Freescale ZigBee™ Cluster Library (ZCL)

Reference Manual

Document Number: ZCLRM
Rev. 1.2
12/2008

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Contents

About This Book

The *ZigBee Cluster Library Reference Manual* describes the API to the ZigBee Cluster Library, an add-on component used in many ZigBee Application Profiles, including Home Automation and Smart Energy.

Audience

This document is intended for software developers who write applications for BeeStack-based products using Freescale development tools.

Organization

This document is organized into the following sections

Chapter 1	Introduction – provides an overview of the ZigBee Cluster Library, as well as its purpose and use in Home Automation. It also contains a concise list of devices, clusters and attributes and commands supported by BeeStack.
Chapter 2	ZigBee Cluster Library API – describes the API and compile-time options for the ZigBee Cluster Library.
Chapter 3	Adding Custom Devices, Clusters and Attributes – introduces the function calls, macros, and APIs available in the Application Framework (AF).

Revision History

The following table summarizes revisions to this document since the previous release (Rev. 1.1).

Revision History

Location	Revision
TBD	Update for MC1322x silicon revision.

Conventions

The *ZigBee Cluster Library Reference Manual* uses the following formatting conventions when detailing commands, parameters, and sample code:

`Courier mono-space` type indicates commands, command parameters, and code examples.

Bold style indicates the command line elements, which must be entered exactly as written.

Italic type indicates command parameters that the user must type in or replace, as well as emphasizes concepts or foreign phrases and words.

Definitions, Acronyms, and Abbreviations

AF	Application framework
API	Application programming interface
APSME	APS management entity
ASL	Application support library, a set of common user interface and application callable routines used across multiple sample applications.
Cluster	A collection of attributes associated with a specific cluster-identifier
Cluster identifier	An enumeration that uniquely identifies a cluster within an application profile
Device/Node	ZigBee network component containing a single IEEE 802.15.4 radio.
Endpoint	Component within a unit; a single IEEE 802.15.4 radio may support up to 240 independent endpoints
IEEE	Institute of Electrical and Electronics Engineers, a standards body
MAC	Medium access control sub-layer
NVM	Non-volatile memory
Octet	Eight bits of data, or one byte
Profile	Set of options in a stack or an application
ZCL	ZigBee cluster library, a set of common clusters and attributes used across multiple application profiles
ZDP	ZigBee device profile
802.15.4	An IEEE standard radio specification that underlies the ZigBee Specification

Reference Materials

This following served as references for this manual:

1. Document 053474r17, *ZigBee Specification*, ZigBee Alliance, October 2007
2. Document 075123r02, *ZigBee Cluster Library Specification*, ZigBee Alliance, May 2008
3. Document 053520r25, *Home Automation Profile Specification*, ZigBee Alliance, October 2007
4. Document 075356r14ZB_AMI_PTG-AMI_Profile_Specification.pdf, ZigBee Alliance, May 2008

Chapter 1
Introduction

This manual describes the API and how to use the ZigBee Cluster Library (ZCL), an add-on component used in many ZigBee Application Profiles, including Home Automation.

This manual assumes that the reader is already familiar with the concept of a cluster and a ZigBee node.

1.1 ZigBee Cluster Library Concepts

The ZigBee Cluster Library is a collection of clusters and attributes that have specific meaning, even across profiles. For example, there is an OnOff cluster that defines how to turn something off, on, or toggle it. This cluster works for light switches, pumps, garage door openers and any other device that can be turned on or off.

The concepts of nodes, endpoints, devices, clusters and attributes are briefly outlined on the following list. For a more complete explanation, see the Freescale *BeeStack Application Development Guide* or the *ZigBee Specification.*

- A ZigBee node contains an 802.15.4 radio, and is a single addressable entity on the ZigBee network
- ZigBee nodes contain one or more application endpoints (applications)
- A device, in ZCL terms, resides on an endpoint. Examples of a devices are an OnOffLight or a Thermostat. A single ZigBee node may contain multiple devices. For example, a single ZigBee node could control 4 separate lights, all individually controllable. An endpoint's simple descriptor describes the device on that endpoint, including the list of input and output clusters.
- Each cluster is a collection of commands and attributes. In programming terms, a cluster is an object. For example, a DimmableLight contains a level-control cluster. This cluster contains attributes that describe it's current level and another attribute that describes the transition time when going to a new level. There are commands on that cluster to turn the light off, turn the light fully on, or to step or move to any light level over time.
- Attributes are the state of the cluster, or the data. An example of an attribute for the OnOff cluster is whether the device is presently on or off. Attributes have types, may be a single byte, a 16-bit word, or even a length encoded string.

The ZigBee Cluster Library contains clusters defining many functional domains. The BeeStack implementation of the ZigBee Cluster Library contains a data and code driven way of generically defining devices, clusters and attributes, and providing multiple instantiations of a given device within a single node if desired.

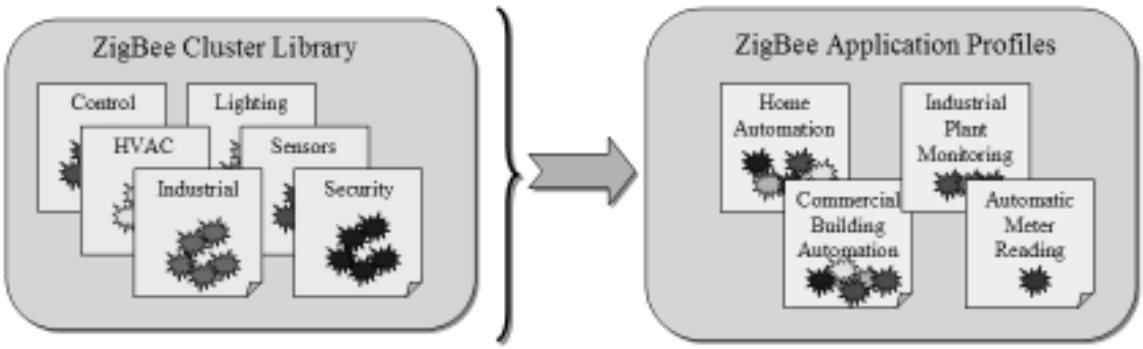

Figure 1-1. ZigBee Cluster Library

The ZigBee Cluster Library defines

- functional domains
- cluster sets for that functional domain
- mandatory and optional clusters, attributes, commands and functional descriptions
- cluster identifiers
- explicit device descriptions are not defined

Application Profiles (e.g. HA) defines

- application domains
- related elements from the cluster library collected into application domains
- device descriptions for each required device
- any specialized or deviated use of ZCL cluster

BeeStack implements the Home Automation application profile, and contains the mandatory and many of the optional clusters from that specification.

1.2 Home Automation and Smart Energy Devices

The ZigBee Home Automation and Smart Energy specification defines a set of wireless devices appropriate for a home environment. That specification is not repeated here. Instead, only the devices, clusters and attributes supported by the BeeStack ZigBee Cluster Library sample implementation are listed and described here.

NOTE

The implementation of the Home Automation and Smart Energy devices are for demonstration purposes only and should not be considered ready for production. The ZCL clusters adhere to the specification, but other system issues, such as the IEEE address, exact contents of NVM storage and other issues are left up to the OEM.

The following is a list of Home Automation and Smart Energy devices supported by BeeStack. All these applications are available as templates from the BeeKit Wireless Connectivity Toolkit software.

Table 1-1. Table 1-1.Home Automation and Smart Energy Devices

Name	Device ID	Functional Description
HaConfigurationTool	0x0005	Set up groups and reporting.
HaCombinedInterfaceDevice	0x0007	Common node used as a gateway into a network of devices. For example, this node might be attached to a PC so the PC or internet can control or detect the state of the lights in a home.
HaRangeExtender	0x0008	No special function other than routing packets.
HaOnOffLight	0x0100	Simple on/off light.
HaDimmableLight	0x0101	Dimmable light, can be controlled by on/off or dimmable switch.
HaOnOffLightSwitch	0x0103	Simple on/off light switch.
HaDimmerSwitch	0x0104	Dimmer switch can control light level on dimmable light..
HaThermostat	0x0301	Thermostat. Supports receiving temperature reports from Temperature Sensor. Can display temperatures on NCB in both Celsius and Fahrenheit.
HaTemperatureSensor	0x0302	Temperature sensor. Can send temperature reports. Uses the hardware temperature sensor on the SRB and simulated temperatures on other boards.
HaConfigurationTool	0x0005	Set up groups and reporting.
SeEnergyServicePortal	0x0500	The Energy Service Portal acts as a gateway to ZigBee network and optionally as a Metering Device. It connects the energy supply company network to the metering and energy management devices within the home.
SeInpremiseDisplay	0x0502	The In-Premise Display demo application will relay the data to the user by way of a text display. The display shows simple text messages or price information to inform the user about the utility company actions or the current price.
SeMeteringDevice	0x0501	The Metering Device demo application utilize a timer based scheme in order to demonstrate continuous energy consumption, so that metering data read through read attribute or reporting will show an increasing consumption.
SePCT SeLoadControlDevice	0x0503 0x0504	The Programmable Communicating Thermostat and Load Control demo applications show how received Demand Response and Load Control (DRLC) and Price events can be manage by a device that supports load control and price.
SeRangeExtender	0x0008	The Range Extender is a simple device that acts as a router for other devices from the network.
SeSmartAppliance SePrepaymentTerminal	0x0505, 0x0506	The Load Control, Smart Appliance, Prepayment Terminal are sample applications that have no functionality in the run mode of the application.

