ZIGBEE SMART ENERGY PROFILE SPECIFICATION

Smart Energy
Profile Specification

ZigBee Profile: 0x0109

Revision 15

ZigBee Document 075356r15

December 1, 2008 1:06 pm

Sponsored by: ZigBee Alliance

Accepted by ZigBee Alliance Board of Directors.

Abstract

Keywords ZigBee, Profile, AMI, Application Framework, Smart Energy.

December 1, 2008

This page intentionally blank

NOTICE OF USE AND DISCLOSURE

ZigBee Alliance, Inc.
2400 Camino Ramon, Suite 375
San Ramon, CA 94583, USA

This page intentionally blank

TABLE OF CONTENTS

This page intentionally blank

LIST OF TABLES

This page intentionally blank

LIST OF FIGURES

PARTICIPANTS

Contact Information

Much of the information in this document is preliminary and subject to change. Members of the ZigBee Working Group are encouraged to review and provide inputs for this proposal. For document status updates, please contact:

Phil Jamieson
Philips, Cross Oak Lane,
Redhill, Surrey, RH1 5HA, UK

E-Mail: phil.jamieson@philips.com
Phone: +44 1293 815265
Fax: +44 1293 815050

You can also submit comments using the ZigBee Alliance reflector. Its web site address is:

www.zigbee.org

The information on this page should be removed when this document is accepted.

Participants

The following is a list of those who were members of the ZigBee Alliance Application Framework Working Group leadership when this document was released:

Phil Jamieson: Chair

Don Sturek: Editor-in-chief

Drew Gislason: Secretary

When the document was released, the Smart Energy Profile Task Group leadership was composed of the following members:

Robby Simpson: Chair

Tim Gillman: Vice Chair

Dan Lohman: Technical Editor

Matt Maupin: Program Manager

Contributions were made to this document by the following members:

Mads Westergreen	Jeff Hugghins	Gary Birk	Don Sturek
Skip Aston	Robert Cragie	Zachary Smith	Howard Ng
Michel Veillette	John Cowburn	Damon Corbin	Tim Enwall
Rick Bojahra	Juan Agüí Martín	Jakob Thomsen	Christopher Leidigh
Jeff Mathews	John Prater	Wally Barnum	
Rob Alexander	Donald Hasson	John Knuth	
John Buffington	Yuri Shteinman	Michael G. Stuber	

DOCUMENT HISTORY

Table 1.1 shows the change history for this specification.

Table 1.1 Document Revision Change History

Revision	Version	Description
0		Original version.
1		First draft to include annexes and cluster information.
2		Updated to include Key Establishment Cluster Annex. Added other minor changes within the document.
3		Included comments from internal Smart Energy (formerly Smart Energy) group review. New Items: Added Power Factor in Simple Metering cluster. Added Group support in the DR/LC cluster.
4		Included comments from internal Smart Energy group review scattered throughout. A number of field and attribute adds to the clusters.
5		Corrected Document Number issues, otherwise same as Revision 4.
6		Additional changes: Usual grammar and spelling changes Load Profile commands have been updated. Added Attributes to support the latest partial LP interval. Load Control rules for DR/LC Randomization ESP Historical Attributes. changes in Simple Metering cluster Changes to the Get Current Price command
7		Grammar, spelling, and formatting changes
8		PDF version of 07
9		First pass at comment resolution. Please refer to document #075424r03ZB for changes.
10		Second pass at comment resolution. Please refer to document #075424r04ZB for changes. Renamed to Smart Energy Profile.

Table 1.1 Document Revision Change History (Continued)

11		Third pass at comment resolution. Please refer to document #075424r05ZB for changes. Moved SE Cluster definitions into the annex D. Significant changes in the Security related sections. Best practice information added to more sections. Updated Annex E covering overlapping event examples. Corrected issues relating to the following CCBs: CC-900 [SE] Randomizing Price Events CC-901 [SE] Message Cluster Start Time CC-902 [SE] New Status Field for Get Profile Response Command CC-903 [SE] Support for Binding CC-904 [SE] ESP Historical Consumption Attributes in Simple Metering Server CC-905 [SE] More Precise Event Status Enumeration for Report Event Status Command CC-906 [SE] Additional Description of Device Class bits 0 and 1 CC-907 [SE] Array vs. Series of Intervals for Get Profile Response Command CC-908 [SE] Randomization of Report Event Status Command Send Times CC-909 [SE] Effective Time Field of Cancel Load Control Event to be Mandatory CC-910 [SE] Consolidation of Joining Procedures CC-911 [SE] Out of Bands Methods of Authentication CC-912 [SE] Method to Make Registered Devices Listed on ESP CC-913 [SE] Clarification of Rate Label Field of Publish Price Command.
12		PDF version of 075356r11.

Table 1.1 Document Revision Change History (Continued)

13		Converted from Word to FrameMaker, includes all CCBs called out in the SE Profile Errata 08119r08.
14		Final editorial changes for initial publication.
15		Corrected issues related to the following CCBs (from errata document 084914r05): CC-964 ZigBee Cluster Library reference doesn't contain the revision number. CC-965 Specification needs to clarify the service discovery process steps prior to and after the Key Establishment process. End Devices must also initiate the processes. CC-966 The Identify cluster should be Optional, not mandatory. CC-967 The Common Features and Functions table incorrectly calls out the binding and service discovery requests as mandatory items. CC-968 Future definitions of fields added to the end of commands are to be treated as reserved fields. CC-973 Addition of Greenhouse Gas (CO_2) pricing information to the Publish Price command. CC-974 Addition of Supply Limit tracking in the Simple Metering Cluster. CC-980 Correct and describe CRC Algorithm used for Installation Codes. CC-981 Correct the Installation Codes text examples and provide example source code for testing/using the MMO Hash Algorithm. CC-982 Attributes *CurrentPartialProfileIntervalValueDelievered* and *CurrentPartialProfileIntervalValueReceived* do not list default values or mandatory/optional status. CC-983 Attributes *Power Factor*, *ReadingSnapShotTime*, *CurrentMaxDemandDelieveredTime*, and *CurrentMaxDemandReceivedTime* are incorrectly replicated in another section. Corrected issues related to the following CCBs: CC-984 Addition of Key Establishment test vectors. CC-986 Addition of metering device types to the simple metering cluster attribute MeteringDeviceType Enumeration. CC-993 Initiate Key Establishment Request and Response Payload Format field names need to match field names defined in Payload Format figures.

This page intentionally blank

CHAPTER

1

INTRODUCTION

1.1 Scope

This profile defines device descriptions and standard practices for Demand Response and Load Management "Smart Energy" applications needed in a Smart Energy based residential or light commercial environment. Installation scenarios range from a single home to an entire apartment complex. The key application domains included in this initial version are metering, pricing and demand response and load control applications. Other applications will be added in future versions.

1.2 Purpose

This specification provides standard interfaces and device definitions to allow interoperability among ZigBee devices produced by various manufacturers of electrical equipment, meters, and Smart Energy enabling products.

