monitor Test |
| NV | Non-Volatile |
| NWK | Network Layer |
| OSAL | Operating System Abstraction Layer |
| PAN | Personal Area Network |
| SE | Smart Energy |
| ZCL | ZigBee Cluster Library |
| ZDO | ZigBee Device Object |

 

### 1.4 Applicable Documents

1. ZigBee Alliance – Smart Energy Profile Specification. (Latest revision can be found on ZigBee Alliance website http://www.zigbee.org ). See the Smart Energy Functional Specification for the revision numbers.
2. Texas Instruments – Z-Stack ZCL API (SWRA197).
3. Texas Instruments – Z-Stack Monitor and Test API (SWRA198).

## 2. API Overview

### 2.1 Overview

The SE and General functional domains provide APIs to the higher layers to:

1. Generate Request and Response commands
2. Register Application's Command callback functions

### 2.2 Application/Profile Registration

The ZCL Foundation provides APIs to the Application/Profile to register their Attribute List, Cluster Option List, Attribute Data Validation and Cluster Library Handler callback functions. The General functional domain provides an API to register the Application's Command callback functions. Please refer to [2] Section 2.2 for detailed description of these APIs.

The SE functional domain provides the `zclSE_RegisterCmdCallbacks()` API to register the Application's Command callback functions. The command callback input parameter to this API is of the following type:

```
// Register Callbacks table entry - enter function pointers for callbacks
// that the application would like to receive.

typedef struct
{
  zclSE_SimpleMeter_GetProfileCmd_t                pfnSimpleMeter_GetProfileCmd;
  zclSE_SimpleMeter_GetProfileRsp_t                pfnSimpleMeter_GetProfileRsp;
  zclSE_SimpleMeter_ReqMirrorCmd_t                 pfnSimpleMeter_ReqMirrorCmd;
  zclSE_SimpleMeter_ReqMirrorRsp_t                 pfnSimpleMeter_ReqMirrorRsp;
  zclSE_SimpleMeter_MirrorRemCmd_t                 pfnSimpleMeter_MirrorRemCmd;
  zclSE_SimpleMeter_MirrorRemRsp_t                 pfnSimpleMeter_MirrorRemRsp;
  zclSE_Pricing_GetCurentPrice_t                   pfnPricing_GetCurrentPrice;
  zclSE_Pricing_GetScheduledPrice_t                pfnPricing_GetScheduledPrice;
  zclSE_Pricing_PublishPrice_t                     pfnPricing_PublishPrice;
  zclSE_Message_DisplayMessage_t                   pfnMessage_DisplayMessage;
  zclSE_Message_CancelMessage_t                    pfnMessage_CancelMessage;
  zclSE_Message_GetLastMessage_t                   pfnMessage_GetLastMessage;
  zclSE_Message_MessageConfirmation_t              pfnMessage_MessageConfirmation;
  zclSE_LoadControl_LoadControlEvent_t             pfnLoadControl_LoadControlEvent;
  zclSE_LoadControl_CancelLoadControlEvent_t       pfnLoadControl_CancelLoadControlEvent;
  zclSE_LoadControl_CancelAllLoadControlEvents_t   pfnLoadControl_CancelAllLoadControlEvents;
  zclSE_LoadControl_ReportEventStatus_t            pfnLoadControl_ReportEventStatus;
  zclSE_LoadControl_GetScheduledEvent_t            pfnLoadControl_GetScheduledEvents;
  zclSE_SimpleMeter_ReqFastPollModeCmd_t           pfnSimpleMeter_ReqFastPollModeCmd;
  zclSE_SimpleMeter_ReqFastPollModeRsp_t           pfnSimpleMeter_ReqFastPollModeRsp;
  zclSE_Pricing_PriceAcknowledgement_t             pfnPricing_PriceAcknowledgement;
  zclSE_Pricing_GetBlockPeriod_t                   pfnPricing_GetBlockPeriod;
  zclSE_Pricing_PublishBlockPeriod_t               pfnPricing_PublishBlockPeriod;
  zclSE_Prepayment_SelAvailEmergencyCredit_t       pfnPrepayment_SelAvailEmergencyCredit;
  zclSE_Prepayment_ChangeSupply_t                  pfnPrepayment_ChangeSupply;
  zclSE_Prepayment_SupplyStatusResponse_t          pfnPrepayment_SupplyStatusResponse;
} zclSE_AppCallbacks_t;
```

The prototype of each command callback function is defined in Section 3. If the application does not support some of the callbacks, the table entry shall be input as NULL.

### 2.3 Application Creation

Section 2.5 in [2] outlines the steps to be taken when creating a new ZCL application. Instead of the last step explained in [2] Section 2.5.3, the application's initialization function `zcl<AppName>_Init()` should register its *simple descriptor* with the SE profile by calling `zclSE_Init()`, defined in the *se.c* module. The application also needs to call `zclSE_RegisterCmdCallbacks()`, defined in the *zcl_se.c* module, to register the application's command callback functions.

To support the SE profile, the user also needs to add a number ZCL source files which can be found here:

- $PROJ_DIR$\..\..\..\..\..\Components\stack\zcl
- $PROJ_DIR$\..\Source

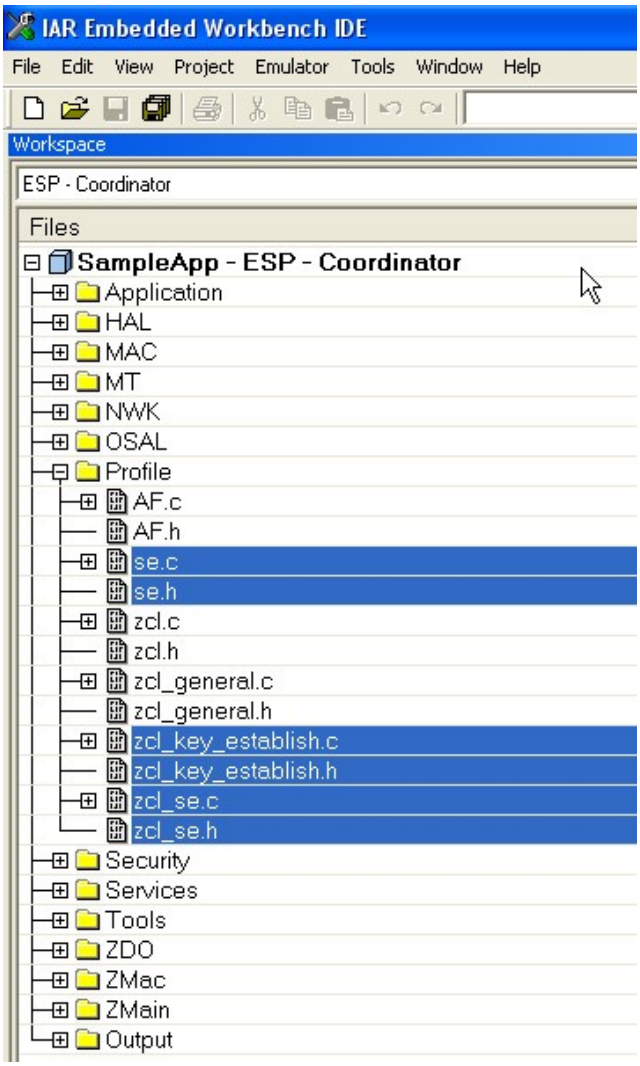


**Figure 1: Screen shot on adding new ZCL source files to support the SE profile**

# 3. SE Functional Domain

## 3.1 Introduction

The SE functional domain contains the following clusters:

- Demand Response and Load Control
- Simple Metering
- Price
- Messaging
- Prepayment

Each SE cluster consists of a group of attributes and commands. Detailed attribute list and command frame format are listed in [1] Annex D. These clusters, namely Price, Demand Response and Load Control, Simple Metering, Message and Prepayment, are all implemented in *zcl_se.c* and *zcl_se.h* files.

## 3.2 Send Get Profile Command (Simple Metering)

### 3.2.1 Description

This function is used to send out a Get Profile Command.

### 3.2.2 Prototype

```
ZStatus_t zclSE_SimpleMetering_Send_GetProfileCmd( uint8 srcEP,
                                                   afAddrType_t *dstAddr,
                                                   uint8 channel,
                                                   uint32 endTime,
                                                   uint8 numOfPeriods,
                                                   uint8 disableDefaultRsp,
                                                   uint8 seqNum );
```

### 3.2.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`channel` - Interval Channel: Enumerated value used to select the quantity of interest returned by the Get Profile Response Command. The Interval Channel value should be set to 0 for consumption delivered and 1 for consumption received..