1.3 ZCL Clusters and Attributes

The ZigBee cluster Library supports a set of commands that work across all clusters. These commands include:

- Read attributes – read one or more attributes from a remote node
- Write attributes – write one or more attributes to a remote node
- Write attributes undivided – write one or more attributes, but only if all attributes can be written.
- Write attributes no response – write one or more attributes, but don't generate a response.
- Configure reporting – configure one or more attributes for reporting in a remote node
- Read reporting configuration – determine the attribute reporting in a remote node
- Report attributes – report one or more attributes
- Default response – sent when a command failed or there is not a specific response for a certain command
- Discover attributes - requests the list of attributes that are available for a certain cluster defined on an endpoint of the receiving device

Read attribute can be used to detect the state of a node, for example to read whether the an HaOnOffLight is currently on or off. Alternately, that data can be "pushed" from the light itself using the configure reporting command.

Writing to an attribute is usually used for configuring the node. For example, the Identify Cluster contains an attribute IdentifyTime, that can be written to causing the node to go into identify mode for that number of seconds.

The command error is used if a node tries to access a cluster, attribute or command that does not exist.

The following table lists the clusters and their attributes supported by the BeeStack ZigBee Cluster Library

Table 1-2. ZCL Clusters and Attributes

Cluster Name	Cluster ID	Description	Attributes
Basic	0x0000	Basic information common to all devices	ZCLVersion ApplicationVersion StackVersion HWVersion ManufacturerName ModelIdentifier DateCode PowerSource LocationDescription PhysicalEnvironment DeviceEnabled AlarmMask
Identify	0x0003	Services to visually identify remote nodes	IdentifyTime
Groups	0x0004	Services to add/remove groups on remote nodes (complements the APS local group services)	NameSupport

Table 1-2. ZCL Clusters and Attributes

Scenes	0x0005	Services to save and restore scenes.	SceneCount CurrentScene CurrentGroup SceneValid NameSupport LastConfiguredBy
OnOff	0x0006	Turn a device on or off	OnOff
OnOffSwitchCfg	0x0007	Configure how a switch turns a device on, off or toggle	SwitchType SwitchActions
LevelControl	0x0008	Control a device with a range, rather than simple on/off. Applies to dimmable lights, shades, etc..	CurrentLevel RemainingTime OnOffTransitionTime OnLevel
Thermostat	0x0201	Thermostat device	LocalTemperature Occupancy AbsMinHeatSetpointLimit AbsMaxHeatSetpointLimit AbsMinCoolSetpointLimit AbsMaxCoolSetpointLimit PICoolingDemand PIHeatingDemand OccupiedCoolingSetpoint OccupiedHeatingSetpoint MinHeatSetpointLimit MaxHeatSetpointLimit MinCoolSetpointLimit MaxCoolSetpointLimit MinSetpointDeadBand ControlSequenceOfOperation SystemMode LocalTemperatureCalibration UnoccupiedCoolingSetpoint UnoccupiedHeatingSetpoint RemoteSensing AlarmMask
FanControl	0x0202	Controls speed of a fan part of an HVAC system	FanMode FanModeSequence
ThermostatUiCfg	0x0204	Configure level of access to a thermostat keypad and if degrees displayed are in Celsius or Fahrenheit	TempDisplayMode KeypadLockout
TemperatureMeasurement	0x0402	Temperature sensor, outputs to thermostat	MeasuredValue MinMeasuredValued MaxMeasuredValued Tolerance

Chapter 2
ZigBee Cluster Library API

The ZigBee Cluster Library is implemented in BeeStack to be extensible and flexible. Choose any of the Home Automation templates in BeeKit as a starting point to enable the ZigBee Cluster Library framework.

2.1 ZCL API Overview

This section describes the ZCL functions available to the application. The section ZCL Requests describes all of the cluster command requests that can be made from an application. See the section ZCL_Register for a description of how to receive the results of the requests in the application.

2.1.1 ZCL_Init

Prototype

```
void ZCL_Init(void);
```

Description

Initialize the ZigBee Cluster Library. This function must be called after endpoints are registered, usually in BeeAppInit().

2.1.2 ZCL_Reset

Prototype

```
void ZCL_Reset(void);
```

Description

This will reset the ZCL engine to the original, including turning off timers and clear all scene tables, etc.. This will not reset the contents of NVM that store ZCL data.

2.1.3 ZCL_Register

Prototype

```
void ZCL_Register
  (
  fnZclResponseHandler_t fnResponseHandler  /* IN response handler */
  );
```

Description

This function registers a callback function to receive the responses from a ZCL request. Usually called in the applications BeeAppInit() function. See the `BeeApp.c` file in the HaThermostat for an example of the use of this function.

2.1.4 ZCL_InterpretFrame

Prototype

```
zbStatus_t ZCL_InterpretFrame
  (
  zbApsdeDataIndication_t *pIndication  /* IN: from application */
  );
```

Description

This function is typically called from BeeAppDataIndication() in the application. This function will automatically handle ZCL requests to read or write attributes, and will pass requests to the proper cluster handler if the endpoint supports a device definition. See Chapter 3, “ Adding Custom Devices, Clusters and Attributes” for more details on device definitions.

This function returns gZbSuccess_c if it handled the request and no further action is required on the part of the application. Returns gZclMfgSpecific_c if this was a manufacturer specific cluster or if the endpoint does not have a device or cluster definition. In the latter case, the application must process the data indication.

WARNING

ZCL_InterpretFrame() does NOT free the message buffer from the data indication. It is up to the application’s BeeAppDataIndication() function to do so.

2.1.5 ZCL Requests

The following is a list of ZCL requests that can be made of remote nodes, or the local node.

The requests generally follow the same format. The request structure begins with an afAddrInfo_t structure which describes the source and destination nodes, endpoints and cluster ID, followed by a request specific structure.

```
typedef struct afAddrInfo_tag
{
  zbAddrMode_t  dstAddrMode;    /* indirect, group, direct-16, direct-64 */
  zbApsAddr_t   dstAddr;        /* short address, long address or group */
  zbEndPoint_t  dstEndPoint;    /* destination endpoint */
  zbClusterId_t aClusterId;     /* what cluster to send from/to? */
  zbEndPoint_t  srcEndPoint;    /* what source endpoint to send from? */
  zbApsTxOption_t txOptions;    /* ACK, security options */
  uint8_t       radiusCounter;  /* radius */
} afAddrInfo_t;
```

A code-snippet example of using a request would be

```
void AppSendAddGroupIfIdentifying(void)
{
  zclGroupAddGroupIfIdentifyingReq_t request;

  /* set up destination address to broadcast this command */
  /* to all nodes, all endpoints */
  request.addrInfo.dstAddrMode = gZbAddrMode16Bit_c;
  Copy2Bytes(request.addrInfo.dstAddr, gaBroadcastAddress);
  request.addrInfo.dstEndPoint = gZbBroadcastEndPoint_c;
  Set2Bytes(request.addrInfo.aClusterId, gZclClusterGroups_c);
  request.addrInfo.srcEndPoint = appEndPoint;
  request.addrInfo.txOptions = afTxOptionsDefault_c;
  request.addrInfo.radiusCountr = afDefaultRadius_c;

  /* add group 5 if the destination node(s) are in identify mode */
  Set2Bytes(request.cmdFrame.aGroupId, 0x0500); /* group 5, little endian */
  *request.cmdFrame. szGroupName = 0;  /* no group name */
ASL_ZclGroupAddGroupIfIdentifypReq(&request);
   }
```

NOTE

All multi-byte fields, such as group ID, cluster ID, attribute ID, etc.. are set up in the structures as an array, little endian (low-byte first), because the in ZigBee specification all multi-byte fields are little endian over the air. The HCS08 is a big endian processor, so care must be taken with values that have mathematical meaning such as time.

The rest of this section describes the individual request functions, use and prototypes.

Prototype

```
zclStatus_t ASL_ZclIdentifyReq(zclIdentifyReq_t *pReq);
```

Description

Indicates that the receiving device should enter identify mode for instance by blinking a led. The parameter structure contains a 16-bit identifier containing the number of seconds the device should identify itself.