This page intentionally blank

CHAPTER

2

REFERENCES

2.1 References

The following standards and specifications contain provisions, which through reference in this document constitute provisions of this specification. All the standards and specifications listed are normative references. At the time of publication, the editions indicated were valid. All standards and specifications are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the standards and specifications indicated below.

2.1.1 ZigBee Alliance Documents

[B1] ZigBee Document 064321r08, The ZigBee Stack Profile, ZigBee Architectural Sub-Committee of the TSC (TAG)

[B2] ZigBee document 075123r01[1], ZigBee Cluster Library Specification, ZigBee Application Framework Working Group.

[B3] ZigBee document 064309r04, Commissioning Framework

[B4] ZigBee Document 053474r17, The ZigBee Specification, ZigBee Technical Steering Committee (TSC)

[B5] ZigBee Document 074855r04, The ZigBee PRO Stack Profile, ZigBee Architectural Sub-Committee of the TSC (TAG)

[B6] ZigBee Document 03084r00, ZigBee Key Establishment Proposal Certicom

[B7] ZigBee 075297r04, Proposal for Inter-PAN Exchange of Data in ZigBee

1. CCB 964

2.1.2 External Reference Documents

[B8] Institute of Electrical and Electronics Engineers, Inc., IEEE Std. 802.15.4 2003, IEEE Standard for Information Technology Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs). New York: IEEE Press. 2003

[B9] ANSI X9.62-2005, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), American Bankers Association. Available from http://www.ansi.org.

[B10] ANSI X9.63-2001, Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography, American Bankers Association, November 20, 2001. Available from http://www.ansi.org.

[B11] NIST Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised), March 2007. Available from http://csrc.nist.gov.

[B12] NIST Special Publication 800-38C, Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, May 2004. Available from http://csrc.nist.gov.

[B13] FIPS Pub 197, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, US Department of Commerce/N.I.S.T, Springfield, Virginia, November 26, 2001. Available from http://csrc.nist.gov.

[B14] FIPS Pub 198, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication 198, US Department of Commerce/N.I.S.T., Springfield, Virginia, March 6, 2002. Available from http://csrc.nist.gov.

[B15] Standards for Efficient Cryptography: SEC 1 (working draft) ver 1.7: Elliptic Curve Cryptography, Certicom Research, November 13, 2006. Available from http://www.secg.org

[B16] Standards for Efficient Cryptography: SEC 4 (draft) ver 1.1r1: Elliptic Curve Cryptography, Certicom Research, June 9, 2006. Available from http://www.secg.org

[B17] RFC 3280: Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile. IETF, April 2002. Available from http://www.ietf.org

CHAPTER

3

DEFINITIONS

3.1 Conformance Levels

Expected: A key word used to describe the behavior of the hardware or software in the design models assumed by this Profile. Other hardware and software design models may also be implemented.

May: A key word indicating a course of action permissible within the limits of the standard (“may” equals “is permitted”).

Shall: A key word indicating mandatory requirements to be strictly followed in order to conform to the standard; deviations from shall are prohibited (“shall” equals “is required to”).

Should: A key word indicating that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; that a certain course of action is preferred but not necessarily required; or, that (in the negative form) a certain course of action is deprecated but not prohibited (“should” equals “is recommended that”).

3.2 ZigBee Definitions

Attribute: A data entity which represents a physical quantity or state. This data is communicated to other devices using commands.

Cluster: A container for one or more attributes and/or messages in a command structure.

Cluster identifier: A reference to the unique enumeration of clusters within a specific application profile. The cluster identifier is a 16-bit number unique within the scope of the application profile and identifies a specific cluster. Cluster identifiers are designated as inputs or outputs in the simple descriptor for use in creating a binding table.

Device: A description of a specific device within an application profile. For example, the light sensor device description is a member of the home automation application profile. The device description also has a unique identifier that is exchanged as part of the discovery process.

Node: Same as a unit.

Product: A product is a unit that is intended to be marketed. It implements application profiles that may be a combination of private, published, and standard.

Service discovery: The ability of a device to locate services of interest.

Unit: A unit consists of one or more physical objects (e.g., switch, controller, etc.) and their corresponding application profile(s) that share a single 802.15.4 radio. Each unit has a unique 64-bit IEEE address.

ZigBee coordinator: An IEEE 802.15.4-2003 PAN coordinator.

ZigBee end device: an IEEE 802.15.4-2003 RFD or FFD participating in a ZigBee network, which is neither the ZigBee coordinator nor a ZigBee router.

ZigBee router: an IEEE 802.15.4-2003 FFD participating in a ZigBee network, which is not the ZigBee coordinator but may act as an IEEE 802.15.4-2003 coordinator within its personal operating space, that is capable of routing messages between devices and supporting associations.

CHAPTER

4

ACRONYMS AND ABBREVIATIONS

AES	Advanced Encryption Standard
AMI	Advanced Metering Infrastructure or Advanced Metering Initiative
BPL	Broadband over Power Lines
CA	Certificate Authority
CBKE	Certificate-based Key Establishment
CT	Commissioning Tool
ECDSA	Elliptic Curve Digital Signature Algorithm
ECMQV	Elliptic Curve Menezes-Qu-Vanstone
EMS	Energy Management System
EPID	Extended PAN Identifier
ESP	Energy Service Portal
EUI64	Extended Universal Identifier-64
GPRS	General Packet Radio Service
HA	Home Automation
HAN	Home Area Network
IHD	In-Home Display
IVR	Interactive Voice Response
MAC	Medium Access Control (referring to protocol stack sublayer)
MAC	Message Authentication Code (referring to cryptographic operation)
MRD	Market Requirements Document
NAN	Neighborhood Area Network
PAN	Personal Area Network

PKKE	Public Key Key Establishment
PCT	Programmable Communicating Thermostat
PID	PAN Identifier
RFD	Reduced Functionality Device
SAS	Startup Attribute Set
SE	Smart Energy
SKKE	Symmetric Key Key Exchange
TC	Trust Center
TOU	Time of Use
UKE	Unprotected Key Establishment
UTF-8	8-bit Unicode Transformation Format Unicode Transformation Format
ZCL	ZigBee Cluster Library
ZDP	ZigBee Device Profile

CHAPTER

5

PROFILE DESCRIPTION

5.1 A ZigBee Smart Energy Network

The Smart Energy market requires two types of ZigBee networks for metering and energy management. These include neighborhood area networks for meters, using ZigBee for sub-metering within a home or apartment, and using ZigBee to communicate to devices within the home. Different installations and utility preferences will result in different network topologies and operation and this profile must allow for these differences. However, each of these networks will operate using the same Basic Principles to ensure interoperability.

Because of the type of data and control within the Smart Energy network, application security is a key requirement. The application will use link keys which are optional in the ZigBee and ZigBee Pro stack profiles but are required within a Smart Energy network. The Trust Center and all devices on the Smart Energy network must support the installation and use of these keys as described in the security section.

Other devices within a home may also be capable of receiving public pricing information and messages from the metering network. These devices may not have or need all the capabilities required to join a Smart Energy network. Mechanisms are provided to publish public pricing data and messages to these devices without requiring they join the Smart Energy network. These mechanisms are described in the sections describing both the public pricing and message exchanges.