`endTime` - UTC time for the starting time of requested interval.

`numOfPeriods` - Number of periods requested.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

### 3.2.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

 

### 3.3 Send Get Profile Response (Simple Metering)

#### 3.3.1 Description

This function is used to send out a Get Profile Response. It is normally sent out in response to a Get Profile Command.

#### 3.3.2 Prototype

```
ZStatus_t zclSE_SimpleMetering_Send_GetProfileRsp( uint8 srcEP,
                                                   afAddrType_t *dstAddr,
                                                   uint32 endTime,
                                                   uint8 rspStatus,
                                                   uint8 profileIntervalPeriod,
                                                   uint8 numOfPeriodDelivered,
                                                   uint24 *intervals,
                                                   uint8 disableDefaultRsp,
                                                   uint8 seqNum );
```

#### 3.3.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`endTime` - UTC time for the starting time of requested interval.

`rspStatus` – status.

`profileIntervalPeriod` - number of periods requested.

`numOfPeriodDelivered` - Number of entries in the intervals array.

`intervals` - Array of interval data captured using the period specified by profileIntervalPeriod. Data is organized in a reverse chronological order, the most recent interval is transmitted first and the oldest interval is transmitted last. Invalid intervals should be marked as 0xFFFFFF.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.3.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.4 Send Request Mirror Command (Simple Metering)

#### 3.4.1 Description

This function is used to send out Request Mirror Command.

#### 3.4.2 Prototype

```
ZStatus_t zclSE_SimpleMetering_Send_ReqMirrorCmd ( uint8 srcEP,
                                                   afAddrType_t *dstAddr,
                                                   uint8 disableDefaultRsp,
                                                   uint8 seqNum )
```

 

#### 3.4.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.4.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.5 Send Request Mirror Response (Simple Metering)

#### 3.5.1 Description

This function is used to send out Request Mirror Response.

#### 3.5.2 Prototype

```
ZStatus_t zclSE_SimpleMetering_Send_ReqMirrorRsp( uint8 srcEP,
                                                  afAddrType_t *dstAddr,
                                                  uint16 endpointId,
                                                  uint8 disableDefaultRsp,
                                                  uint8 seqNum )
```

#### 3.5.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`endpointId` – endpoint ID to contain the Metering Devices meter data.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.5.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.6 Send Remove Mirror Command (Simple Metering)

#### 3.6.1 Description

This function is used to send out Remove Mirror Command.

#### 3.6.2 Prototype

```
ZStatus_t zclSE_SimpleMetering_Send_RemMirrorCmd ( uint8 srcEP,
                                                   afAddrType_t *dstAddr,
```

 

```
                                                    uint8 disableDefaultRsp,
                                                    uint8 seqNum )
```

#### 3.6.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.6.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.7 Send Remove Mirror Response (Simple Metering)

#### 3.7.1 Description

This function is used to send out Remove Mirror Response.

#### 3.7.2 Prototype

```
ZStatus_t zclSE_SimpleMetering_Send_RemMirrorRsp( uint8 srcEP,
                                                  afAddrType_t *dstAddr,
                                                  uint16 endpointId,
                                                  uint8 disableDefaultRsp,
                                                  uint8 seqNum )
```

#### 3.7.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`endpointId` – endpoint ID to contain the Metering Devices meter data.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.7.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.8 Send Request Fast Poll Mode Command (Simple Metering)

#### 3.8.1 Description

This function is used to send out Request Fast Poll Mode Command.

 

### 3.8.2 Prototype

```
ZStatus_t zclSE_SimpleMetering_Send_ReqFastPollModeCmd( uint8 srcEP,
                                                        afAddrType_t *dstAddr,
                                                        zclCCReqFastPollModeCmd_t *cmd,
                                                        uint8 disableDefaultRsp,
                                                        uint8 seqNum)
```

### 3.8.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Request Fast Poll Mode command. The structure of the command is as follows.

```
typedef struct
{
    uint8 fastPollUpdatePeriod;
    uint8 duration;
} zclCCReqFastPollModeCmd_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

### 3.8.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

## 3.9 Send Request Fast Poll Mode Response (Simple Metering)

### 3.9.1 Description

This function is used to send out Request Fast Poll Mode Response. It is normally sent out in response to a Request Fast Poll Mode Command.

### 3.9.2 Prototype

```
ZStatus_t zclSE_SimpleMetering_Send_ReqFastPollModeRsp( uint8 srcEP,
                                                        afAddrType_t *dstAddr,
                                                        zclCCReqFastPollModeRsp_t *cmd,
                                                        uint8 disableDefaultRsp,
                                                        uint8 seqNum )
```

### 3.9.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` – Where you want the message to go

`cmd` – Request Fast Poll Mode response. The structure of the command is as follows.

```
typedef struct
{
```

 

```
        uint8 appliedUpdatePeriod;
        uint8 fastPollModeEndTime;
    } zclCCReqFastPollModeRsp_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.9.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.10 Send Get Scheduled Price Command (Price)

#### 3.10.1 Description

This function is used to send out a Get Scheduled Price Command.

#### 3.10.2 Prototype

```
ZStatus_t zclSE_Pricing_Send_GetScheduledPrice(uint8 srcEP,
                                               afAddrType_t *dstAddr,
                                               zclCCGetScheduledPrice_t *cmd,
                                               uint8 disableDefaultRsp,
                                               uint8 seqNum );
```

#### 3.10.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Get Scheduled price command. The structure of the command is as follows.

```
    typedef struct
    {
        uint32 startTime;
        uint8  numEvents;
    } zclCCGetScheduledPrice_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.10.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.11 Send Get Current Price Command (Price)

#### 3.11.1 Description

 

This function is used to send out a Get Current Price Command.

### 3.11.2 Prototype

```
ZStatus_t zclSE_Pricing_Send_GetCurrentPrice ( uint8 srcEP,
                                               afAddrType_t *dstAddr,
                                               uint8 option,
                                               uint8 disableDefaultRsp,
                                               uint8 seqNum );
```

### 3.11.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`option` – Command option field. The command options field is 8 Bits in length and is formatted as a bit field. Bit 0 is the Requestor Rx On When Idle sub-field.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

### 3.11.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

## 3.12 Send Publish Price Command (Price)

### 3.12.1 Description

This function is used to send out a Publish Price Command.

### 3.12.2 Prototype

```
ZStatus_t zclSE_Pricing_Send_PublishPrice( uint8 srcEP,
                                           afAddrType_t *dstAddr,
                                           zclCCPublishPrice_t *cmd,
                                           uint8 disableDefaultRsp,
                                           uint8 seqNum );
```

### 3.12.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Publish price command. See section 8.1.1 for description of changes to the structure. The structure of the command is as follows. Please refer to Reference 2 Section Publish Price Command for details of each field.

```
typedef struct
{
  uint32  providerId;
  UTF8String_t rateLabel;
```

 

```
        uint32  issuerEventId;
        uint32  currentTime;
        uint8   unitOfMeasure;
        uint16  currency;
        uint8   priceTrailingDigit;
        uint8   numberOfPriceTiers;
        uint32  startTime;
        uint16  durationInMinutes;
        uint32  price;
        uint8   priceRatio;
        uint32  generationPrice;
        uint8   generationPriceRatio;
        uint32  alternateCostDelivered;
        uint8   alternateCostUnit;
        uint8   alternateCostTrailingDigit;
        uint8   numberOfBlockThresholds;
        uint8   priceControl;
      } zclCCPublishPrice_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.12.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.13 Send Price Acknowledgment Command (Price)

#### 3.13.1 Description

This function is used to send out a Price Acknowledgment Command.

#### 3.13.2 Prototype

```
ZStatus_t zclSE_Pricing_Send_PriceAcknowledgement( uint8 srcEP,
                                                   afAddrType_t *dstAddr,
                                                   zclCCPriceAcknowledgement_t *cmd,
                                                   uint8 disableDefaultRsp,
                                                   uint8 seqNum );
```

#### 3.13.3 Parameter Details

`srcEP` – Sending application's endpoint.

 

dstAddr - Where you want the message to go.

cmd – Price Acknowledgment command. The structure of the command is as follows.

```
typedef struct
{
  uint32 providerId;
  uint32 issuerEventId;
  uint32 priceAckTime;
  uint8  control;
} zclCCPriceAcknowledgement_t;
```

disableDefaultRsp - Disable Default Response command.

seqNum - ZCL sequence number.