Prototype

```
zclStatus_t ASL_ZclIdentifyQueryReq(zclIdentifyQueryReq_t *pReq);
```

Description

This command queries a device to determine if it is currently in identify mode. The response also contains the number of seconds the device will still be in this mode.

The stucture indicated in the parameter does not contain a payload except for the afAddrInfo_t addressing information.

Prototype

```
zclStatus_t ASL_ZclGroupAddGroupReq(zclGroupAddGroupReq_t *pReq);
```

Description

Adds a group to an endpoint on a remote node. Equivalent to local call APSME_AddGroupRequest(), but for remote nodes.

Prototype

```
zclStatus_t ASL_ZclGroupAddGroupIfIdentifyReq
(
zclGroupAddGroupIfIdentifyingReq_t *pReq
);
```

Description

The same as ASL_ZclGroupAddGroupReq(), but only adds nodes if they are currently in identify mode.

Prototype

```
zclStatus_t ASL_ZclGroupViewGroupReq(zclGroupViewGroupReq_t *pReq);
```

Description

Returns the group information of a remote node. Make sure to use ZCL_Register() to set up a function to receive the data.

Used to determine if the remote node is a member of that group. Returns the following structure:

```
typedef struct zclCmdGroup_ViewGroupRsp_tag
{
  zclStatus_t    status;
  zbGroupId_t   aGroupId;
  char          szName[1];/* variable length name, length encoded */
} zclCmdGroup_ViewGroupRsp_t;
```

Prototype

```
zclStatus_t ASL_ZclGetGroupMembershipReq
(
zclGroupGetGroupMembershipReq_t *pReq
);
```

Description

Returns the group membership of a particular endpoint. The results are a list of group IDs that come back through the applications data indication callback.

Returns the following structure from the remote node if successful (and the node exists):

```
typedef struct zclCmdGroup_GetGroupMembershipRsp_tag
{
  haCapacity_t  capacity;     /* # of groups total */
  uint8_t       count;        /*  */
```

```
  zbGroupId_t   aGroupId[1];
} zclCmdGroup_GetGroupMembershipRsp_t;
```

Prototype

```
zclStatus_t ASL_ZclRemoveGroupReq(zclGroupRemoveGroupReq_t *pReq);
```

Description

Remove one particular group from a remote node. Only removes it from the one endpoint, unless the broadcast endpoint is used.

Prototype

```
zclStatus_t ASL_ZclRemoveAllGroupsReq(zclGroupRemoveAllGroupsReq_t *pReq);
```

Description

Remove all groups from a particular node. Only removes it from on endpoint, unless the broadcast endpoint is used.

Prototype

```
zclStatus_t ASL_ZclSceneAddSceneReq(zclSceneAddSceneReq_t *pReq,uint8_t len);
```

Description

Adds a scene. Scenes are a method for setting a node in a particular state. One way to think of this is the buttons on a car radio. Press the button, and the radio goes to a particular channel. That button can be reprogrammed to go to a different channel at the users request.

The add scene request structure is as follows

```
typedef struct zclCmdScene_AddScene_tag
{
  zbGroupId_t       aGroupId;         /* group ID */
  zclSceneId_t      sceneId;          /* scene ID */
  uint16_t          transitionTime;   /* in seconds */
  char              szSceneName[16];  /* length-encoded string */
  zclSceneOtaData_t aData[1];         /* variable scene length data */
} zclCmdScene_AddScene_t;
```

The following example shows setting up a scene in an HaOnOffLight, to turn the light on when this scene is recalled.

```
void ExampleAddScene(void)
{
  zclSceneAddSceneReq_t request;

  /* fill in destination address info here… */
/* … */

/* indicate scene should turn light on after 4 seconds */
Set2Bytes(request.cmdFrame, Native2OTA16(5)); /* group 5, little endian */
request.cmdFrame.sceneId = 8; /* scene 8 */
request.cmdFrame.transitionTime = Native2OTA16(0); /* no transition time */
```

```
*request.cmdFrame.szSceneName = 0; /* no scene name */
request.cmdFrame.aData[0].clusterId = gZclClusterOnOff_c;
request.cmdFrame.aData[0].length = 1;
request.cmdFrame.aData[0].aData[0] = 1;   /* turn light on */

ASL_ZclSceneAddSceneReq(&request, sizeof(zclCmdScene_AddScene_t));
}
```

Prototype

```
zclStatus_t ASL_ZclViewSceneReq(zclSceneViewSceneReq_t *pReq);
```

Description

Retrieves the scene information for a particular group and scene ID. The returned structure is

```
typedef struct zclCmdScene_ViewSceneRsp_tag
{
  zbGroupId_t       aGroupId;          /* group ID */
  zclSceneId_t      sceneId;           /* scene ID */
  char              szSceneName[16];   /* scene name */
  uint16_t          transitionTime;    /* in seconds, little endian */
  zclSceneOtaData_t aData[1];          /* variable scene length data */
} zclCmdScene_ViewSceneRsp_t;
```

Prototype

```
zclStatus_t ASL_ZclRemoveSceneReq(zclSceneRemoveSceneReq_t *pReq);
```

Description

Removes a single scene as identified by group and scene ID.

Prototype

```
zclStatus_t ASL_ZclRemoveAllScenesReq(zclSceneRemoveAllScenesReq_t *pReq);
```

Description

Remove all scenes eliminates all scenes on specified group.

Prototype

```
zbStatus_t ASL_ZclStoreSceneReq(zclSceneStoreSceneReq_t *pReq);
```

Description

Stores a scene in the remote node with the current settings. This command can be used with a variety of devices. For example, the lights may be on and the shades down for a “night-time” scene. The lights off and the shades up for a “day-time” scene. A scene is just the state of the device stored for later retrieval.

For example, set the HaDimmableLight to 33% brightness (1 bar on the LEDs). Then issue an ASL_ZclStoreSceneReq() call to that node from another node. That scene can later be recalled by another device.

The HaDimmerSwitch application (a BeeKit template) uses SW4 and LSW4 (long switch 4) to do this store scene and recall scene behavior.

Prototype

```
zbStatus_t ASL_ZclRecallSceneReq(zclSceneRecallSceneReq_t *pReq);
```

Description

Recalls a previously added or stored scene in a remote node.

Prototype

```
zclStatus_t ASL_ZclGetSceneMembershipReq
(
zclSceneGetSceneMembershipReq_t *pReq
);
```

Description

Retrieves the scene membership for a given group, a list of scene IDs in the remote node that belong to this group. The membership structure returned looks like

```
typedef struct zclCmdScene_GetSceneMembershipRsp_tag
{
  zclStatus_t   status;          /* status */
  haCapacity_t  capacity;        /* # of scenes available (capacity) */
  zbGroupId_t   aGroupId;        /* group */
  uint8_t       sceneCount;      /* scenes that belong to the group */
  zclSceneId_t  scenes[1];       /* list of scenes IDs */
} zclCmdScene_GetSceneMembershipRsp_t;
```

Prototype

```
zbStatus_t ASL_ZclOnOffReq(zclOnOffReq_t *pReq);
```

Description

Turns a remote device on or off, for example an on/off light.

Prototype

```
zbStatus_t ASL_ZclLevelControlReq
(
zclLevelControlReq_t *pReq,
uint8_t command
);
```

Description

Sends a command that changes the level (for example the light level on a dimmable light) of a remote device.

Level control commands must be one of

- gZclCmdLevelControl_MoveToLevel_c
- gZclCmdLevelControl_Move_c
- gZclCmdLevelControl_Step_c
- gZclCmdLevelControl_Stop_c
- gZclCmdLevelControl_MoveToLevelOnOff_c
- gZclCmdLevelControl_MoveOnOff_c
- gZclCmdLevelControl_StepOnOff_c
- gZclCmdLevelControl_StopOnOff_c

2.1.6 Smart Energy (SE) Cluster Description

This section covers the Smart Energy Zigbee Cluster Library (ZCL) functions and data type available to the application.

The SE functional domain includes the following ZCL clusters:

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

The Key Establishment cluster is added to the General functional domain. All SE clusters are implemented in SeProfile.h, ZclSe.h, ZclSe.c and are explained in the following sections.

All the SE requests follow the same format as it shows in the section 2.1.5 ZCL Requests.