Metering networks are primarily installed by specialized service personnel, but other devices in the network may be added by home owners, or home automation professionals who may not have any ZigBee expertise. Installation concepts must be easy and uniform across Smart Energy device manufacturers.

Smart Energy networks could include both ZigBee 2007 and ZigBee 2007 Pro nodes. It is recommended the majority of the nodes in the network should be

based on one stack profile or the other to get consistent performance. ZigBee Smart Energy certified products must be based upon a ZigBee Compliant Platform (ZCP). If the Smart Energy profile resides in conjunction with a private profile, the product should be ZigBee Manufacturer Specific Profile (MSP) certified and must be Smart Energy ZCP certified. This additional certification provides a reassurance that the underlying stack is behaving properly and the application is not abusive to the network.

Smart Energy networks will not interact with a consumer ZigBee Home Area Network unless a device is used to perform an "application level bridge" between the two profiles or the HA devices satisfy the Smart Energy profile security requirements. This is due to the higher security requirements on the Smart Energy network that are not required on a Home network. However, it is expected that Home Automation devices that are extended to include the Smart Energy profile can still operate in a home network.

The ZigBee Smart Energy Network makes possible networks such as the following:

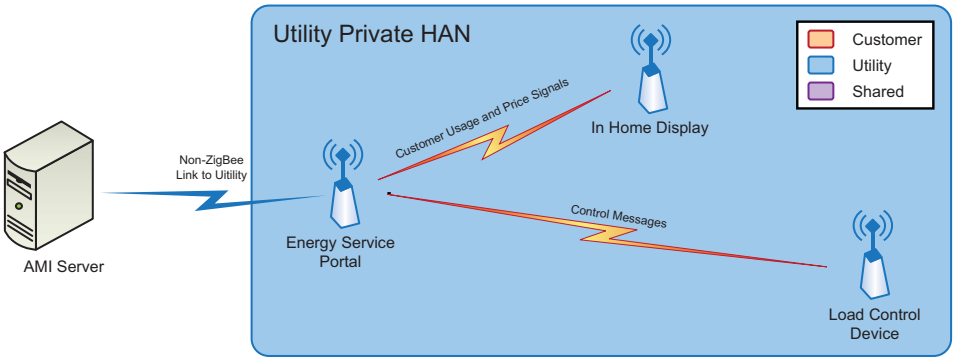

Figure 5.1 Utility Private HAN

Utility Private HAN might include an in-home display, or a load control device working in conjunction with energy service portal, but it would not include any customer controlled devices.

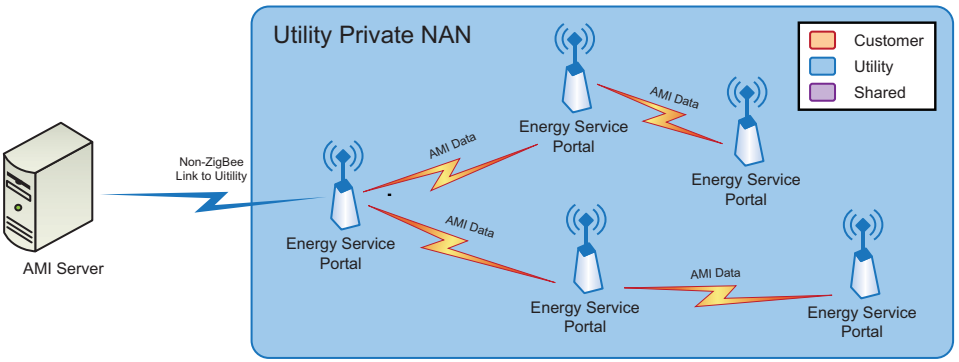

Figure 5.2 Utility Private NAN

Utility Private ZigBee network might also be used as a NAN, where ZigBee provided the primary communications for a Smart Energy deployment.

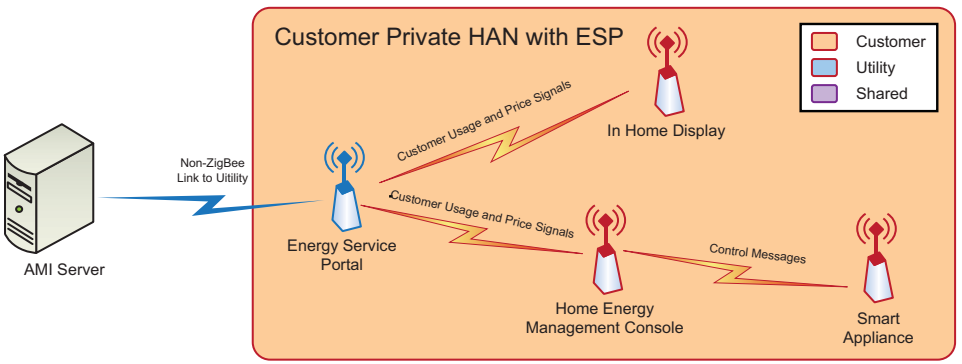

Figure 5.3 Customer Private HAN

ESP provided by utility, but limited to the role of information provider (Usage and Pricing) into a customer HAN that utilizes an Energy Management Console for conveying or controlling local devices. An example is controlling a smart appliance based upon a pricing signal.

5.2 ZigBee Stack Profile

Products that conform to this specification shall use stack profile number 0x01 or profile 0x02, as defined in [B1] [B5]. In addition to the requirements specified in [B1], the following requirements are mandatory for this application profile.

- Support for Application link keys is required.

- Fragmentation is required. Please refer to 5.3.8 regarding fragmentation sizes and parameter settings.
- In their normal operating state, ZigBee end devices shall poll no more frequently than once every 7.5 seconds except where this specification indicates otherwise for a particular device description, or under the following conditions.
  - ZigBee end devices may operate with a higher polling rate during commissioning, network maintenance, alarm states, and for short periods after transmitting a message to allow for acknowledgements and or responses to be received quickly, but they must return to the standard rate indicated previously during normal operation.
  - It is recommended that ZigBee end devices poll much less frequently than once per 7.5 seconds, especially when the device normally only communicates due to user interaction. To further clarify, except for one condition in the Simple Metering cluster (refer to DD.3.3), all cluster interactions that read or write attributes, or cause command exchanges should limit transactions to once every 30 seconds.

5.2.1 ZigBee Coordinator and Trust Center Recommendations

- In a Smart Energy based HAN network the ESP should act as the coordinator and trust center of the network.
- In a Smart Energy based NAN the backhaul point is likely to be the coordinator and trust center.

5.3 Startup Attribute Set (SAS)

In order to insure interoperability, all ZigBee Smart Energy devices shall implement compatible Startup Attribute Sets (SAS) as defined in this specification. This does not mean that the set must be modifiable through a commissioning cluster, but that the device must internally implement these stack settings to insure compatibility and consistent user experience. The startup set parameters described by the commissioning cluster in [B3] provide a good basis to specify a Smart Energy start up set.

Because Smart Energy Devices are likely to be preconfigured at a warehouse and installed by a technician, a specific start up set values may be established by a particular utility or service area and these startup set values used in place of these below for installation. The startup set values that would be expected to be set by the installer are noted below.