#### 3.13.4 Return

ZStatus_t – status definitions found in ZComDef.h.

### 3.14 Send Get Block Period Command (Price)

#### 3.14.1 Description

This function is used to send out a Get Block Period Command.

#### 3.14.2 Prototype

```
ZStatus_t zclSE_Pricing_Send_GetBlockPeriod( uint8 srcEP,
                                             afAddrType_t *dstAddr,
                                             zclCCGetBlockPeriod_t *cmd,
                                             uint8 disableDefaultRsp,
                                             uint8 seqNum );
```

#### 3.14.3 Parameter Details

srcEP – Sending application's endpoint.

dstAddr - Where you want the message to go.

cmd – Get Block Period command. The structure of the command is as follows.

```
typedef struct
{
  uint32 startTime;
  uint8  numEvents;
} zclCCGetBlockPeriod_t;
```

disableDefaultRsp - Disable Default Response command.

seqNum - ZCL sequence number.

 

#### 3.14.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.15 Send Publish Block Period Command (Price)

#### 3.15.1 Description

This function is used to send out a Publish Block Period Command.

#### 3.15.2 Prototype

```
ZStatus_t zclSE_Pricing_Send_PublishBlockPeriod( uint8 srcEP,
                                                 afAddrType_t *dstAddr,
                                                 zclCCPublishBlockPeriod_t *cmd,
                                                 uint8 disableDefaultRsp,
                                                 uint8 seqNum );
```

#### 3.15.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Publish Block Period command. The structure of the command is as follows.

```
typedef struct
{
  uint32 providerId;
  uint32 issuerEventId;
  uint32 blockPeriodStartTime;
  uint24 blockPeriodDurInMins;
  uint8  numPriceTiersAndBlock;
  uint8  blockPeriodControl;
} zclCCPublishBlockPeriod_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.15.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.16 Send Display Message Command (Message)

 

### 3.16.1 Description

This function is used to send out a Display Message Command. It is normally sent out by the ESP or in response to Get Last Message Command.

### 3.16.2 Prototype

```
ZStatus_t zclSE_Message_Send_DisplayMessage( uint8 srcEP,
                                             afAddrType_t *dstAddr,
                                             zclCCDisplayMessage_t *cmd,
                                             uint8 disableDefaultRsp,
                                             uint8 seqNum );
```

### 3.16.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Message command. The structure of the command is as follows. Please refer to Reference 2 Section Display Message for details of each field.

```
typedef struct
{
  uint32  messageId;
  zclMessageCtrl_t messageCtrl;
  uint32  startTime;
  uint16  durationInMinutes;
  UTF8String_t msgString;
} zclCCDisplayMessage_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

### 3.16.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

## 3.17 Send Cancel Message Command (Message)

### 3.17.1 Description

This function is used to send out a Cancel Message Command to cancel an existing message.

### 3.17.2 Prototype

```
ZStatus_t zclSE_Message_Send_CancelMessage( uint8 srcEP,
                                            afAddrType_t *dstAddr,
                                            uint32 msgId,
                                            uint8 msgCtrl,
```

 

```
                                                uint8 disableDefaultRsp,
                                                uint8 seqNum );
```

### 3.17.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`msgId` – A unique unsigned 32 bit number identifier for the message being cancelled.

`msgCtrl` – An enumerated field indicating the optional ability to pass the cancel message request onto the Anonymous Inter-PAN transmission mechanism.. Please refer to Reference 2Section Cancel Message for description of this field

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

### 3.17.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

## 3.18 Send Get Last Message Command (Message)

### 3.18.1 Description

This function is used to send out a Get Last Message Command. On receipt of this command, the device shall send a Display Message command

### 3.18.2 Prototype

```
ZStatus_t zclSE_Message_Send_GetLastMessage( uint8 srcEP,
                                             afAddrType_t *dstAddr,
                                             uint8 disableDefaultRsp,
                                             uint8 seqNum );
```

### 3.18.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

### 3.18.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

## 3.19 Send Message Confirmation Command (Message)

 

### 3.19.1 Description

This function is used to send out a Message Confirmation Command to acknowledge a previously received Display Message command or Cancel Message command.

### 3.19.2 Prototype

```
ZStatus_t zclSE_Message_Send_MessageConfirmation( uint8 srcEP,
                                                  afAddrType_t *dstAddr,
                                                  uint32 msgId,
                                                  uint32 confirmTime,
                                                  uint8 disableDefaultRsp,
                                                  uint8 seqNum );
```

### 3.19.3 Parameter Details

`srcEP` – The source endpoint.

`dstAddr` - The destination address.

`msgId` – A unique unsigned 32 bit number identifier for the message being confirmed.

`confirmTime` – Confirmation Time.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - The identification number for the transaction.

### 3.19.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

## 3.20 Send Load Control Event (Load Control)

### 3.20.1 Description

This function is used to send out a Load Control Event to schedule a load control event.

### 3.20.2 Prototype

```
ZStatus_t zclSE_LoadControl_Send_LoadControlEven( uint8 srcEP,
                                                  afAddrType_t *dstAddr,
                                                  zclCCLoadControlEvent_t* cmd,
                                                  uint8 disableDefaultRsp,
                                                  uint8 seqNum );
```

### 3.20.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Load Control Event. The structure of the command is as follows. Please refer to Reference 2 Section Load Control Event for details of each field.

```
typedef struct
{
```

 

```
        uint32  issuerEvent;
        uint24  deviceGroupClass;
        uint32  startTime;
        uint16  durationInMinutes;
        uint8   criticalityLevel;
        uint8   coolingTemperatureOffset;
        uint8   heatingTemperatureOffset;
        uint16  coolingTemperatureSetPoint;
        uint16  heatingTemperatureSetPoint;
        int8    averageLoadAdjustmentPercentage;
        uint8   dutyCycle;
        uint8   eventControl;
      } zclCCLoadControlEvent_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.20.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.21 Send Cancel Load Control Event (Load Control)

#### 3.21.1 Description

This function is used to send out a Cancel Load Control Event to cancel a scheduled load control event.

#### 3.21.2 Prototype

```
ZStatus_t zclSE_LoadControl_Send_CancelLoadControlEven( uint8 srcEP,
                                        afAddrType_t *dstAddr,
                                        zclCCCancelLoadControlEvent_t* cmd,
                                        uint8 disableDefaultRsp,
                                        uint8 seqNum );
```

#### 3.21.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Cancel Load Control Event. The structure of the command is as follows. Please refer to Reference 2 Section Cancel Load Control Event for details of each field.

```
      typedef struct
      {
```

 

```
        uint32 issuerEventID;
        uint24 deviceGroupClass;
        uint8  cancelControl;
        uint32 effectiveTime;
      } zclCCCancelLoadControlEvent_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.21.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.22 Send Cancel All Load Control Event (Load Control)

#### 3.22.1 Description

This function is used to send out a Cancel All Load Control Event to cancel all scheduled load control event.

#### 3.22.2 Prototype

```
ZStatus_t zclSE_LoadControl_Send_CancelAllLoadControlEvent( uint8 srcEP,
                                                    afAddrType_t *dstAddr,
                                                    uint8 cancelControl,
                                                    uint8 disableDefaultRsp,
                                                    uint8 seqNum );
```

#### 3.22.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cancelControl` – Cancel Control bit field. Bit 0 is used when the Event is currently in process and acted upon as specified by the Effective Time field of the Cancel Load Control Event command. A value of 0 indicates that randomization is overridden and the event should be terminated immediately at the Effective Time. A value of 1 indicates the event should end using randomization settings in the original event.

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.22.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.23 Send Report Event Status (Load Control)

 

### 3.23.1 Description

This function is used to send out a Report Event Status .This command is generated when the device detects a change of state for an active Load Control event.