A code-snippet example of Inter-Pan and In-Pan requests definition type would be as follows:

```
/* Cancel All Load Control Events command payload */
typedef struct zclCmdDmndRspLdCtrl_CancelAllLdCtrlEvtsReq_tag
{
  uint8_t           CancelCtrl;
} zclCmdDmndRspLdCtrl_CancelAllLdCtrlEvtsReq_t;

/* Cancel All Load Control Events command request */
typedef struct zclDmndRspLdCtrl_CancelAllLdCtrlEvtsReq_tag
{
  afAddrInfo_t addrInfo;
  uint8_t zclTransactionId;
  zclCmdDmndRspLdCtrl_CancelAllLdCtrlEvtsReq_t cmdFrame;
} zclDmndRspLdCtrl_CancelAllLdCtrlEvtsReq_t;

typedef struct zclCmdMsg_DisplayMsgReq_tag
{
  MsgId_t       MsgID;
  uint8_t       MsgCtrl;
  ZCLTime_t     StartTime;
  Duration_t    DurationInMinutes;
  zclStr32_t    Msg;
} zclCmdMsg_DisplayMsgReq_t;
```

```
typedef struct  zclInterPanDisplayMsgReq_tag
{
  InterPanAddrInfo_t addrInfo;
  uint8_t zclTransactionId;
  zclCmdMsg_DisplayMsgReq_t cmdFrame;
} zclInterPanDisplayMsgReq_t;

typedef struct InterPanAddrInfo_tag
{
  zbAddrMode_t  srcAddrMode;    /* indirect, group, direct-16, direct-64 */
  zbAddrMode_t  dstAddrMode;    /* indirect, group, direct-16, direct-64 */
  zbPanId_t     dstPanId;
  zbApsAddr_t   dstAddr;        /* short address, long address or group (ignored on indirect
mode)*/
  zbProfileId_t aProfileId;     /* application profile (either private or public) */
  zbClusterId_t aClusterId;     /* cluster identifier */
} InterPanAddrInfo_t;
```

All devices, attributes, clusters IDs, macro definitions and data type definitions for all SE requests are in the `SeProfile.h` file.

The following code example shows how to form and send (over the air) a Get Current Price request. All Cluster requests are formed and sent in the same manner. They call the specific functions that are covered in the following sections.

```
void BeeAppGetCurrentPrice(void)
{
  zclPrice_GetCurrPriceReq_t req;

  req.addrInfo.dstAddrMode = 0x02; /*short address*/
  Copy2Bytes(req.addrInfo.dstAddr.aNwkAddr, ESPAddr);
  req.addrInfo.dstEndPoint = gSendingNwkData.endPoint;
  Set2Bytes(req.addrInfo.aClusterId, gZclClusterPrice_c);
  req.addrInfo.srcEndPoint = appEndPoint;
  req.addrInfo.txOptions = afTxOptionsDefault_c;
  req.addrInfo.radiusCounter = afDefaultRadius_c;
  req.zclTransactionId = gZclTransactionId++;
  // receive price updates from ESP
  req.cmdFrame.CmdOptions = gGetCurrPrice_RequestorRxOnWhenIdle_c;

  (void)zclPrice_GetCurrPriceReq (&req);
}
```

2.1.6.1 Demand Response and Load Control Cluster

The Demand Response and Load Control Cluster handles the Client and Server Load Control events.

- The Server Load Control Cluster passes the received load control events from the utility company to the devices from Zigbee Smart Energy network that supports load control.
- The Client Load Control Cluster implements the rules and guidelines for overlapping events and signals the application when a load control event occurs. For this, the application function

BeeAppUpdateDevice() is called with the gZclUI_LdCtrlEvt_c application event (specifies that a load control event occurs) and the corresponding information fields of the load control event.

The Client Load Control Cluster maintains a table of load control events with the following fields:

```
typedef struct zclLdCtrl_EventsTableEntry_tag
{
  afAddrInfo_t addrInfo;    /*where to send the Reports Event Status for this event */
  zclCmdDmndRspLdCtrl_LdCtrlEvtReq_t cmdFrame; /* keep the Event */
  uint8_t         CurrentStatus;     /* the current event status; can be modify by the user */
  uint8_t           NextStatus;         /* keep track of next status which depend of current
status (almost all the time ) */
  uint8_t           IsSuccessiveEvent;  /* TRUE - is a "back to back" event; else FAlSE */
  ZCLTime_t         CancelTime; /* When the event should be canceled; 0Xffffffff meens event
not canceled; apply the random time stop if needed */
  uint8_t        CancelCtrl;  /* to check if the cancelling on CancelTime use the randomization
information from cmdFrame */
  LCDRDevCls_t      CancelClass; /* 0xffff meens not used */
  LCDRDevCls_t      NotSupersededClass; /* used for supressing events; 0xffff - not used yet;
0x0000 - event is seperseded */
  ZCLTime_t         SupersededTime;
  uint8_t           EntryStatus; /* the tabel entry is in use (0xFF) or not (0x00) */
}zclLdCtrl_EventsTableEntry_t;
```

Every time the “CurrentStatus” field of an entry from Load Control Table is changing, the function BeeAppUpdateDevice() is called with the following parameters: source end point, application event, attribute ID if need, cluster ID and the Load Control entry.

Here is a code example of how the application is signaled by the Load Control Cluster:

```
BeeAppUpdateDevice(pAddrInfo->srcEndPoint, gZclUI_LdCtrlEvt_c, 0, pAddrInfo->aClusterId,
&gaEventsTable[mReportStatusEntryIndex]);
```

The application should check the “CurrentStatus” field to handle the load control event every time the function BeeAppUpdateDevice() is called. The following Status of the load control events can occur:

```
/* Event Status Field Values */
#define gSELCDR_LdCtrlEvtCode_CmdRcvd_c                    0x01
#define gSELCDR_LdCtrlEvtCode_Started_c                    0x02
#define gSELCDR_LdCtrlEvtCode_Completed_c                  0x03
#define gSELCDR_LdCtrlEvtCode_UserHaveToChooseOptOut_c     0x04
#define gSELCDR_LdCtrlEvtCode_UserHaveToChooseOptIn_c      0x05
#define gSELCDR_LdCtrlEvtCode_EvtCancelled_c               0x06
#define gSELCDR_LdCtrlEvtCode_EvtSuperseded_c               0x07
#define gSELCDR_LdCtrlEvtCode_EvtPrtlCompletedWithUserOptOut_c   0x08
#define gSELCDR_LdCtrlEvtCode_EvtPrtlCompletedWithUserOptIn_c    0x09
#define gSELCDR_LdCtrlEvtCode_EvtCompletedWithNoUser_c           0x0A
#define gSELCDR_LdCtrlEvtCode_EvtInvalidCancelCmdDefault_c       0xF8
#define gSELCDR_LdCtrlEvtCode_EvtInvalidEffectiveTime_c          0xF9
#define gSELCDR_LdCtrlEvtCode_EvtReserved_c                      0xFA
#define gSELCDR_LdCtrlEvtCode_EvtHadExpired_c                    0xFB
#define gSELCDR_LdCtrlEvtCode_EvtUndef_c                         0xFD
#define gSELCDR_LdCtrlEvtCode_LdCtrlEvtCmdRjctd_c                0xFE
```

A code-snippet example of how the application should handle the load control events would be:

```
void BeeAppUpdateDevice
  (
```

```
  zbEndPoint_t endPoint,    /* IN: endpoint update happend on */
  zclUIEvent_t event,        /* IN: state to update */
  zclAttrId_t attrId,
  zbClusterId_t clusterId,
  void *pData
  )
{
 (void) attrId;
 (void) clusterId;
 (void) pData;
  /* no application specific events, let ASL handle all of them */

  switch(event)
  {
  case gZclUI_LdCtrlEvt_c:
         BeeAppHandleLdCtrlEvt(pData);
         break;
  }
}


void BeeAppHandleLdCtrlEvt(void *pData)
{
  zclLdCtrl_EventsTableEntry_t *pEvtData;
  zclCmdDmndRspLdCtrl_LdCtrlEvtReq_t *pEvt;
  /* get the entry data */
  pEvtData = (zclLdCtrl_EventsTableEntry_t *)pData;
  /* get the load control event */
  pEvt = &pEvtData->cmdFrame;
  /* user should process the load control event */
  (void)pEvt;
  switch(pEvtData->CurrentStatus)
  {
  case gSELCDR_LdCtrlEvtCode_CmdRcvd_c:
    break;
  case gSELCDR_LdCtrlEvtCode_Started_c:
    break;
  case gSELCDR_LdCtrlEvtCode_UserHaveToChooseOptOut_c:
    break;
  case gSELCDR_LdCtrlEvtCode_UserHaveToChooseOptIn_c:
    break;
  case gSELCDR_LdCtrlEvtCode_EvtCancelled_c:
    break;
  case gSELCDR_LdCtrlEvtCode_EvtSuperseded_c:
    break;
  case gSELCDR_LdCtrlEvtCode_Completed_c:
    break;
  case gSELCDR_LdCtrlEvtCode_EvtPrtlCompletedWithUserOptOut_c:
    break;
  case gSELCDR_LdCtrlEvtCode_EvtPrtlCompletedWithUserOptIn_c:
    break;
  case gSELCDR_LdCtrlEvtCode_EvtCompletedWithNoUser_c:
    break;
  default:
    break;

  }
```

```
}
```

The following section describes the individual request and process functions, use and prototypes.