5.3.1 Startup Parameters

The startup parameters and their default values are listed in Table 5.1.

Table 5.1 Startup Parameters

Parameter	Value	Comment
Short Address	0xFFFF or installer specified.	
E PANiD	0x0000000000000000 or installer specified.	
PAN ID	0xFFFF or installer specified.	
Channel Mask	All channels in frequency band.	If needed, the power transmitted by the device on channel 26 can be lowered to comply with FCC regulations.
Protocol Version	0x02 (ZigBee and later)	
Stack Profile	1 (ZigBee) or 2 (ZigBee PRO)	
Startup Control	2 (two) if un-commissioned, so it will join network by association when join command is indicated. 0 (zero) if commissioned. Indicates that the device should consider itself a part of the network indicated by the ExtendedPANId attribute. In this case it will not perform any explicit join or rejoin operation.	
Trust Center Address	0x0000000000000000 or installer specified.	Please note: Identifying or establishing the Trust Center as a device other than the Coordinator is an optional stack profile feature. Since this is an implementation specific issue, installation tools and setting address is managed by the Smart Energy vendors.
Master Key		Not used, high security is not used in this profile.

Table 5.1 Startup Parameters (Continued)

Link Key	0x00000000000000000000000000000001 if the Key Establishment Cluster is being used to install a link key Installer provided if using preconfigured link keys	
Network Key	0x00000000000000000000000000000001 if no pre-installed key present	
Use Insecure Join	0x00 (False)	flag that disables the use of insecure join as a fallback case at startup time

5.3.2 Join Parameters

The join parameters and their default values are listed in Table 5.2.

Table 5.2 Join Parameters

Parameter	Value	Comment
ScanAttempts		At boot time or when instructed to join a network, the device should complete up to three (3) scan attempts to find a ZigBee Coordinator or Router with which to associate. If it has not been commissioned, this means that when the user presses a button or uses another methodology to join a network, it will scan all of the channels up to three times to find a network that allows joining. If it has already been commissioned, it should scan up to three times to find its original PAN to join. (ZigBee Pro devices should scan for their original extended PAN ID and ZigBee (2007) devices can only scan for their original PAN ID).

Table 5.2 Join Parameters

TimeBetween Scans	1 second	Determines the number of seconds between each scan attempt.
RejoinInterval	60 seconds or shorter	How quickly a device will attempt to rejoin the network if it finds itself disconnected.
MaxRejoinInterval	15 minutes	Imposes an upper bound on the RejoinInterval parameter - this must be restarted if device is touched by human user, i.e. by a button press. This parameter is intended to throttle how often a device will scan to find its network in case the network is no longer present and therefore a scan attempt by the device would always fail (i.e., if a device finds it has lost network connectivity, it will try to rejoin the network, scanning all channels if necessary). If the scan fails to find the network, or fails to successfully rejoin, the device will wait for 15 minutes before attempting to rejoin again. To be network friendly, it would be recommended to adaptively extend this time period if successive rejoins fail. It would also be recommended the device should try a rejoin when triggered (via a control, button, etc.) and fall back to this interval if rejoins fail again.

5.3.3 Security Parameters

The security parameters and their default values are listed in Table 5.3.

Table 5.3 Security Parameters

Parameter	Value	Comment
SecurityTimeoutPeriod	Set by stack profile.	
TrustCenterNetworkKey	The Trust Center will pick the network key.	ZigBee Smart Energy devices shall depend on either pre-configured keys to be commissioned or the use of the Key Establishment Cluster with a pre-configured Trust Center link key to get the network key (not in the clear). ZigBee Smart Energy networks will not generally send keys in the clear.

5.3.4 End Device Parameters

The end device parameters and their default values are listed in Table 5.4.

Table 5.4 End Device Parameters

Parameter	Value	Comment
IndirectPollRate	Set by stack profile	This is how often a device will poll its parent for new data. It is recommended that an end device that is designed to receive data should poll its parent every 60 seconds.

5.3.5 Link Status Parameters

The link status parameters and their default values are listed in Table 5.5.

Table 5.5 Link Status Parameters

Parameter	Value	Comment
LinkStatusPeriod	Set by stack profile	
RouterAgeLimit	Set by stack profile	
RepairThreshold	Set by stack profile	

5.3.6 Concentrator Parameters

The concentrator parameters and their default values are listed in Table 5.6.

Table 5.6 Concentrator Parameters

Parameter	Value	Comment
ConcentratorFlag	Set by stack profile	Identifies the device to be a concentrator.
ConcentratorRadius	11 (eleven)	Device manufacturers that produce a concentrator product will set the max concentrator radius to this value.
ConcentratorDiscoveryTime	Set by stack profile	Identifies how often the Concentrator network layer should issue a route request command frame.

5.3.7 APS Transport Parameters

The APS transport parameters and their default values are listed in Table 5.7.

Table 5.7 APS Transport Parameters

Parameter	Value	Comment
MaxFrameRetries	Set by stack profile	This determines the maximum number of retries allowed after a transmission failure.
AckWaitDuration	Set by stack profile	This is the maximum number of seconds to wait for acknowledgement of an APS frame.

5.3.8 APS Fragmentation Parameters

For fragmentation there are application settings from the APS IB that must be defined by the application profile. For Smart Energy these parameters are to be set as shown in Table 5.8.

Table 5.8 APS Fragmentation Parameters

Parameters	Identifier	Type	Value	Description
apsInterframe Delay	0xc9	Integer	50	Standard delay in milliseconds between sending two blocks of a fragmented transmission (see [B4] sub-clause 2.2.8.4.5)
apsMaxWindowSize	0xcd	Integer	1	Fragmentation parameter – the maximum number of unacknowledged frames that can be active at once (see [B4] sub-clause 2.2.8.4.5).

In addition the Maximum Incoming Transfer Size Field in the Node descriptor defines the largest ASDU that can be transferred using fragmentation. For the Smart Energy Profile the default value shall be set to 128 bytes. Maximum ASDU size allowed is specified in [B4] and dictated by solution needs and RAM capacities of the communicating devices.

It is highly recommended all devices first query the Node Descriptor of the device it will communicate with to determine the Maximum incoming transfer size (if ASDU size is greater 128 bytes). This will establish the largest ASDU that can be supported with fragmentation. The sending device must use a message size during fragmentation that is smaller than this value.

5.3.9 Binding Parameters

The binding parameters and their default values are listed in Table 5.9.

Table 5.9 Binding Parameters

Parameter	Value	Comment
EndDeviceBindTimeout	60 seconds	Timeout value for end device binding. End Device binding is set by the coordinator.

5.4 Smart Energy Profile Security

To be part of a Smart Energy network, a device shall associate using one of the two association methods described below and require the use of the Key Establishment Cluster (see Annex C) for installation and updating of link keys.

All devices shall have the ability to retain their joining and security settings through power outages.