### 3.23.2 Prototype

```
ZStatus_t zclSE_LoadControl_Send_ReportEventStatus( uint8 srcEP,
                                                    afAddrType_t *dstAddr,
                                                    zclCCReportEventStatus_t* cmd,
                                                    uint8 disableDefaultRsp,
                                                    uint8 seqNum );
```

### 3.23.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Report Event Status command. . The structure of the command is as follows. Please refer to Reference 2 Section Report Event Status for details of each field.

```
typedef struct
{
  uint32 issuerEventID;
  uint32 eventStartTime;
  uint8  eventStatus;
  uint8  criticalityLevelApplied;
  uint16 coolingTemperatureSetPointApplied;
  uint16 heatingTemperatureSetPointApplied;
  int8   averageLoadAdjustment;
  uint8  dutyCycleApplied;
  uint8  eventControl;
  uint8  signatureType;
  uint8  signature[SE_PROFILE_SIGNATURE_LENGTH];
} zclCCReportEventStatus_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

### 3.23.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

## 3.24 Send Select Available Emergency Credit Command (Prepayment)

### 3.24.1 Description

This function is used to send out a Select Available Emergency Credit command.

 

### 3.24.2 Prototype

```
ZStatus_t zclSE_Prepayment_Send_SelAvailEmergencyCredit (uint8 srcEP,
                                        afAddrType_t *dstAddr,
                                        zclCCSelAvailEmergencyCredit_t *cmd,
                                        uint8 disableDefaultRsp,
                                        uint8 seqNum );
```

### 3.24.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Select Available Emergency Credit command. . The structure of the command is as follows. Please refer to Reference 2 Select Available Emergency Credit command for details of each field.

```
typedef struct
{
  uint32 commandDateTime;
  uint8  originatingDevice;
  UTF8String_t siteId;
  UTF8String_t meterSerialNumber;
} zclCCSelAvailEmergencyCredit_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

### 3.24.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

## 3.25 Send Change Supply Command (Prepayment)

### 3.25.1 Description

This function is used to send out a Change Supply command.

### 3.25.2 Prototype

```
ZStatus_t zclSE_Prepayment_Send_ChangeSupply( uint8 srcEP,
                                              afAddrType_t *dstAddr,
                                              zclCCChangeSupply_t *cmd,
                                              uint8 disableDefaultRsp,
                                              uint8 seqNum );
```

### 3.25.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Change Supply command. . The structure of the command is as follows. Please refer to Reference 2 Select Change Supply command for details of each field.

 

```
typedef struct
{
  uint32 providerId;
  uint32 requestDateTime;
  UTF8String_t siteId;
  UTF8String_t meterSerialNumber;
  uint32 implementationDateTime;
  uint8 proposedSupplyStatus;
  uint8 origIdSupplyControlBits;
} zclCCChangeSupply_t;
```

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.25.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.26 Send Supply Status Response (Prepayment)

#### 3.26.1 Description

This function is used to send out a Supply Status Response command.

#### 3.26.2 Prototype

```
ZStatus_t zclSE_Prepayment_Send_SupplyStatusResponse( uint8 srcEP,
                                        afAddrType_t *dstAddr,
                                        zclCCSupplyStatusResponse_t *cmd,
                                        uint8 disableDefaultRsp,
                                        uint8 seqNum );
```

#### 3.26.3 Parameter Details

`srcEP` – Sending application's endpoint.

`dstAddr` - Where you want the message to go.

`cmd` – Supply Status Response command. . The structure of the command is as follows. Please refer to Reference 2 Supply Status Response command for details of each field.

```
typedef struct
{
  uint32 providerId;
  uint32 implementationDateTime;
  uint8 supplyStatus;
} zclCCSupplyStatusResponse_t;
```

 

`disableDefaultRsp` - Disable Default Response command.

`seqNum` - ZCL sequence number.

#### 3.26.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.27 Register Command Callbacks

#### 3.27.1 Description

This callback is called to register an application's command callbacks.

#### 3.27.2 Prototype

```
ZStaus_t zclSE_RegisterCmdCallbacks( uint8 endpoint,
                                     zclSE_AppCallbacks_t *callbacks );
```

#### 3.27.3 Parameter Details

`srcEP` – Application's endpoint.

`callbacks` - pointer to the callback record defined in Section 2.2.

#### 3.27.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 3.28 Get Profile Command Callback

#### 3.28.1 Description

This callback is called to process an incoming Get Profile Command. On receipt of this command, the device responds with Get Profile Response.

#### 3.28.2 Prototype

```
typedef void (*zclSE_SimpleMeter_GetProfileCmd_t)( zclCCGetProfileCmd_t*pCmd,
                                                   afAddrType_t *srcAddr,
                                                   uint8 seqNum );
```

#### 3.28.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint8   channel;
```

 

```
  uint32  endTime;
  uint8   numOfPeriods;
} zclCCGetProfileCmd_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.28.4 Return

None.

### 3.29 Get Profile Response Callback

#### 3.29.1 Description

This callback is called to process an incoming Get Profile Response.

#### 3.29.2 Prototype

```
typedef void (*zclSE_SimpleMeter_GetProfileRsp_t)( zclCCGetProfileRsp_t*pCmd,
                                                   afAddrType_t *srcAddr,
                                                   uint8 seqNum );
```

#### 3.29.3 Parameter Details

`pCmd` – Received response. The structure of the command is as follows:

```
typedef struct
{
  uint32  endTime;
  uint8   status;
  uint8   profileIntervalPeriod;
  uint8   numOfPeriodDelivered;
  uint24 *intervals;
} zclCCGetProfileRsp_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.29.4 Return

None.

 

### 3.30 Request Mirror Command Callback

#### 3.30.1 Description

This callback is called to process an incoming Request Mirror Command. On receipt of this command, the device finds a mirroring endpoint and responds with a Request Mirror Response.

#### 3.30.2 Prototype

```
typedef void (*zclSE_SimpleMeter_ReqMirrorCmd_t)( afAddrType_t *srcAddr,
                                                  uint8 seqNum );
```

#### 3.30.3 Parameter Details

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.30.4 Return

None.

### 3.31 Request Mirror Response Callback

#### 3.31.1 Description

This callback is called to process an incoming Request Mirror Response.

#### 3.31.2 Prototype

```
typedef void (*zclSE_SimpleMeter_ReqMirrorRsp_t)( zclCCReqMirrorRsp_t *pRsp,
                                                  afAddrType_t *srcAddr,
                                                  uint8 seqNum );
```

#### 3.31.3 Parameter Details

`pRsp` – Received response. The structure of the command is as follows:

```
typedef struct
{
  uint16 endpointId ;
} zclCCReqMirrorRsp_t ;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.31.4 Return

None.

 

## 3.32 Mirror Remove Command Callback

### 3.32.1 Description

This callback is called to process an incoming Mirror Removed Command. On receipt of this command, the device removes the mirrored data from the metering device and responds with a Mirror Removed response.

### 3.32.2 Prototype

```
typedef void (*zclSE_SimpleMeter_ MirrorRemCmd_t)( afAddrType_t *srcAddr,
                                                   uint8 seqNum );
```

### 3.32.3 Parameter Details

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

### 3.32.4 Return

None.

## 3.33 Mirror Remove Response Callback

### 3.33.1 Description

This callback is called to process an incoming Mirror Removed Response.

### 3.33.2 Prototype

```
typedef void (*zclSE_SimpleMeter_ MirrorRemRsp_t)( zclCCMirrorRemRsp_t *pRsp,
                                                   afAddrType_t *srcAddr,
                                                   uint8 seqNum );
```

### 3.33.3 Parameter Details

`pRsp` – Received response. The structure of the command is as follows:

```
typedef struct
{
  uint16 endpointId ;
} zclCCMirrorRemRsp_t ;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

 

### 3.33.4 Return

None.

## 3.34 Request Fast Poll Mode Command Callback

### 3.34.1 Description

This callback is called to process an incoming Request Fast Poll Mode Command. On receipt of this command, the device starts fast poll mode and responds with a Request Fast Poll Mode response.

### 3.34.2 Prototype

```
typedef void (*zclSE_SimpleMeter_ReqFastPollModeCmd_t)(
                                          zclCCReqFastPollModeCmd_t *pCmd,
                                          afAddrType_t *srcAddr,
                                          uint8 seqNum );
```

### 3.34.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint8 fastPollUpdatePeriod;
  uint8 duration;
} zclCCReqFastPollModeCmd_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

### 3.34.4 Return

None.

## 3.35 Request Fast Poll Mode Response Callback

### 3.35.1 Description

This callback is called to process an incoming Request Fast Poll Mode Response.

### 3.35.2 Prototype

```
typedef void (*zclSE_SimpleMeter_ReqFastPollModeRsp_t)(
                                            zclCCReqFastPollModeRsp_t *pRsp,
                                            afAddrType_t *srcAddr,
                                            uint8 seqNum );
```

 

### 3.35.3 Parameter Details

pRsp – Received response. The structure of the command is as follows:

```
typedef struct
{
  uint8 appliedUpdatePeriod;
  uint32 fastPollModeEndTime;
} zclCCReqFastPollModeRsp_t;
```

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

### 3.35.4 Return

None.

## 3.36 Get Current Price Command Callback

### 3.36.1 Description

This callback is called to process an incoming Get Current Price command. On receipt of this command, the device responds with Publish Price.

### 3.36.2 Prototype

```
typedef void (*zclSE_ Pricing_GetCurentPrice_t)( zclCCGetCurrentPrice_t *pCmd
                                                 afAddrType_t *srcAddr,
                                                 uint8 seqNum );
```

### 3.36.3 Parameter Details

pCmd – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint8 option;
} zclCCGetCurrentPrice_t;
```

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

### 3.36.4 Return

None.