Prototype

```
void ZCL_LdCtrlClientInit(void);
```

Description

This function initializes the Client Load Control Cluster. This function should be called in the Init function (BeeAppInit()) if the device support load control.

Prototype

```
uint8_t FindEventInEventsTable(uint8_t *pEvtId);
```

Description

This function finds an existing event in the Load Control Table Events (for Client Load Control).

Prototype

```
extern zbStatus_t ZCL_SendReportEvtStatus
(
afAddrInfo_t *pAddrInfo,
zclCmdDmndRspLdCtrl_LdCtrlEvtReq_t *pMsg,
uint8_t eventStatus,
bool_t invalidValueFlag /* if TRUE sent RES with invalid values for fields */
);
```

Description

This function sends over the air (to the ESP) a status report of a Load Control event; when an event is changing its “Current Status”, the Status is signaled to the application and also send to the ESP (from the Client to Server).

Prototype

```
void ZCL_AcceptVoluntaryLdCrtlEvt(bool_t flag);
```

Description

This function accepts (TRUE or FALSE) voluntary load control events and is specific for the Client Load Control Cluster.

Prototype

```
uint8_t ZCL_SetOptStatusOnEvent(uint8_t *pEvtId, uint8_t optStatus);
```

Description

Sends an “Opt” Status for a specify Load Control Event ID.

Prototype

```
zbStatus_t ZCL_DmndRspLdCtrlClusterServer(zbApsdeDataIndication_t *pIndication, afDeviceDef_t
*pDev);
```

Description

It is empty function that is called each time a request is received from the Client (the application should handle the Client requests in this function).

Prototype

```
zbStatus_t ZCL_DmndRspLdCtrlClusterClient(zbApsdeDataIndication_t *pIndication, afDeviceDef_t
*pDev);
```

Description

This function is processing the requests received from the ESP (requests like “Load Control Event request”, “Cancel Load Control Event request” etc.). If the device support load control commands should have this function included in the table that defines the list of clusters and functions handlers as follow:

```
/* list of clusters and function handlers */
afClusterDef_t const gaSEPCTClusterList[] =
{
  { { gaZclClusterBasic_c    }, (pfnIndication_t)ZCL_BasicClusterServer, NULL, (void
*)(&gZclBasicClusterAttrDefList) },
  { { gaZclClusterIdentify_c }, (pfnIndication_t)ZCL_IdentifyClusterServer, NULL, (void
*)(&gZclIdentifyClusterAttrDefList) },
  { { gaKeyEstabCluster_c}, (pfnIndication_t)ZCL_KeyEstabClusterServer,(pfnIndication_t)
ZCL_KeyEstabClusterClient, (void *)(&gZclKeyEstabServerAttrDefList) },
  { { gaZclClusterDmndRspLdCtrl_c}, NULL, (pfnIndication_t)ZCL_DmndRspLdCtrlClusterClient,
(void *)(&gZclDRLCClientServerClusterAttrDefList)},
  { { gaZclClusterTime_c}, NULL, NULL, (void *)(&gZcltimeServerClusterAttrDefList)}
};
```

Prototype

```
zbStatus_t ZclDmndRspLdCtrl_ReportEvtStatus(zclDmndRspLdCtrl_ReportEvtStatus_t *pReq);
```

Description

This function sends over the air (to the ESP) a status report of a Load Control event; when an event is changing its “Current Status”, the Status is signaled to the application and also is sent over the air to the ESP (from Client to Server).

Prototype

```
zbStatus_t ZclDmndRspLdCtrl_LdCtrlEvtReq(zclDmndRspLdCtrl_LdCtrlEvtReq_t *pReq);
```

Description

This function sends over the air a Load Control Event request; the utility company can send Load Control requests via the ESP to devices that support load control (from Server to Client).

Prototype

```
zbStatus_t ZclDmndRspLdCtrl_CancelLdCtrlEvtReq (zclDmndRspLdCtrl_CancelLdCtrlEvtReq_t *pReq);
```

Description

This function sends over the air a Cancel Load Control Event request; the utility company can send Cancel Load Control requests via the ESP to devices that support load control (from Server to Client).

Prototype

```
zbStatus_t ZclDmndRspLdCtrl_CancelAllLdCtrlEvtReq (zclDmndRspLdCtrl_CancelAllLdCtrlEvtsReq_t
*pReq);
```

Description

This function sends over the air a Cancel All Load Control Event request; the utility company can send Cancel All Load Control requests via the ESP to devices that support load control (from Server to Client).

Prototype

```
zbStatus_t ZclDmndRspLdCtrl_GetScheduledEvtsReq (zclDmndRspLdCtrl_GetScheduledEvts_t *pReq);
```

Description

This function sends over the air a Get Scheduled Events request; a device that supports load control can get the scheduled events from utility company through the ESP (from Client to Server).

2.1.6.2 Price Cluster

The Price Cluster provides the API functions to communicate the price information to Zigbee Smart Energy or non-Smart Energy devices. This price information is distributed from the utility company to the devices via the ESP.

The Server Price Cluster just stores and updates the received price information events from the utility company (the Server Price Cluster is typically implemented by the ESP). If a price update occurs, that price is sent to all devices that support price updates. A device supports price updates if it has once before sent a “Get Current Price request” to signal to the ESP that it has the Rx On when Idle. A device that supports the Server Price Cluster has the capability to respond at the Client requests (“Get Current Price request” and “Get Scheduled Prices request”).

The Client Price Cluster is implemented by the devices that have the capability to receive and handle the price information. This devices could be in a a Zigbee Smart Energy or non-Smart Energy network (Home Automation Profile) and maintain a table of price events with the following fields:

```
typedef struct publishPriceEntry_tag
{
  zclCmdPrice_PublishPriceRsp_t Price;
  ZCLTime_t EffectiveStartTime;
  uint8_t EntryStatus;
} publishPriceEntry_t;

typedef struct zclCmdPrice_PublishPriceRsp_tag
{
  ProviderID_t ProviderID;
  zclStr12_t   RateLabel;
  SEEvtId_t    IssuerEvt;
```

```
  ZCLTime_t    CurrTime;
  uint8_t      UnitOfMeasure;
  Currency_t   Currency;
  uint8_t      PriceTrailingDigitAndPriceTier;
  uint8_t      NumOfPriceTiersAndRgstrTier;
  ZCLTime_t    StartTime;
  Duration_t   DurationInMinutes;
  Price_t      Price;
  uint8_t      PriceRatio;  /* O */
  Price_t      GenerationPrice;   /* O */
  uint8_t      GenerationPriceRatio;  /* O */
} zclCmdPrice_PublishPriceRsp_t;
```

The Client Price Cluster signals the application when “EntryStatus“ field of a price entry is changing. For this, the function BeeAppUpdateDevice() is called, passing to the application the corresponding entry from Price table and issuing the application event gZclUI_PriceEvt_c. The following status of the price event can occur:

```
#define gPriceReceivedStatus_c 0x01
#define gPriceStartedStatus_c 0x02
#define gPriceUpdateStatus_c 0x03
#define gPriceCompletedStatus_c 0x04
```

The following code example shows how to handle the price events in the application:

```
void BeeAppUpdateDevice
  (
  zbEndPoint_t endPoint,    /* IN: endpoint update happend on */
  zclUIEvent_t event,        /* IN: state to update */
  zclAttrId_t attrId,
  zbClusterId_t clusterId,
  void *pData
  )
{
 (void) attrId;
 (void) clusterId;
 (void) pData;
  /* no application specific events, let ASL handle all of them */

  switch(event)
  {
    case gZclUI_PriceEvt_c:
         BeeAppHandlePrice(pData);
         break;

  }
}

void BeeAppHandlePrice(void *pData)
{
  publishPriceEntry_t *pPriceData;

  pPriceData = (publishPriceEntry_t *)pData;
  /* Print Price Information */
#if gTargetMC1322xNCB
  ASL_PrintPrice(pPriceData);
#endif
  switch(pPriceData->EntryStatus)
```

```
{
case gPriceReceivedStatus_c:
  break;
case gPriceStartedStatus_c:
  FLib_MemCpy(&currentPrice, pData, sizeof(publishPriceEntry_t));
  break;
case gPriceUpdateStatus_c:
  break;
case gPriceCompletedStatus_c:
  currentPrice.EntryStatus = 0xff;//current price expired (not available)
  BeeAppGetCurrentPrice();//get new price from ESP
  break;
default:
  break;
}
```

The following section describes the individual request and process functions, use and prototypes.

Prototype

```
void ZCL_PriceClientInit(void);
```

Description

This function initializes the Client Price Cluster. This function have to be called in the Init function (BeeAppInit()) if the device support Client Price Cluster.