5.4.1 Joining with Preinstalled Trust Center Link Keys

When using preinstalled trust center link keys, the following steps are used:

1. Trust Center link keys are installed in each device prior to joining the utility network.
2. The trust center link key for a device that is to be joined is provided to the local trust center through an out of band means as described in sub-clause 5.4.8.1 “Out of Band Pre-Configured Link Key Process”.
3. Permit joining is turned on in the network.
4. The device joins the network and is sent the network key encrypted with the key-transport key derived for the preinstalled trust center link key. The procedure for doing this is detailed in Annex F, also reference [B4] section 4.5.4 on key-transport keys and [B4] section 4.4.1 on frame security for the APS layer.
5. The trust center must update the pre-configured trust center link key in the joining device using the Key Establishment Cluster after completion of the joining procedure.
6. The trust center of the network has the option of later updating the trust center link keys with devices in the network as desired by the application using the Key Establishment Cluster. Updating security keys should be an infrequent operation.

7 If devices leave the network, the trust center shall update remove the network trust center link key assigned to that device.

5.4.2 Re-Joining a Secured Network

5.4.2.1 Rejoining Node Operation

When a device is re-joining a secured network, the following steps are used:

1 Permit joining is not required to be on in the network.

2 The device shall attempt a rejoin using the procedure detailed in [B4] Section 3.6.1.4.2 with network security. The network key and sequence number used will be the ones previously obtained from the trust center.

3 If the secured rejoin is successful, nothing more is required from the device.

4 If the secured rejoin fails, the device shall attempt a rejoin using the procedure detailed in [B4] Section 3.6.1.4.2 without network security. The re-joining device is assumed to have previously joined the network and obtained a link key using the key establishment cluster procedures. If the device does not have a link key obtained via the key establishment cluster, it cannot rejoin the network.

5 If the unsecured rejoin fails the device may attempt it again. If the device is told to leave the network it may employ the Joining using the Key Establishment Cluster procedure.

5.4.2.2 Trust Center Operation

When the trust center receives notification that a device has rejoined the network, the following steps are used:

1 If the device performed a secured rejoin the trust center is not required to take any action.

2 If the device performed an unsecured rejoin the trust center shall determine if the device is authorized to be on the network. If the trust center has a link key with the device that was established using the key establishment cluster then it shall be allowed back on the network. The trust center should send out an updated copy of the network key encrypted with the corresponding link key.

3 If the trust center determines that the device is not authorized to be on the network, it shall send an APS remove device command to the parent of the rejoining device, with the target address of the rejoining device's IEEE address. The parent will then remove that device from its child table.

5.4.3 Devices Leaving the Network

Upon receipt of an APS update device command indicating a device has left the network, the trust center shall remove the trust center link key assigned to that device.

5.4.4 Updating the Network Key

Periodically the trust center shall update the network key. This allows the trust center to phase out a previous instance of the network key so that devices that are no longer on the network will not be able to perform a secure rejoin. Those devices must then perform an unsecured rejoin, which allows the trust center to authorize whether or not they are allowed to be on the network.

When the trust center wishes to update the network key it will broadcast the network key to all devices in the network. All devices receiving the key update will store but will not start using the new key.

It is assumed that routers will receive the network key update sent by the Trust Center. Sleepy end devices are unlikely to get the network key update sent by the Trust Center unless the device polls frequently.

After sending an updated network key, the trust center shall wait a minimum of *nwkNetworkBroadcastDeliveryTime* before sending the switch key message. Devices that miss the key switch broadcast message will implicitly switch when they receive any network message that is encrypted using the new key sequence number.

Once the network has started using the new key, any device that has missed the key update message will not be able to communicate on the network. Those devices must that missed the key update must follow the Re-joining a Secured Network procedure.

5.4.5 Updating the Link Key

Periodically the trust center may update the link key associated with a particular device. This allows the trust center to phase out the existing key and refresh it with a new key. The trust center can decide on its own what the policy is for how long a link key may be used and how often it should be updated.

Trust Center link keys are used for sending application messages as well as stack commands. Therefore a trust center cannot simply delete a link key that it wants to update. The trust center must accept and or send encrypted APS commands to or from a device even if has retired that link key from encryption of application data messages. This is especially necessary for sleeping end devices, which may not

have the current network key and need to use their link key to obtain an updated copy during a rejoin.

When the trust center deems that a particular link key should no longer be used, it shall mark the key as stale. A stale key shall not be used to send data messages. Devices that receive a message using a stale key should discard the message and shall not send an APS acknowledgement to the sender.

Devices shall accept and process APS commands that are encrypted with a stale key.

When the trust center receives a message encrypted with a stale link key, it shall initiate the key establishment procedure to negotiate a new link key. Upon successful establishment of the new link key with the device, the device shall clear the stale indicator for that key.

Devices that are not acting as the trust center may utilize their own policy for retiring and updating application link keys with other devices that are not the trust center. Those devices are not required to keep around retired keys and therefore may delete them prior to establishing a updated link key using the key establishment cluster.

5.4.5.1 Network Joining and Registration Diagram

Figure 5.4 depicts an example of a successful network startup and certificate exchange (with pre-established link keys). Please refer to Annex C for further discussions on communication exchanges and key support.