 

### 3.37 Get Scheduled Price Command Callback

#### 3.37.1 Description

This callback is called to process an incoming Get Scheduled Price Command. On receipt of this command, the device responds with Publish Price.

#### 3.37.2 Prototype

```
typedef void (*zclSE_Pricing_GetScheduledPrice_t)(
                                              zclCCGetScheduledPrice_t *pCmd,
                                              afAddrType_t *srcAddr,
                                              uint8 seqNum );
```

#### 3.37.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 startTime;
  uint8  numEvents;
} zclCCGetScheduledPrice_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.37.4 Return

None.

### 3.38 Publish Price Command Callback

#### 3.38.1 Description

This callback is called to process an incoming Publish Price Command.

#### 3.38.2 Prototype

```
typedef void (*zclSE_Pricing_PublishPrice_t)( zclCCPublishPrice_t *pCmd
                                              afAddrType_t *srcAddr,
                                              uint8 seqNum );
```

#### 3.38.3 Parameter Details

`pCmd` – Received command. See section 8.1.1 for description of changes to the structure. The structure of the command is as follows:

 

```
typedef struct
{
  uint32  providerId;
  UTF8String_t rateLabel;
  uint32  issuerEventId;
  uint32  currentTime;
  uint8   unitOfMeasure;
  uint16  currency;
  uint8   priceTrailingDigit;
  uint8   numberOfPriceTiers;
  uint32  startTime;
  uint16  durationInMinutes;
  uint32  price;
  uint8   priceRatio;
  uint32  generationPrice;
  uint8   generationPriceRatio;
  uint32  alternateCostDelivered;
  uint8   alternateCostUnit;
  uint8   alternateCostTrailingDigit;
  uint8   numberOfBlockThresholds;
  uint8   priceControl;
} zclCCPublishPrice_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.38.4 Return

None.

### 3.39 Price Acknowledgment Command Callback

#### 3.39.1 Description

This callback is called to process an incoming Price Acknowledgment Command.

#### 3.39.2 Prototype

```
typedef void (*zclSE_Pricing_PriceAcknowledgement_t)(
                                          zclCCPriceAcknowledgement_t *pCmd,
                                          afAddrType_t *srcAddr,
```

 

```
                                        uint8 seqNum );
```

### 3.39.3 Parameter Details

pCmd – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 providerId;
  uint32 issuerEventId;
  uint32 priceAckTime;
  uint8  control;
} zclCCPriceAcknowledgement_t;
```

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

### 3.39.4 Return

None.

## 3.40 Get Block Period Command Callback

### 3.40.1 Description

This callback is called to process an incoming Get Block Period Command. On receipt of this command, the device responds with Publish Block Period.

### 3.40.2 Prototype

```
typedef void (*zclSE_ Pricing_GetBlockPeriod_t)( zclCCGetBlockPeriod_t *pCmd
                                                 afAddrType_t *srcAddr,
                                                 uint8 seqNum );
```

### 3.40.3 Parameter Details

pCmd – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 startTime;
  uint8  numEvents;
} zclCCGetBlockPeriod_t;
```

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

 

### 3.40.4 Return

None.

## 3.41 Publish Block Period Command Callback

### 3.41.1 Description

This callback is called to process an incoming Publish Block Period Command.

### 3.41.2 Prototype

```
typedef void (*zclSE_ Pricing_PublishBlockPeriod_t)(
                                              zclCCPublishBlockPeriod_t *pCmd
                                              afAddrType_t *srcAddr,
                                              uint8 seqNum );
```

### 3.41.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 providerId;
  uint32 issuerEventId;
  uint32 blockPeriodStartTime;
  uint24 blockPeriodDurInMins;
  uint8  numPriceTiersAndBlock;
  uint8  blockPeriodControl;
} zclCCPublishBlockPeriod_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

### 3.41.4 Return

None.

## 3.42 Display Message Command Callback

### 3.42.1 Description

 

This callback is called to process an incoming Display Message Command.

### 3.42.2 Prototype

```
typedef void (*zclSE_Message_DisplayMessage_t)( zclCCDisplayMessage_t *pCmd
                                                afAddrType_t *srcAddr,
                                                uint8 seqNum );
```

### 3.42.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32  messageId;
  zclMessageCtrl_t messageCtrl;
  uint32  startTime;
  uint16  durationInMinutes;
  UTF8String_t msgString;
} zclCCDisplayMessage_t;
```

messageCtrl field is as follows:

```
typedef struct
{
  uint8 transmissionMode;     // valid value  0~2
  uint8 importance;           // 0~3
  uint8 confirmationRequired; // 0~1
} zclMessageCtrl_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

### 3.42.4 Return

None.

## 3.43 Cancel Message Command Callback

### 3.43.1 Description

This callback is called to process an incoming Cancel Message Command.

### 3.43.2 Prototype

 

```
typedef void (*zclSE_Message_CancelMessage_t)( zclCCCancelMessage_t *pCmd
                                               afAddrType_t *srcAddr,
                                               uint8 seqNum );
```

### 3.43.3 Parameter Details

pCmd – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32  messageId;
  zclMessageCtrl_t messageCtrl;
} zclCCCancelMessage_t;
```

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

### 3.43.4 Return

None.

## 3.44 Get Last Message Command Callback

### 3.44.1 Description

This callback is called to process an incoming Get Last Message command. On receipt of this command, the device responds with Display Message.

### 3.44.2 Prototype

```
typedef void (*zclSE_Message_ GetLastMessage_t)( afAddrType_t *srcAddr,
                                                 uint8 seqNum );
```

### 3.44.3 Parameter Details

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

### 3.44.4 Return

None.

## 3.45 Message Confirmation Command Callback

 

### 3.45.1 Description

This callback is called to process an incoming Message Confirmation Command.

### 3.45.2 Prototype

```
typedef void (*zclSE_Message_ MessageConfirmation_t)(
                                              zclCCMessageConfirmation_t *pCmd,
                                              afAddrType_t *srcAddr,
                                              uint8 seqNum );
```

### 3.45.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32  messageId;
  uint32  confirmTime;
} zclCCMessageConfirmation_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

### 3.45.4 Return

None.

## 3.46 Load Control Event Callback

### 3.46.1 Description

This callback is called to process an incoming Load Control Event Command.

### 3.46.2 Prototype

```
typedef void (*zclSE_LoadControl_LoadControlEvent_t)(
                                                zclCCLoadControlEvent_t *pCmd,
                                                afAddrType_t *srcAddr,
                                                uint8 seqNum );
```

### 3.46.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32  issuerEvent;
```

 

```
  uint24  deviceGroupClass;
  uint32  startTime;
  uint16  durationInMinutes;
  uint8   criticalityLevel;
  uint8   coolingTemperatureOffset;
  uint8   heatingTemperatureOffset;
  uint16  coolingTemperatureSetPoint;
  uint16  heatingTemperatureSetPoint;
  int8    averageLoadAdjustmentPercentage;
  uint8   dutyCycle;
  uint8   eventControl;
} zclCCLoadControlEvent_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.46.4 Return

None.

### 3.47 Cancel Load Control Event Callback

#### 3.47.1 Description

This callback is called to process an incoming Cancel Load Control Event Command.

#### 3.47.2 Prototype

```
typedef void (*zclSE_LoadControl_CancelLoadControlEvent_t)(
                                        zclCCCancelLoadControlEvent_t *pCmd,
                                        afAddrType_t *srcAddr,
                                        uint8 seqNum );
```

#### 3.47.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 issuerEventID;
  uint24 deviceGroupClass;
  uint8  cancelControl;
  uint32 effectiveTime;
```

 

```
} zclCCCancelLoadControlEvent_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.47.4 Return

None.

### 3.48 Cancel All Load Control Events Callback

#### 3.48.1 Description

This callback is called to process an incoming Cancel All Load Control Events Command.

#### 3.48.2 Prototype

```
typedef void (*zclSE_LoadControl_CancelAllLoadControlEvents_t)(
                                        zclCCCancelLoadControlEvents_t *pCmd,
                                        afAddrType_t *srcAddr,
                                        uint8 seqNum);
```

#### 3.48.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint8 cancelControl;
} zclCCCancelAllLoadControlEvents_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.48.4 Return

None.