Prototype

```
zbStatus_t ZCL_ScheduleServerPriceEvents(zclCmdPrice_PublishPriceRsp_t *pMsg);
```

Description

This function is specific to the Server Price Cluster and stores and schedules the received price. The Server Price Cluster doesn't keep track of the price status, only stores the received prices and take care that nested and overlapping Publish Price commands not to occur. This function is called when prices are received from the HOST (Test Tool Software) to be stores in the Server Price Cluster.

Prototype

```
zbStatus_t ZCL_UpdateServerPriceEvents(zclCmdPrice_PublishPriceRsp_t *pMsg);
```

Description

This function is specific to the Server Price Cluster and updates a store price from the Provider. When new pricing information is provided that replaces older pricing information for the same time period, “IssuerEvt” field allows devices to determine which information is newer. It is expected that the value contained in this field is a unique number managed by upstream servers. Thus, newer pricing information will have a value in the Issuer Event ID field that is larger than older pricing information.

Prototype

```
void ZCL_DeleteServerScheduledPrices(void);
```

Description

This function deletes all the stores prices from the Server Price Cluster.

Prototype

```
zbStatus_t ZCL_PriceClusterServer(zbApsdeDataIndication_t *pIndication, afDeviceDef_t *pDev);
```

Description

This function processes the price requests received from a Smart Energy Client device that support price (process Smart Energy requests like “Get Current Price request” or “Get scheduled Price request”).

Prototype

```
zbStatus_t ZCL_PriceClusterClient(zbApsdeDataIndication_t *pIndication, afDeviceDef_t *pDev );
```

Description

This function processes the publish prices received from a Server device (ESP) that support prices (process Publish Price response).

Prototype

```
zbStatus_t ZCL_InterPanPriceClusterServer (zbApsdeDataIndication_t *pIndication, afDeviceDef_t
*pDev);
```

Description

This function processes the Inter-Pan Price requests received from a Client device that support price (processes requests like “Inter-Pan Get Current Price request” or “Inter-Pan Get scheduled Price request”).

Prototype

```
zbStatus_t ZCL_InterPanPriceClusterClient(zbApsdeDataIndication_t *pIndication,afDeviceDef_t
*pDev );
```

Description

This function processes the processes Inter-Pan Publish Price response received from a Server device (ESP) that support price.

Prototype

```
zbStatus_t zclPrice_GetCurrPriceReq(zclPrice_GetCurrPriceReq_t *pReq);
```

Description

This function sends over the air a “Get Current Price request”. This request is sent by a device that supports the Client Price Cluster to the ESP (the ESP have the prices stored from the utility company).

Prototype

```
zbStatus_t zclPrice_GetScheduledPricesReq(zclPrice_GetScheduledPricesReq_t *pReq);
```

Description

This function sends over the air a “Get Scheduled Price request”. This request is sent by a device that supports the Client Price Cluster to the ESP (the ESP have the prices stored from the utility company).

Prototype

```
zbStatus_t zclPrice_PublishPriceRsp(zclPrice_PublishPriceRsp_t *pReq);
```

Description

This function sends over the air a “Publish Price response”. This response is sent when an update price is made by the ESP (Server Price Cluster) or as a response to a price request made by a device that supports the Client Price Cluster.

Prototypes

```
zbStatus_t zclPrice_InterPanGetCurrPriceReq(zclPrice_InterPanGetCurrPriceReq_t *pReq);
zclPrice_t zclPrice_InterPanGetScheduledPricesReq(zclPrice_InterPanGetScheduledPricesReq_t
*pReq);
zbStatus_t zclPrice_InterPanPublishPriceRsp(zclPrice_InterPanPublishPriceRsp_t *pReq);
```

Description

These functions have the same functionality as the requests functions described before; only the communication is between devices from different PANs (Inter-Pan communication).

If the application uses the Inter-Pan communication, the Init function (BeeAppInit()) have to register the function that process the Inter-Pan messages as the following code example:

```
#if gInterPanCommunicationEnabled_c
ZCL_RegisterInterPanClient((pfnInterPanIndication_t)ZCL_InterPanClusterClient);
#endif
```

2.1.6.3 Messaging Cluster

This cluster provides API functions for passing text messages between ZigBee devices. The utility company delivers text messages to ZigBee devices that implement the Client Messaging Cluster via the ESP that implements the Server Messaging Cluster.

The Server Messaging Cluster (ESP) just passes the text messages received from HOST (Test Tool Software) to the devices from the network. Also the devices that support Server Messaging Cluster have the capability to handle the received requests like “Get Last Message request” or “Message Confirmation request”.

The Client Messaging Cluster process the requests received from the ESP (requests like “Display Message request”, “Cancel Message request”).

When a text message is received by a device that has implemented the Client Messaging Custer, the application is signaled calling BeeAppUpdateDevice() function. The application should check in this function the received application events (gZclUI_MsgDisplayMessageReceived_c or gZclUI_MsgCancelMessageReceived_c) and process the received message.

The following section describes the individual request and process functions, use and prototypes.

Prototype

```
void ZCL_MsgInit(void);
```

Description

This function have to be called in the Init function (BeeAppInit()) to initialize the messaging cluster.

Prototype

```
msgAddrInfo_t *ZCL_GetMsgSourceaddrForResponse(void);
```

Description

This function return the source address information of the device that sent the message so that to confirm or respond to the sender (msgAddrInfo_t structure provides information that shows if the message was sent from a different PAN or from the same PAN etc.).

Prototype

```
zbStatus_t ZCL_MsgClusterClient(zbApsdeDataIndication_t *pIndication, afDeviceDef_t *pDev);
```

Description

This function processes the requests sent from a Server device (ESP) that support messaging (process “Display Message request”, “Cancel Message request”).

Prototype

```
zbStatus_t ZCL_MsgClusterServer(zbApsdeDataIndication_t *pIndication, afDeviceDef_t *pDev);
```

Description

This function processes the requests sent from a Client device that support messaging (process the following requests: “Get Last Message request” and “Message Confirmation request”).

Prototype

```
zbStatus_t ZCL_InterPanMsgClusterClient(zbInterPanDataIndication_t *pIndication, afDeviceDef_t
*pDev);
zbStatus_t ZCL_InterPanMsgClusterServer(zbInterPanDataIndication_t *pIndication, afDeviceDef_t
*pDev);
```

Description

These functions have the same functionality as the Client and Server message functions described before; only the communication is between devices from different PANs (Inter-Pan communication).

Prototype

```
zbStatus_t ZclMsg_DisplayMsgReq(zclDisplayMsgReq_t *pReq);
```

Description

This function sends over the air a “Display Message request” (from Server (ESP) to Client).

Prototype

```
zbStatus_t ZclMsg_CancelMsgReq(zclCancelMsgReq_t *pReq);
```

Description

This function sends over the air a “Cancel Message request” to cancel a displayed message (from Server (ESP) to Client).

Prototype

```
zbStatus_t ZclMsg_GetLastMsgReq(zclGetLastMsgReq_t *pReq);
```

Description

This function sends over the air a “Get Last Message request” (from Client To Server).

Prototype

```
zbStatus_t ZclMsg_MsgConf(zclMsgConfReq_t *pReq);
```

Description

This function sends over the air a “Message Confirmation request” (from Client to Server).

Prototype

```
zbStatus_t ZclMsg_InterPanDisplayMsgReq(zclInterPanDisplayMsgReq_t *pReq);
zbStatus_t ZclMsg_InterPanCancelMsgReq(zclInterPanCancelMsgReq_t *pReq);
zbStatus_t ZclMsg_InterPanGetLastMsgReq(zclInterPanGetLastMsgReq_t *pReq);
zbStatus_t ZclMsg_InterPanMsgConf(zclInterPanMsgConfReq_t *pReq);
```

Description

These functions have the same functionality as the requests functions described before; only the communication is between devices from different PANs (Inter-Pan communication).

2.1.6.4 Key Establishment Cluster

This cluster provides attributes and commands to perform mutual authentication and establish keys between two ZigBee devices.

NOTE

The Key Establishment Cluster requires that an Elliptic Curve Cryptography (ECC) Library and certificate is acquired.

By default, the BeeStack codebase is NOT delivered with a working ECC library. A working ECC library must be acquired and placed in the following project folder:

```
\SolutionName\ProjectName\BeeApps\se
```

The key establishment is NOT performed if this property is FALSE, and the trust center link key will be used if available. A valid certificate must be added to the `seprofile.c` file.

Figure 2-1 shows a successful key establishment process which is handled by the cluster client and server handler.