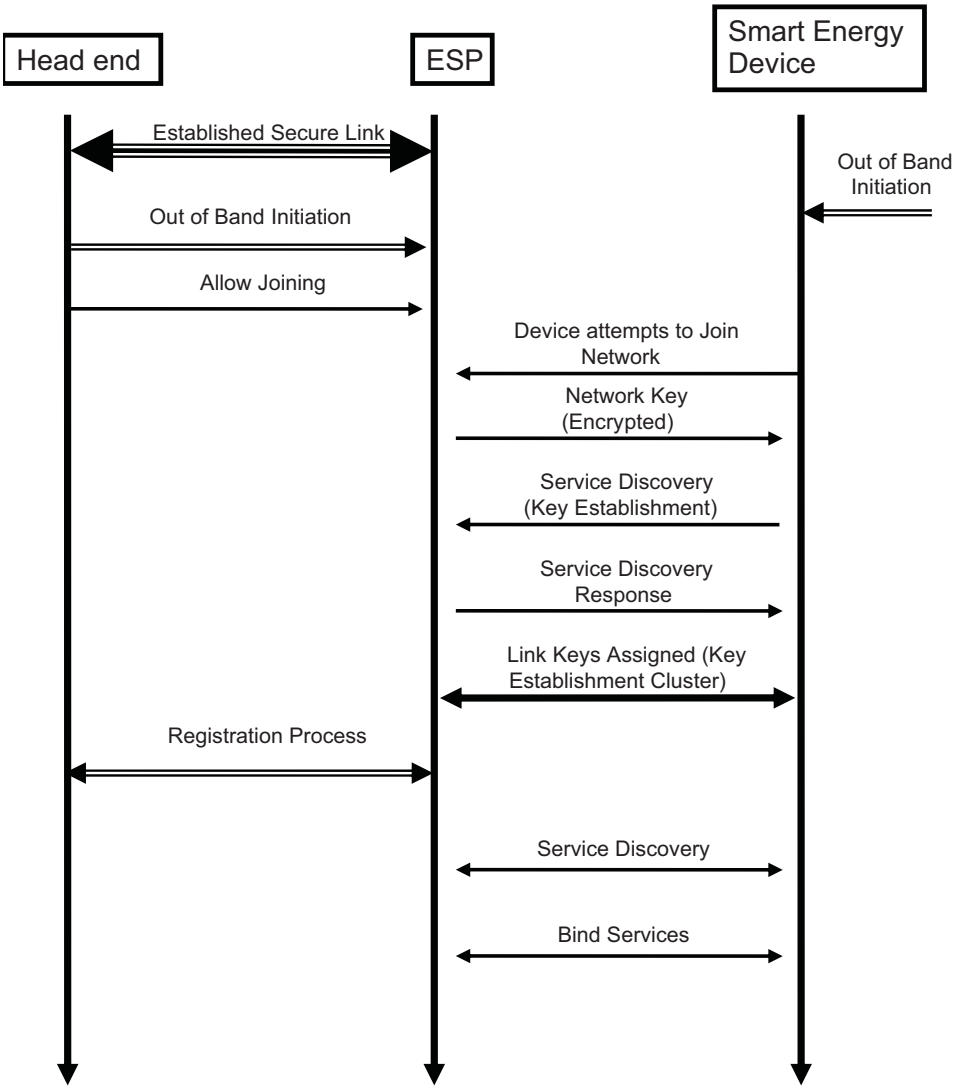

Figure 5.4 Successful Join and Registration

Please note: After joining the network and acquiring a Network Key, the Smart Energy End Device shall initiate the Service Discovery process to locate the Key Establishment Cluster. As recommended best practice, the ESP should support a fault tolerant behavior by initiating Key Establishment Cluster service discovery process whenever it detects the Smart Energy End Device fails to do[2] so.

2. CCB 965

5.4.6 Cluster Usage of Security Keys

The SE Profile utilizes a higher level of security on the network but not all clusters need to utilize Application Link keys. Table 5.10 identifies the security keys utilized by each cluster:

Table 5.10 Security Key Assignments per Cluster

Functional Domain	Cluster Name	Security Key
General	Basic	Network Key
General	Identify	Network Key
General	Alarms	Network Key
General	Time	Application Link Key
General	Commissioning	Application Link Key
General	Power Configuration	Network Key
General	Key Establishment	Network Key
Smart Energy	Price	Application Link Key
Smart Energy	Demand Response and Load Control	Application Link Key
Smart Energy	Simple Metering	Application Link Key
Smart Energy	Message	Application Link Key
Smart Energy	Smart Energy Tunneling (Complex Metering)	Application Link Key
Smart Energy	Pre-Payment	Application Link Key

Once a Registered SE device has an Application Link Key established with the ESP, it may also establish Application Link Keys with any other device on the SE Network. This is accomplished by using the ZigBee service and device discovery process (employing the Network Key). Regardless of the communication paths, all SE applications shall use and validate the Security key assignment as listed in listed in Table 5.10. If incorrect key usage is found the application shall generate a ZCL Default Response, employing the Network Key, with a FAILURE (0x01) status code.

5.4.7 Key Establishment Related Security Policies

The following are the policies relating to Key Establishment that are recommended for Smart Energy networks.

5.4.7.1 Joining

If the device doesn't need to perform discovery queries or other non-secure operations after it joins an SE network and receives the Network Key, it should immediately initiate Key Establishment with the Trust Center to obtain a new Trust Center Link Key.

If Key Establishment fails with a result of UNKNOWN_ISSUER the device should leave the network. A device that does not initiate Key Establishment with the Trust Center within a reasonable period of time should be told to leave.

Upon successful negotiation of a new Trust Center Link Key the device may communicate using clusters that require APS security.

5.4.7.2 Trust Center

The Trust Center should keep track of whether a particular device has negotiated a CBKE Trust Center Link Key, or whether only a preconfigured Trust Center Link Key exists. The Trust Center should not use the preconfigured link key to send encrypted APS Data messages to the device. The Trust Center should discard any APS encrypted APS Data messages that use the preconfigured link key, and it should not send APS Acks for those messages.

The Trust Center shall accept and send APS Data messages that do not use APS Encryption to a device that has not negotiated a CBKE Trust Center Link key provided that the security usage for that cluster only allows using Network layer security (encrypted with the Network Key). See sub-clause 5.4.6, “Cluster Usage of Security Keys”.

5.4.7.3 During Joining

Normal operation of a device in a Smart Energy network requires use of a preconfigured link key, establish by using the Installation Code (refer to sub-clause 5.4.6), to join a Zigbee Pro network. After joining the network a device is required to initiate key establishment using ECMQV key agreement with the ESP (which is also the Trust Center), to obtain a new link key authorized for use in application messages.

Prior to updating the preconfigured link key using key establishment, the ESP shall not allow Smart Energy messages that require APS encryption. Although the node has a link key, that node has not been authenticated and thus the key's use is not authorized for application messages. Its use is still required for certain stack messages (e.g. the APS Command Update Device) and must be accepted by the trust center.

Once a node has authenticated by the ESP and obtained an authorized link key using key establishment, it may communicate with the ESP using APS layer

security. The ESP should accept valid APS encrypted message using that new link key.

If a device never establishes a trust center link key after joining, the trust center may send it a network leave command. This is only done for non-security reasons, such as encouraging a well-behaved device that it is not on the correct network. Malicious nodes will never leave the network once they have the network key and there is no practical way to force them off the network. However if the above mentioned security policies are adhered to, then the malicious node will be unable to communicate with other devices since it will not have access to an authorized link key.

5.4.7.4 After Joining

After a node has joined, been authenticated using key establishment, and obtained an authorized link key, it may need to communicate with other nodes on the network using APS layer encryption.

Rather than use key establishment with each node on the network, it would be advantageous to leverage the ESP to broker trust with other devices on the network. If two nodes have both obtained link keys with the ESP using key establishment then they both trust the ESP. Both nodes will use the ESP to request a link key with each other. The trust center will respond to each node individually, sending a randomly generated link key. Each message will be encrypted using the individual nodes' link keys. The ESP would not send a link key to either node if one of the nodes has not authenticated using key establishment.

Both nodes are required to request a link key with the other node, rather than have a single node requesting the key. This added requirement insures that both nodes are online and ready to receive a key (i.e. not sleeping) and insures that a node is not forced to accept a key it cannot support or did not want.

The originating node would start this process by sending a bind request command with APS ack to the key establishment cluster of the destination device. If a bind confirm is received with a status of success, the initiating device will perform a request key of the trust center (for an application link key using the EUI of the other device in the pair). The trust center will then send a link key to each device using the key transport. If the bind confirm is received with a status other than success, the request key should not be sent to the trust center.

This functionality is optional however support of this is required for ESP devices acting as trust centers. All devices sending the request key command and the trust center should have a timeout of 5 seconds.