### 3.49 Report Event Status Callback

#### 3.49.1 Description

This callback is called to process an incoming Report Event Status Command.

 

### 3.49.2 Prototype

```
typedef void (*zclSE_LoadControl_ReportEventStatus_t)(
                                              zclCCReportEventStatus_t *pCmd,
                                              afAddrType_t *srcAddr,
                                              uint8 seqNum );
```

### 3.49.3 Parameter Details

pCmd – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 issuerEventID;
  uint32 eventStartTime;
  uint8  eventStatus;
  uint8  criticalityLevelApplied;
  uint16 coolingTemperatureSetPointApplied;
  uint16 heatingTemperatureSetPointApplied;
  int8   averageLoadAdjustment;
  uint8  dutyCycleApplied;
  uint8  eventControl;
  uint8  signatureType;
  uint8  signature[SE_PROFILE_SIGNATURE_LENGTH];
} zclCCReportEventStatus_t;
```

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

### 3.49.4 Return

None.

## 3.50 Get Scheduled Event Callback

### 3.50.1 Description

This callback is called to process an incoming Get Scheduled Event Command.

### 3.50.2 Prototype

```
typedef void (*zclSE_LoadControl_GetScheduledEvent_t)(
                                              zclCCGetScheduledEvent_t *pCmd,
                                              afAddrType_t *srcAddr,
                                              uint8 seqNum);
```

 

### 3.50.3 Parameter Details

pCmd – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 startTime;
  uint8  numEvents;
} zclCCGetScheduledEvent_t;
```

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

### 3.50.4 Return

None.

## 3.51 Select Available Emergency Credit Command Callback

### 3.51.1 Description

This callback is called to process an incoming Select Available Emergency Credit Command.

### 3.51.2 Prototype

```
typedef void (*zclSE_Prepayment_SelAvailEmergencyCredit_t)(
                                    zclCCSelAvailEmergencyCredit_t *pCmd,
                                    afAddrType_t *srcAddr,
                                    uint8 seqNum);
```

### 3.51.3 Parameter Details

pCmd – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 commandDateTime;
  uint8  originatingDevice;
  UTF8String_t siteId;
  UTF8String_t meterSerialNumber;
} zclCCSelAvailEmergencyCredit_t;
```

srcAddr – Requestor's address.

seqNum – ZCL sequence number.

 

#### 3.51.4 Return

None.

### 3.52 Change Supply Command Callback

#### 3.52.1 Description

This callback is called to process an incoming Change Supply Command.

#### 3.52.2 Prototype

```
typedef void (*zclSE_Prepayment_ChangeSupply_t)(
                                        zclCCSupplyStatusResponse_t *pCmd,
                                        afAddrType_t *srcAddr,
                                        uint8 seqNum);
```

#### 3.52.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 providerId;
  uint32 requestDateTime;
  UTF8String_t siteId;
  UTF8String_t meterSerialNumber;
  uint32 implementationDateTime;
  uint8 proposedSupplyStatus;
  uint8 origIdSupplyControlBits;
} zclCCChangeSupply_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

#### 3.52.4 Return

None.

### 3.53 Supply Status Response Command Callback

#### 3.53.1 Description

This callback is called to process an incoming Supply Status Response Command.

 

### 3.53.2 Prototype

```
typedef void (*zclSE_Prepayment_SupplyStatusResponse_t)(
                                        zclCCSupplyStatusResponse_t *pCmd,
                                        afAddrType_t *srcAddr,
                                        uint8 seqNum);
```

### 3.53.3 Parameter Details

`pCmd` – Received command. The structure of the command is as follows:

```
typedef struct
{
  uint32 providerId;
  uint32 implementationDateTime;
  uint8 supplyStatus;
} zclCCSupplyStatusResponse_t;
```

`srcAddr` – Requestor's address.

`seqNum` – ZCL sequence number.

### 3.53.4 Return

None.

## 4. General Functional Domain – SE Security

### 4.1 Introduction

The following new cluster is added to the existing General functional domain clusters (listed in [2] Section 3):

- Key Establishment

To enable this feature in Z-Stack, compiler flag ZCL_KEY_ESTABLISH defined in the ZCL linker control file *f8wZCL.cfg* needs to be enabled.

To handle the key establishment message sequence, a new task called the Key Establishment Task is created. The application is responsible for initiating the key establishment with a partner device. After the initiation, the Key Establishment Task handles the key establishment message sequence. Upon the key establishment completion, the Key Establishment Task sends a ZCL Key Establishment Completion Indication (`ZCL_KEY_ESTABLISH_IND`) OSAL message to the application to indicate the completion.

This Key Establishment cluster consists of a group of attributes and commands. Detailed attribute list and command frame format are listed in [1] Annex C. The sequence of the commands during a successful key establishment session is illustrated in [1] Section Key Establishment Cluster. When key establishment procedure fails, a Terminate Key Establishment Command will be sent out, this command and status field are described in [1] section Terminate Key Establishment Command . This cluster is implemented in *zcl_ key_establish.c* and *zcl_ key_establish.h* files.

### 4.2 SE Secure Joining

 

The SE Profile requires that all devices have a pre-configured Trust Center Link Key and that the network key is delivered to joining devices secured with that link key. There are basically 2 joining scenarios for a SE Profile device.

When a device joins the network, but its parent isn't the Trust Center, the transport key command is tunneled from the Trust Center, through the parent of the joining device, to the joining device.

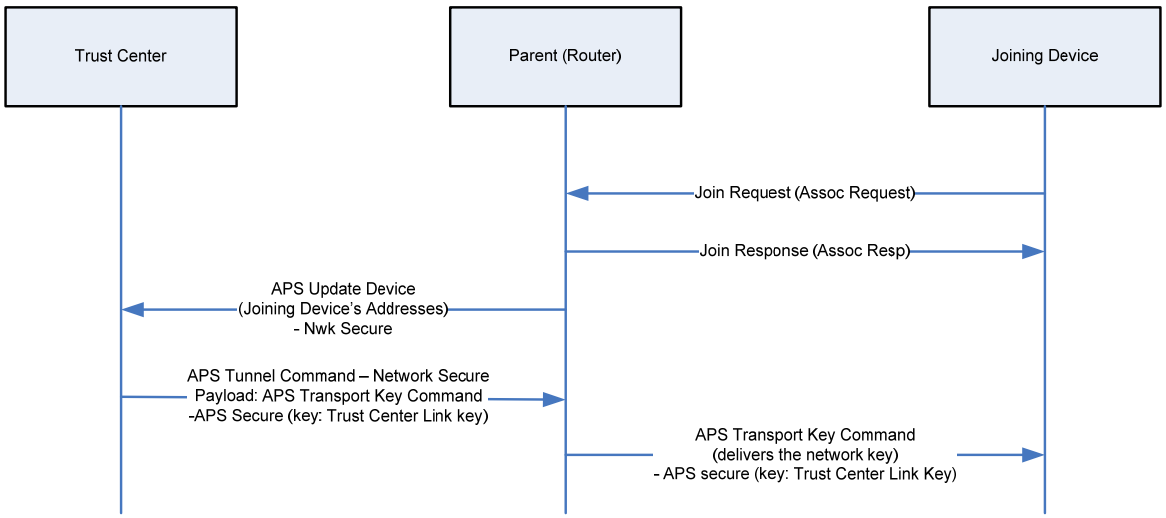


**Figure 2: SE Joining the network**

When a device joins the network, and its parent is the Trust Center, the transport key command is encrypted in the pre-configured Trust Center Link key.

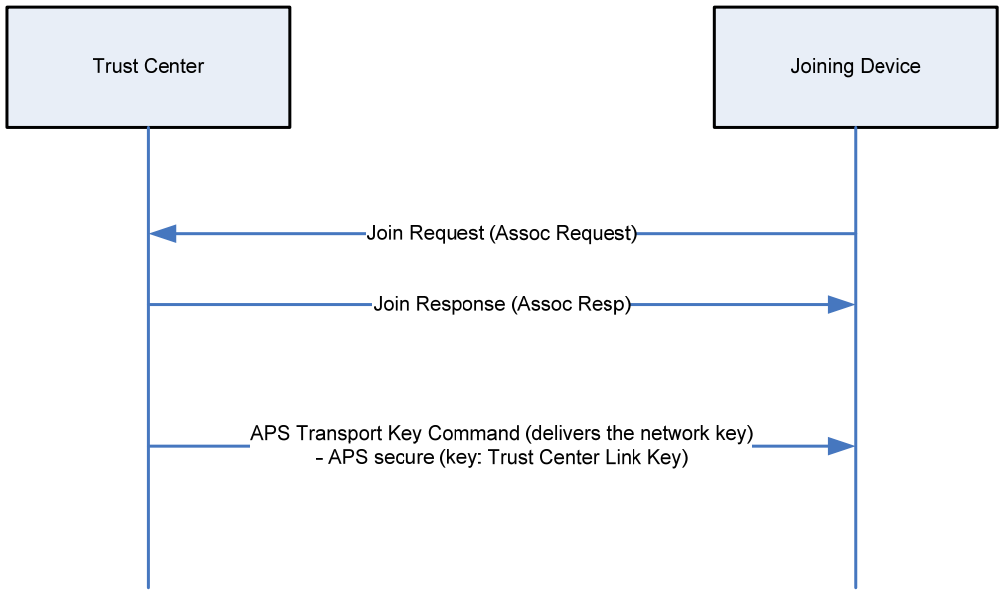


**Figure 3: SE Joining the Trust Center**

To enable the SE Secure Joining feature, set SECURE=1 in f8wConfig.cfg and include the SE_PROFILE compile flag. Also, there are needed compiler flags, global variables (ZGlobals) and NV Items:

- zgPreConfigTCLinkKey (defined in ZGlobals) is used as the Pre-Configured Trust Center Link Key. The NV Item for this global is ZCD_NV_SECURE_PRECFG_TCLINKKEY (defined in ZComDef.h). This global is initialized with defaultTCLinkKey (defined in nwk_globals).
- The Trust Center Link Key parameters for RX and TX frame counters are also stored in NV as (defined in ZComDef.h):
    - ZCD_NV_SECURE_TCLINKKEY_TXFRAME
    - ZCD_NV_SECURE_TCLINKKEY_RXFRAME

## 4.3 Register Key Establishment Task with OSAL

As mentioned in the previous section, key establishment messages are handled by a separate task: Key Establishment Task. Therefore, the application needs to register Key Establishment Task with OSAL to support the key establishment cluster. The registration consists of two parts: initialize the task and specify the task event handler. They are described in details in the following subsections.

### 4.3.1 Initialize Key Establishment Task (Key Establishment)

#### 4.3.1.1 Description

This function is used to initialize the Key Establishment Task, which is responsible for handling the key establishment message sequence and notifying the application when the key establishment is completed. This function shall be called inside function *osalInitTasks()*, which is normally defined in *OSAL_<app name>.c*

#### 4.3.1.2 Prototype