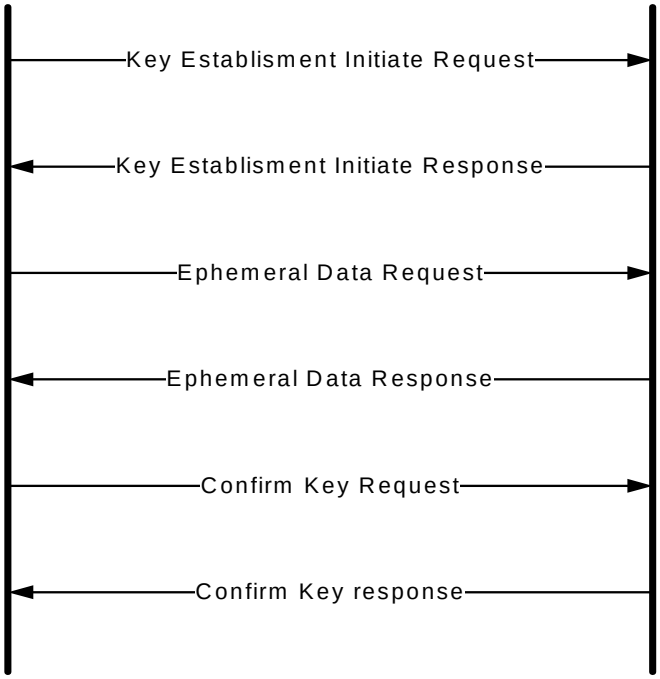

Figure 2-1. Key Establishment Process

The Key Establishment cluster has the following properties that can be configured from BeeKit:

- gEccIncluded_d — Enable support for ECC library
- KeyEstab_DefaultWaitTime_c — The wait time sent in an Terminate Key Establishment request
- gKeyEstab_ConfirmKeyMessageTimeout_c — The Confirm key Time out
- gKeyEstab_EphemeralDataMessageTimeout_c — The Ephemeral Data Message time-out

The following sections describes the individual request and process functions, use and prototypes.

Prototype

```
bool_t ZCL_InitiateKeyEstab(zbEndPoint_t DstEndpoint,zbEndPoint_t SrcEndpoint, zbNwkAddr_t
DstAddr)
```

Description

The function initiates an Key establishment process. the Destination address and endpoint of an device (typically the ESP) must be supplied.

Prototype

```
zbStatus_t ZCL_KeyEstabClusterServer(zbApsdeDataIndication_t *pIndication, afDeviceDef_t
*pDev)
zbStatus_t ZCL_KeyEstabClusterClient (zbApsdeDataIndication_t *pIndication, afDeviceDef_t
*pDev)
```

Description

These two functions contain the Key Establishment client and server function handlers. Both contain a state for handling the client or server state machine of the key establishment exchange.

2.2 ZCL Configurable Properties

The following table lists the configurable ZigBee Cluster Library properties (also called compile-time options). These properties can be found in the `ZclOptions.h` file and are configurable through BeeKit. Disabling some of the requests can make code size smaller.

Table 2-1. ZCL Configurable Properties

Option	Description
gHaMaxScenes_c	Set number of scenes. Default to 2
gHaMaxSceneSize_c	Sets the maximum size in bytes for the storable scene. OnOff light scene require needs 1 byte, Dimmer light scene requires 11 bytes, Thermostat scene requires 45 bytes.
gZclClusterOptionals_d	Enables optional clusters and attributes, as defined by the Home Automation and ZigBee Cluster Library specifications. Defaults to FALSE.
gZclEnableReporting_c	Enable attribute reporting. Attribute read/write are always enabled. Defaults to FALSE.
gZclEnableOver32BitAttrsReporting_c	Enable/disable long attribute types reporting
gZclEnableLongStringTypes_c	Enable/disable long string types handling
gZclDiscoverAttrReq_d	Enable/disable DiscoverAttrReq
gZclDiscoverAttrRsp_d	Enable/disable DiscoverAttrRsp
gASL_ZclIdentifyQueryReq_d	Enable/disable IdentifyQueryReq
gASL_ZclIdentifyReq_d	Enable/disable IdentifyReq
gASL_ZclGroupAddGroupReq_d	Enable/disable GroupAddGroupReq
gASL_ZclGroupViewGroupReq_d	Enable/disable GroupViewGroupReq
gASL_ZclGetGroupMembershipReq_d	Enable/disable GetGroupMembershipReq
gASL_ZclRemoveGroupReq_d	Enable/disable RemoveGroupReq
gASL_ZclRemoveAllGroupsReq_d	Enable/disable RemoveAllGroupsReq
gASL_ZclStoreSceneReq_d	Enable/disable StoreSceneReq
gASL_ZclRecallSceneReq_d	Enable/disable RecallSceneReq
gASL_ZclGetSceneMembershipReq_d	Enable/disable GetSceneMembershipReq
gASL_ZclOnOffReq_d	Enable/disable OnOffReq
gASL_ZclLevelControlReq_d	Enable/disable LevelControlReq
gASL_ZclSceneAddSceneReq_d	Enable/disable SceneAddSceneReq
gASL_ZclViewSceneReq_d	Enable/disable ViewSceneReq
gASL_ZclRemoveSceneReq_d	Enable/disable RemoveSceneReq
gASL_ZclRemoveAllScenesReq_d	Enable/disable RemoveAllScenesReq
gASL_ZclCommissioningRestartDeviceRequest_d	Enable/disable CommissioningRestartDeviceRequest

Table 2-1. ZCL Configurable Properties (continued)

Option	Description
gASL_ZclCommissioningSaveStartupParametersRequest_d	Enable/disable CommissioningSaveStartupParametersRequest
gASL_ZclCommissioningRestoreStartupParametersRequest_d	Enable/disable CommissioningRestoreStartupParametersRequest
gASL_ZclCommissioningResetStartupParametersRequest_d	Enable/disable CommissioningResetStartupParametersRequest
gASL_ZclDisplayMsgReq_d	Enable/disable DisplayMsgReq
gASL_ZclCancelMsgReq_d	Enable/disable CancelMsgReq
gASL_ZclMsgConfReq_d	Enable/disable MsgConfReq
gASL_ZclGetLastMsgReq_d	Enable/disable GetLastMsgReq
gASL_ZclInterPanDisplayMsgReq_d	Enable/disable InterPanDisplayMsgReq
gASL_ZclInterPanCancelMsgReq_d	Enable/disable InterPanCancelMsgReq
gASL_ZclInterPanMsgConfReq_d	Enable/disable InterPanMsgConfReq
gASL_ZclInterPanGetLastMsgReq_d	Enable/disable InterPanGetLastMsgReq
gASL_ZclRemoveSceneReq_d	Enable/disable RemoveSceneReq
gASL_ZclSmplMet_GetProfReq_d	Enable/disable GetProfReq
gASL_ZclSmplMet_GetProfRsp_d	Enable/disable GetProfRsp
gASL_ZclDmndRspLdCtrl_ReportEvtStatus_d	Enable/disable ReportEvtStatus
gASL_ZclDmndRspLdCtrl_LdCtrlEvtReq_d	Enable/disable LdCtrlEvtReq
gASL_ZclDmndRspLdCtrl_CancelLdCtrlEvtReq_d	Enable/disable CancelLdCtrlEvtReq
gASL_ZclDmndRspLdCtrl_CancelAllLdCtrlEvtsReq_d	Enable/disable CancelAllLdCtrlEvtsReq
gASL_ZclPrice_GetCurrPriceReq_d	Enable/disable GetCurrPriceReq
gASL_ZclPrice_GetSheduledPricesReq_d	Enable/disable GetSheduledPricesReq
gASL_ZclPrice_PublishPriceRsp_d	Enable/disable PublishPriceRsp
gASL_ZclPrice_InterPanGetCurrPriceReq_d	Enable/disable InterPanGetCurrPriceReq
gASL_ZclPrice_InterPanGetSheduledPricesReq_d	Enable/disable InterPanGetSheduledPricesReq
gASL_ZclPrice_InterPanPublishPriceRsp_d	Enable/disable InterPanPublishPriceRsp
gASL_ZclInterPanGetLastMsgReq_d	Enable/disable InterPanGetLastMsgReq
gASL_ZclKeyEstab_InitKeyEstabReq_d	Enable/disable InitKeyEstabReq

Table 2-1. ZCL Configurable Properties (continued)

Option	Description
gASL_ZclKeyEstab_EphemeralDataReq_d	Enable/disable EphemeralDataReq
gASL_ZclKeyEstab_ConfirmKeyDataReq_d	Enable/disable ConfirmKeyDataReq
gASL_ZclKeyEstab_TerminateKeyEstabServer_d	Enable/disable TerminateKeyEstabServer
gASL_ZclKeyEstab_InitKeyEstabRsp_d	Enable/disable InitKeyEstabRsp
gASL_ZclKeyEstab_EphemeralDataRsp_d	Enable/disable EphemeralDataRsp
gASL_ZclKeyEstab_ConfirmKeyDataRsp_d	Enable/disable ConfirmKeyDataRsp
gASL_ZclKeyEstab_TerminateKeyEstabClient_d	Enable/disable TerminateKeyEstabClient
gASL_ZclSE_RegisterDevice_d	Enable/disableRegisterDevice
gASL_ZclSE_DeRegisterDevice_d	Enable/disable DeRegisterDevice
gASL_ZclSE_InitTime_d	Enable/disable InitTime

Chapter 3
Adding Custom Devices, Clusters and Attributes

The BeeStack ZigBee Cluster Library implementation is largely table driven. Only that code which is used, as described by the device, cluster and attribute definitions is included in any given application. Key points are

- Endpoints include device definitions
- Device definitions include cluster definitions
- Cluster definitions include attribute definitions
- The application endpoint code must have special handlers for adding, viewing and recalling scenes.