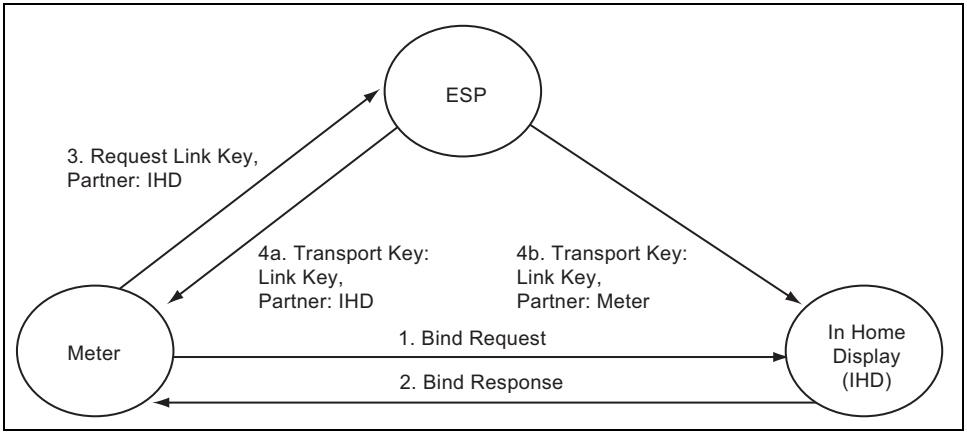

Figure 5.5 Node Communication with Other Nodes on the Network Using APS Layer Encryption

The advantages of using the stack primitives to request keys rather than key establishment are that devices can forego the expensive ECC operations. Small microprocessors have extremely limited resources and requiring full key establishment with all devices where link keys are required is overly burdensome. In addition, ESPs may have other security policies in place (such as node blacklists or certificate revocation lists) that individual nodes do not have knowledge of, or have the resources to keep track of.

Nodes that are not the trust center would not be allowed to initiate key establishment with another device that is not the Trust Center. If a device receives an Initiate Key Establishment Request from a device that is not the Trust Center, and it is not the Trust Center, it shall terminate the key establishment immediately with a status of NO_RESOURCES. This insures that the ESP authenticates all devices with key establishment after joining, and limits the use of key establishment in the network.

Other ESP devices on the network that are not the trust center would have to go through the same procedure as above, contacting the ESP trust center, in order to send/receive messages that require APS layer encryption with another node.

5.4.8 Security Best Practices

5.4.8.1 Out of Band Pre-Configured Link Key Process

This section describes the out of band process for establishing pre-configured Trust Center link keys, the format of the Installation Code required, and the hashing function used to derive the pre-configured link key from the Installation Code.

As portrayed in Figure 5.6, during the manufacturing process an Installation Code is created for each of the Smart Energy devices. This Installation Code is provided for the device in a manufacturer specific way (labeling, etc.) and referenced to during installation. The associated Pre-configured Link Key is derived using the hashing function described below and programmed in the device.

Step #1: An Installation Code is created and made available

Step #2: The Pre-configured Link Key is derived from the Installation Code using the Matyas-Meyer-Oseas hash function.

Step #3: The Pre-configured Link Key is configured in the device

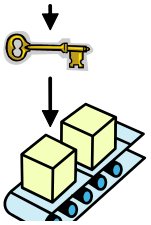

Figure 5.6 Smart Energy Device Installation Code Process

As portrayed in Figure 5.7, during the installation process the initial Trust Center Link Key is derived from the Installation Code and sent via an out of band communication channel to the Trust center (ESP). The Trust center uses this Key as the Trust Center Link Key to subsequently configure the Network Key of the associating device.

Step #1: The Installation Code is sent out of band

Step #2: The Pre-configured Link Key is derived from the Installation Code using the Matyas-Meyer-Oseas hash function

Step #3: The Pre-configured Link Key is sent to the Trust center using the AMI network

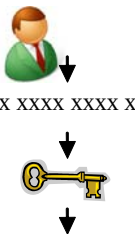

Figure 5.7 Installation Code Use with the Trust Center

5.4.8.1.1 Installation Code Format

The Installation Code consists of a 48, 64, 96, or 128 bit number and a 16 bit CRC (using CCITT CRC standard polynomial X16 + X12 + X5 + 1). When printed or displayed, Installation Codes are represented as multiple groups of 4 hexadecimal digits.

48 Bit example:

Installation Code of “83FE D340 7A93 2B70”
Where values 0x83, 0x FE, 0xD3, 0x40, 0x 7A, and 0x93 are used to calculate the CRC16 with the result returning 0x702B.

Note: The Octet order of the CRC code in the printed Installation code is Least Significant Octet followed by Most Significant Octet, giving the printed result of “2B70”.[3]

64 Bit example:

Installation Code of “83FE D340 7A93 9738 C552”
Where values 0x83, 0x FE, 0xD3, 0x40, 0x 7A, 0x93, 0x 97, and 0x38 are used to calculate the CRC16 with the result returning 0x52C5.[4]

96 Bit example:

Installation Code of “83FE D340 7A93 9723 A5C6 39FF 4C12”
Where values 0x83, 0xFE, 0xD3, 0x40, 0x7A, 0x93, 0x97, 0x23, 0xA5, 0xC6, 0x39 and 0xFF are used to calculate the CRC16 with the result returning 0x124C.[5]

128 Bit example:

Installation Code of “83FE D340 7A93 9723 A5C6 39B2 6916 D505 C3B5”
Where values 0x83, 0xFE, 0xD3, 0x40, 0x7A, 0x93, 0x97, 0x23, 0xA5, 0xC6, 0x39, 0xB2, 0x69, 0x16, 0xD5, and 0x05 are used to calculate the CRC16 with the result returning 0xB5C3.[6]

5.4.8.1.1.1 CRC Algorithm Information

As stated earlier, the Installation Code CRC calculation is based upon the CRC 16-CCITT algorithm and uses the following parameters:

Length : 16
Polynomial : x16 + x12 + x5 + 1 (0x1021)
Initialization method : Direct
Initialization value : 0xFFFF
Final XOR value : 0xFFFF
Reflected In : True
Reflected Out : True

The following free code example, copied from http://zorc.breitbandkatze.de/crctester.c can be referenced at http://zorc.breitbandkatze.de/crc.html. The code example has been modified to match the parameters needed for the Smart Energy Profile Installation code examples. The example is:

```
// ------------------------------------------------------------------------
// CRC Test V1.3 was copied from URL:
```