```
void zclGeneral_KeyEstablish_Init(uint8 task_id );
```

#### 4.3.1.3 Parameter Details

`task_id` – Task ID.

### 4.3.2 Specify the Task Event Handler

#### 4.3.2.1 Description

This function is used to specify the Key Establishment Task event handler, which is responsible for handling events set for the key establishment task and OSAL messages sent to the task This function shall be added to the *tasksArr[*] table, which is normally defined in *OSAL_<app name>.c.* This function is called from OSAL directly and user does not need to pass in the parameters.

#### 4.3.2.2 Prototype

```
uint16 zclKeyEstablish_event_loop( uint8 task_id, uint16 events )
```

## 4.4 Initiate Key Establishment (Key Establishment)

 

#### 4.4.1 Description

This function is called to initiate key establishment with a partner device. The initiating application will be notified when the key establishment is completed.

#### 4.4.2 Prototype

```
ZStatus_t zclGeneral_KeyEstablish_InitiateKeyEstablishment(
                                                    uint8 appTaskID,
                                                    afAddrType_t *partnerAddr,
                                                    uint8 seqNum );
```

#### 4.4.3 Parameter Details

`appTaskID` – Task ID of the application that initiates the key establishment.

`partnerAddr` - short address and endpoint of the partner to establish key with.

`seqNum` – sequence number of application (ZCL).

#### 4.4.4 Return

`ZStatus_t` – status definitions found in ZComDef.h.

### 4.5 Application Link Key NV Management for Sleeping End Device

#### 4.5.1 Introduction

Sleeping end devices need to store all critical information in Non-Volatile memory during the power down period, and restore all the information when they power back up. This section explains the NV management for application link key information.

#### 4.5.2 Link Key List

The link key list is an array of link key entries. Each link key entry includes information of partner address, application link key between the local device and the partner device, last transmitted frame counter to the partner device and last received frame counter from the partner device, as well as authentication flag to indicate whether the partner device has been authenticated or not (only applies to trust center). :

#### 4.5.3 Save Off Link Key List to NV

##### 4.5.3.1 Description

This function is called to save off the link key list to NV. Due to limited flash memory writing times, Z-Stack do not periodically save link key list to NV, instead, it leave it to the application to call this function as needed.

##### 4.5.3.2 Prototype

void ZDSecMgrWriteNV( void )

 

### 4.5.4 Restore Link Key List from NV

#### 4.5.4.1 Description

Link Key List is restored from the NV during the initialization of ZDO Security manager by Z-Stack unless implemented otherwise in the application. Since the link key list is not always in sync with the NV. A constant MAX_APS_FRAMECOUNTER_CHANGES defined in *ZDSecMgr.c* is added to the transmission frame counter of each entry in the list, which is defined as the number of times the frame counter can change before saving to NV. This constant shall be configured based on the frequency of sending packets from the local device to the partner device.

## 4.6 Communicating With Non Trust Center Devices

When a device needs to exchange APS messages with another device, and neither of them is the Trust Center, and these messages have to be encrypted with APS security, one of them should request an APS key from the Trust Center. The Trust Center will generate an APS key that will be sent to both devices. Once both devices have the newly generated key, they will use it to exchange SE messages with APS security.

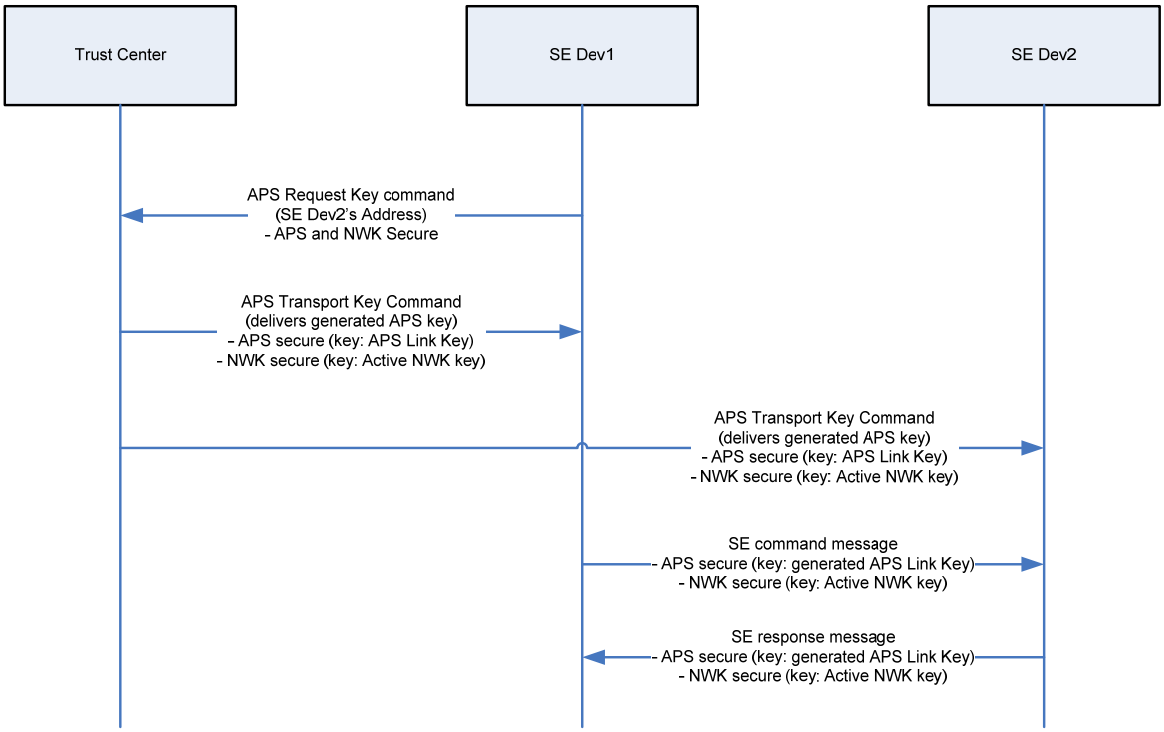


**Figure 4: SE Request APS link key to Trust Center**

### 4.6.1 Request APS Link Key

#### 4.6.1.1 Description

This function is called to request an APS link key from the Trust Center. A pointer to the partner extended address must be specified as a parameter.

 

#### 4.6.1.2 Prototype