3.1 Device Definitions

The device definitions are referred to in the `EndpointConfig.c` file, but are actually defined in an application specific endpoint files, such as `HaOnOffLightEndPoint.c`.

Device definitions include a pointer to an instance structure that defines the RAM data per instance of the device. For example, if a node contained three separately controllable lights, there would be three device definitions pointing to three separate instances of the light's data. Cluster and attribute definitions are the same for all instances.

The device definition includes the following fields:

```
typedef struct afDeviceDef_tag
{
  pfnIndication_t     pfnZCL;        /* ptr to handler to bring in code */
  uint8_t             clusterCount;  /* # of clusters in the list */
  afClusterDef_t      *pClusterDef;  /* a list of clusters */
  uint8_t             reportCount;   /* # of items in the report */
  void                *pReportList;  /* a list of reportable attributes */
  void                *pData;        /* instance data */
} afDeviceDef_t;
```

The device definition includes a count of clusters and the cluster definitions, a list of reportable attributes (if any), and a pointer to the instance data.

NOTE

ZigBee does not specify device or cluster definition fields. This is an implementation specific detail to implement the specified over-the-air behavior of the ZigBee Cluster Library.

3.2 Cluster Definitions

Cluster definitions define a particular cluster and lead to its attribute list. The cluster definition looks as follows:

```
typedef struct afClusterDef_tag
{
  zbClusterId_t    aClusterId;    /* cluster ID */
  pfnIndication_t  pfnServerIndication; /* server indication handler for this cluster */
  pfnIndication_t  pfnClientIndication; /* client indication handler for this cluster */
  void            *pAttrList;     /* ptr to attribute list (zclAttrDefList_t) */
  uint8_t          dataOffset;    /* offset of this cluster's data within instance data */
} afClusterDef_t;
```

The cluster ID is specified by the ZigBee Cluster Library and is always stored little-endian, the same as it is over-the-air (that is, cluster 0x0001 would be stored as 0x01,0x00).

NOTE

The HCS08 is big-endian.

The indication function lists the handler that will handle data indications for this cluster (for example, the scene cluster uses ZCL_SceneClusterServer().

3.3 Attribute Definitions

An attribute definition includes:

```
typedef struct zclAttrDef_tag
{
  zclAttrId_t    id;       /* attribute ID, cluster specific */
  zclAttrType_t  type;     /* attribute type (see 8.2 in ZCL Foundation) */
  zclAttrFlags_t flags;    /* attribute flags */
  uint8_t        maxLen;   /* max length for strings */
  zclAttrData_t  data;     /* ptr to data, or the data itself or offset */
} zclAttrDef_t;
```

Attributes include an identifier and a type as defined by the ZigBee Cluster Library. The flags are unique to BeeStack and allow BeeStack to be more efficient on RAM usage or indicate certain properties of the attributes. The possible flags are:

- gZclAttrFlagsInROM_c - in ROM only (not in RAM) (implies RdOnly, strings are C encoded)
- gZclAttrFlagsInLine_c - attribute is in-line in ROM (not pointed to)
- gZclAttrFlagsCommon_c - in RAM, stared among endpoints
- gZclAttrFlagsInRAM_c - in the RAM data set (one per endpoint), not reportable
- gZclAttrFlagsReportable_c - Can this attribute be reported? (also add to reportable list)
- gZclAttrFlagsRdOnly_c - in RAM, but attribute is read only (defaults to RD/WR). This is a modifier to the above flags.
- gZclAttrFlagsInSceneTable_c - the attribute is specified in the scene table extension
- gZclAttrFlagsWrOnly_c - the attribute is write only.
- gZclAttrFlagsDefinedOutOfBand_c - the attribute is defined using an out-of-band method and not over the air.

- gZclAttrFlagsNwkSecurityRequired_c – needs security to access the attribute
- gZclAttrFlagsApsLinkKeySecurityRequired_c – needs Link Key to access the attribute
- gZclAttrFlagsAsynchronous_c - ZCL code will not read the attribute and send the response automatically, it must be done by the application.

The maxLen field is used only for strings and specifies their maximum (as opposed to current) length.

The data is variable, but if the data is small (16-bits or less) and read-only, then it may be in-line using the gZclAttrFlagsInLine_c flag.

3.4 Adding an Attribute

Adding an attribute takes a number of steps. This example shows how to add a new custom attribute to the on/off cluster. The attribute indicates whether the light is working or not (or is non-functioning, such as a burnt out bulb).

1. Add the attribute to the attribute definitions for the cluster. For example, to add an attribute to the OnOff Cluster, the following code was added. This code can be found in `ZclGeneral.c`.

```
const zclAttrDef_t gaZclOnOffClusterAttrDef[] = {
{ gZclAttrOnOff_OnOffId_c, gZclDataTypeBool_c,
  ZclAttrFlagsInRAM_c|gZclAttrFlagsRdOnly_c| gZclAttrFlagsReportable_c,
  sizeof(uint8_t), (void *)MbrOfs(zclOnOffAttrsRAM_t, onOff) },
{ gZclAttrOnOff_Working_c, gZclDataTypeBool_c, gZclAttrFlagsInRAM_c,
  sizeof(uint8_t), (void *)MbrOfs(zclOnOffAttrsRAM_t, working) }
};
```

Notice that the type and size of the attribute are specified in addition to the attribute identifier. Also notice the offset within the zclOnOffAttrsRAM_t structure. The ZigBee Cluster Library knows how to find the data item by using the following formula:

Offset to variable = Pointer to instance data (in device definition) +
offset within instance data to cluster data (in cluster definition) +
offset within cluster data to attribute data (in attribute definition)

2. Update the clusters RAM data structure, adding in the new attribute field. This code can be found in `ZclGeneral.h`.

```
typedef struct zclOnOffAttrsRAM_tag
{
  uint8_t  onOff[zclReportableCopies_c];
  uint8_t  working;
} zclOnOffAttrsRAM_t;
```

The field “working” was added to the structure, as specified in the attribute definition. However, unlike the OnOff attribute, “working” only has one copy. The OnOff attribute requires three fields to represent the attribute because it is reportable. The “working” field is not reportable and so needs only one field.

The reportable attributes need three copies because they can be reported on change in value, so there is one field for the current value, one field for the old value, and one field for the difference required before a report will be issued.

3.5 Adding a Second Device Instance

This section describes adding a second device instance (a separate endpoint). For example, a single ZigBee node may contain multiple lights.

This example will add a second light to an OnOffLight application. This example assumes the reader is proficient in the C programming language.

1. Create a new project from the HaOnOffLight template.
2. Add the second endpoint (including simple descriptor).

For this step, open the file `EndPointConfig.c`.

- Make a copy of the Endpoint8_simpleDescriptor structure, and call it Endpoint9_simpleDescriptor, changing the endpoint # (the first field) to 9.
- Make a copy of the Endpoint8_EndPointDesc and call it Endpoint9_EndPointDesc and make sure the first field points to the Endpoint9_simpleDescriptor.
- Add a new entry in the endPointList structure that looks like the following {&Endpoint9_EndPointDesc, &gHaOnOffLightDeviceDef9}

3. Update number of endpoints in `EndPointConfig.h`. This will ensure the application registers the endpoint. See BeeAppInit()

- Set gNum_EndPoints_c to 2.

4. Add new data instance to HaOnOffLightEndPoint.

- Make a copy of the structure gHaOnOffLightData, and call it gHaOnOffLightData9
- Make a copy of the structure called gHaOnOffLightDeviceDef and call it gHaOnOffLightDeviceDef9. Make sure the last field in field in gHaOnOffLightDeviceDef is modified to point to gHaOnOffLightData9 (the new instance of the HaOnOffLight device).

5. Modify `BeeApp.c` for the HaOnOffLight to include code support for the new light.

Modify the function BeeAppUpdateDevice(). Within the template, the endpoint is ignored. Update the case gZclUI_Off_c and the case gZclUI_On_c to use another LED (perhaps 4?) or some other mechanism for the second endpoint.