```
ZStatus_t ZDSecMgrRequestAppKey( uint8 *partExtAddr )
```

#### 4.6.1.3 MT Command

The MT command, UTIL_APSME_REQUEST_KEY_CMD, can be used to request the APS Link Key. This command calls the ZDSecMgrRequestAppKey() function and returns the status. See [3] for a detailed description of the command and its parameters.

## 5. ECC Lib

Two dummy files, namely *eccapi.c* and *eccapi.h*, are provided which contain dummy functions for the ECC library. These functions serve as place holders to allow projects to compile without the required library, but will not perform the ECC functionality. The user must obtain a valid ECC library from Certicom Corp (http://www.certicom.com).

## 6. Cluster Security and APS ACK Options

Table 5.10 in 1 Section 5.4.6 indicates Security Key assignments per Cluster. The following cluster option table should be registered with the ZCL Foundation to meet the SE Security requirements:

```
/*********************************************************************
 * CLUSTER OPTION DEFINITIONS
 */
zclOptionRec_t zclTestApp_Options[] =
{
  // *** General Cluster Options ***
  {
    ZCL_CLUSTER_ID_GEN_TIME,                  // Cluster ID - defined in zcl.h
    ( AF_EN_SECURITY /*| AF_ACK_REQUEST*/ ), // Options – Found in AF.h
  },

  // *** Smart Energy Cluster Options ***
  {
    ZCL_CLUSTER_ID_SE_PRICING,
    ( AF_EN_SECURITY ),
  },
  {
    ZCL_CLUSTER_ID_SE_LOAD_CONTROL,
    ( AF_EN_SECURITY ),
  },
  {
    ZCL_CLUSTER_ID_SE_SIMPLE_METERING,
    ( AF_EN_SECURITY ),
  },
  {
    ZCL_CLUSTER_ID_SE_MESSAGE,
    ( AF_EN_SECURITY ),
  },
  {
    ZCL_CLUSTER_ID_SE_SE_TUNNELING,
    ( AF_EN_SECURITY ),
  },
  {
```

 

```
    ZCL_CLUSTER_ID_SE_PREPAYMENT,
    ( AF_EN_SECURITY ),
  },
};
```

The Cluster Option List is registered with the ZCL using `zcl_registerClusterOptionList()` API explained in [2] Section 3.21.

# 7. NV Items

In order to use the Key Establishment cluster, the user must set the following NV items:

- Local Certificate for SE CBKE (`ZCD_NV_LOCAL_CERTIFICATE`)
- Static Private Key (`ZCD_NV_STATIC_PRIVATE_KEY`)
- Certificate Authority Public Key (`ZCD_NV_CA_PUBLIC_KEY`)
- Remote Static Public Key (`ZCD_NV_STATIC_PUBLIC_KEY`)

Please refer to `ZGlobals.c` file for the definition of these NV items. The Certificate Authority (CA) Public Key is the public key paired with the CA private key. The CA uses its private key to sign the digital certificates and the CA public key is used to verify these signatures. The values of above items shall all be provided by the CA..

# 8. SE 1.1 Updates and Additions

Some updates and addition have been implemented to comply with Smart Energy 1.1 specification.

## 8.1 Updates

### 8.1.1 Publish Price Command Structure

Publish Price command structure has changed, field `priceTier` has been renamed to `numberOfPriceTiers` and five new elements have been added.

```
typedef struct
{
  uint32  providerId;
  UTF8String_t rateLabel;
  uint32  issuerEventId;
  uint32  currentTime;
  uint8   unitOfMeasure;
  uint16  currency;
  uint8   priceTrailingDigit;
  uint8   numberOfPriceTiers;               /* Updated element name */
  uint32  startTime;
  uint16  durationInMinutes;
  uint32  price;
  uint8   priceRatio;
  uint32  generationPrice;
  uint8   generationPriceRatio;
  uint32  alternateCostDelivered;           /* New element */
  uint8   alternateCostUnit;                /* New element */
  uint8   alternateCostTrailingDigit;       /* New element */
  uint8   numberOfBlockThresholds;          /* New element */
  uint8   priceControl;                     /* New element */
} zclCCPublishPrice_t;
```

 

Interoperability between devices running SE 1.0 and SE 1.1 is not affected by changes to the Publish Price structure.

| Send | Receive | Receiver Behavior |
|---|---|---|
| SE 1.0 Device | SE 1.1 Device | Verifies if length of message is SE 1.0, parses the message and fills new elements with specific values to indicate they are not used and returns ZSuccess. Sends Default Response if requested by sender. |
| SE 1.1 Device | SE 1.0 Device | Parses message, ignores new elements and returns ZSuccess. Sends Default Response if requested by sender. |

### 8.2 Additions

New structures have been added to support SE 1.1 features:

- zclCCReqFastPollModeCmd_t
- zclCCReqFastPollModeRsp_t
- zclCCPriceAcknowledgement_t
- zclCCGetBlockPeriod_t
- zclCCPublishBlockPeriod_t
- zclCCSelAvailEmergencyCredit_t
- zclCCChangeSupply_t
- zclCCSupplyStatusResponse_t

Devices running SE 1.0 do not process these commands.

### 8.3 Multiple ESI

The Smart Energy profile allows the use of multiple ESI devices in a network, see the full overview in [1] . There is only one Trust Center but SE messages can be received from more than one ESI.

#### 8.3.1 Request Link Key

Devices in the network should use APS security. When more than one ESI is present, a device acting as ESI that is not the Trust Center should request a Link Key from the Trust Center to communicate directly to other devices in the network, see section 4.6.

#### 8.3.2 Most Authoritative Time Source

Time cluster attributes have been added to *zcl_general.h* so the application can use them to implement the Most Authoritative Time Source procedure.

- ATTRID_TIME_ZONE
- ATTRID_TIME_DST_START
- ATTRID_TIME_DST_END
- ATTRID_TIME_DST_SHIFT
- ATTRID_TIME_STANDARD_TIME
- ATTRID_TIME_LOCAL_TIME
- ATTRID_TIME_LAST_SET_TIME
- ATTRID_TIME_VALID_UNTIL_TIME

#### 8.3.3 MT Commands

Time management MT commands are available to provision the device with real time, which will be used in multiple SE commands and responses:

- SYS_SET_TIME: Set the device's time.
- SYS_GET_TIME: Get the device's time.

 

See [3] for a detailed description on the commands and their parameters.

## 9. Compile Options

The ZCL compile options are defined in the ZCL configuration file *f8wZCL.cfg*, which is located in the **Tools** folder of the Z-Stack installation along with other configuration files. The *f8wZCL.cfg* file is used by all projects that include the ZCL (i.e., all Smart Energy projects). Therefore, any change made to this file will affect all SE projects. If needed, you can create a private version of the *f8wZCL.cfg* file and modify your project to use the new version. The ZCL supported compile options for Smart Energy and their definitions are listed in the following table:

| | |
|---|---|
| **ZCL_LOAD_CONTROL** | Enable the following commands:<br>1) Load Control Event<br>2) Cancel Load Control Event<br>3) Cancel All Load Control Event<br>4) Report Event Status<br>5) Get Scheduled Events Command |
| **ZCL_SIMPLE_METERING** | Enable the following commands:<br>1) Get Profile Command<br>2) Get Profile Response<br>3) Request Mirror Command<br>4) Request Mirror Response<br>5) Remove Mirror Command<br>6) Remove Mirror Response<br>7) Request Fast Poll Mode Command<br>8) Request Fast Poll Mode Response |
| **ZCL_PRICING** | Enable the following commands:<br>1) Get Current Price<br>2) Get Scheduled Price<br>3) Publish Price<br>4) Price Acknowledgment<br>5) Get Block Period<br>6) Publish Block Period |
| **ZCL_MESSAGE** | Enable the following commands:<br>1) Display Message<br>2) Cancel Message<br>3) Get Last Message<br>4) Message Confirmation |
| **ZCL_KEY_ESTABLISH** | Enable Key Establishment cluster |
| **ZCL_PREPAYMENT** | Enable the following commands:<br>1) Select Available Emergency Credit<br>2) Change Supply<br>3) Supply Status Response |

 