3. CCB 980
4. CCB 980
5. CCB 980
6. CCB 980

```
// ---------------------------------------------------------------------------
// CRC tester v1.3 written on 4th of February 2003 by Sven Reifegerste (zorc/reflex)
// This is the complete compilable C program, consisting only of this .c file.
// No guarantee for any mistakes.
//
// changes to CRC tester v1.2:
//
// - remove unneccessary (!(polynom&1)) test for invalid polynoms
//   (now also XMODEM parameters 0x8408 work in c-code as they should)
//
// changes to CRC tester v1.1:
//
// - include an crc&0crcmask after converting non-direct to direct initial
//   value to avoid overflow
//
// changes to CRC tester v1.0:
//
// - most int's were replaced by unsigned long's to allow longer input strings
//   and avoid overflows and unnecessary type-casting's
// ---------------------------------------------------------------------------

// includes:

#include <string.h>
#include <stdio.h>



// CRC parameters (default values are for CRC-32):
// const int order = 32;
// const unsigned long polynom = 0x4c11db7;
// const int direct = 1;
// const unsigned long crcinit = 0xffffffff;
// const unsigned long crcxor = 0xffffffff;
// const int refin = 1;
// const int refout = 1;
```

```
// 'order' [1..32] is the CRC polynom order, counted without the leading '1' bit
// 'polynom' is the CRC polynom without leading '1' bit
// 'direct' [0,1] specifies the kind of algorithm: 1=direct, no augmented zero bits
// 'crcinit' is the initial CRC value belonging to that algorithm
// 'crcxor' is the final XOR value
// 'refin' [0,1] specifies if a data byte is reflected before processing (UART) or not
// 'refout' [0,1] specifies if the CRC will be reflected before XOR

// CRC parameters used for the ZigBee Smart Energy Profile Installation Codes:

const int order = 16;
const unsigned long polynom = 0x1021;
const int direct = 1;
const unsigned long crcinit = 0xffff;
const unsigned long crcxor = 0xffff;
const int refin = 1;
const int refout = 1;

// Data character string

//The following test string should provide a check result of 0x906E
//const unsigned char string[] = {"123456789"};

//The following string values are from the Smart Energy Profile Installation code examples.
//The strings are constructed from the Installation Code values without the CRC Check, plus the strings are null terminated.

const unsigned char string[] = {0x83, 0xFE, 0xD3, 0x40, 0x7A, 0x93, 0x00};
// The above should provide a check result of: 0x702B

//const unsigned char string[] = {0x83, 0xFE, 0xD3, 0x40, 0x7A, 0x93, 0x97, 0x38, 0x00};
// The above should provide a check result of: 0x52C5

//const unsigned char string[] = {0x83, 0xFE, 0xD3, 0x40, 0x7A, 0x93, 0x97, 0x23, 0xA5, 0xC6, 0x39, 0xFF, 0x00};
```

```
// The above should provide a check result of: 0x124C

//const unsigned char string[] = {0x83, 0xFE, 0xD3, 0x40, 0x7A, 0x93, 0x97, 0x23,
0xA5, 0xC6, 0x39, 0xB2, 0x69, 0x16, 0xD5, 0x05, 0x00};
// The above should provide a check result of: 0xB5C3

// internal global values:
unsigned long crcmask;
unsigned long crchighbit;
unsigned long crcinit_direct;
unsigned long crcinit_nondirect;
unsigned long crctab[256];

// subroutines
unsigned long reflect (unsigned long crc, int bitnum) {

        // reflects the lower 'bitnum' bits of 'crc'
        unsigned long i, j=1, crcout=0;

        for (i=(unsigned long)1<<(bitnum-1); i; i>>=1) {
                if (crc & i) crcout|=j;
                j<<= 1;
        }
        return (crcout);
}

void generate_crc_table() {

        // make CRC lookup table used by table algorithms
        int i, j;
        unsigned long bit, crc;

        for (i=0; i<256; i++) {
                crc=(unsigned long)i;
                if (refin) crc=reflect(crc, 8);
                crc<<= order-8;
```

```
                for (j=0; j<8; j++) {
                        bit = crc & crchighbit;
                        crc<<= 1;
                        if (bit) crc^= polynom;
                }
                if (refin) crc = reflect(crc, order);
                crc&= crcmask;
                crctab[i]= crc;
        }
}

unsigned long crctablefast (unsigned char* p, unsigned long len) {

        // fast lookup table algorithm without augmented zero bytes, e.g. used in pkzip.
        // only usable with polynom orders of 8, 16, 24 or 32.
        unsigned long crc = crcinit_direct;
        if (refin) crc = reflect(crc, order);
        if (!refin) while (len--) crc = (crc << 8) ^ crctab[ ((crc >> (order-8)) & 0xff) ^
*p++];
        else while (len--) crc = (crc >> 8) ^ crctab[ (crc & 0xff) ^ *p++];

        if (refout^refin) crc = reflect(crc, order);
        crc^= crcxor;
        crc&= crcmask;
        return(crc);
}

unsigned long crctable (unsigned char* p, unsigned long len) {

        // normal lookup table algorithm with augmented zero bytes.
        // only usable with polynom orders of 8, 16, 24 or 32.
        unsigned long crc = crcinit_nondirect;

        if (refin) crc = reflect(crc, order);
```

```
        if (!refin) while (len--) crc = ((crc << 8) | *p++) ^ crctab[ (crc >> (order-8))  & 0xff];
        else while (len--) crc = ((crc >> 8) | (*p++ << (order-8))) ^ crctab[ crc & 0xff];

        if (!refin) while (++len < order/8) crc = (crc << 8) ^ crctab[ (crc >> (order-8)) & 0xff];
        else while (++len < order/8) crc = (crc >> 8) ^ crctab[crc & 0xff];

        if (refout^refin) crc = reflect(crc, order);
        crc^= crcxor;
        crc&= crcmask;

        return(crc);
}

unsigned long crcbitbybit(unsigned char* p, unsigned long len) {

        // bit by bit algorithm with augmented zero bytes.
        // does not use lookup table, suited for polynom orders between 1...32.
        unsigned long i, j, c, bit;
        unsigned long crc = crcinit_nondirect;

        for (i=0; i<len; i++) {
                c = (unsigned long)*p++;
                if (refin) c = reflect(c, 8);
                for (j=0x80; j; j>>=1) {
                        bit = crc & crchighbit;
                        crc<<= 1;
                        if (c & j) crc|= 1;
                        if (bit) crc^= polynom;
                }
        }
        for (i=0; i<order; i++) {

                bit = crc & crchighbit;
                crc<<= 1;
```

```
                if (bit) crc^= polynom;
        }
        if (refout) crc=reflect(crc, order);
        crc^= crcxor;
        crc&= crcmask;
        return(crc);
}


unsigned long crcbitbybitfast(unsigned char* p, unsigned long len) {

        // fast bit by bit algorithm without augmented zero bytes.
        // does not use lookup table, suited for polynom orders between 1...32.
        unsigned long i, j, c, bit;
        unsigned long crc = crcinit_direct;

        for (i=0; i<len; i++) {
                c = (unsigned long)*p++;
                if (refin) c = reflect(c, 8);

                for (j=0x80; j; j>>=1) {
                        bit = crc & crchighbit;
                        crc<<= 1;
                        if (c & j) bit^= crchighbit;
                        if (bit) crc^= polynom;
                }
        }
        if (refout) crc=reflect(crc, order);
        crc^= crcxor;
        crc&= crcmask;
        return(crc);
}

int main() {
        // test program for checking four different CRC computing types that are:
        // crcbit(), crcbitfast(), crctable() and crctablefast(), see above.
```

```
// parameters are at the top of this program.
// Result will be printed on the console.
int i;
unsigned long bit, crc;

// at first, compute constant bit masks for whole CRC and CRC high bit
crcmask = ((((unsigned long)1<<(order-1))-1)<<1)|1;
crchighbit = (unsigned long)1<<(order-1);

// check parameters
if (order < 1 || order > 32) {
        printf("ERROR, invalid order, it must be between 1..32.\n");
        return(0);
}
if (polynom != (polynom & crcmask)) {
        printf("ERROR, invalid polynom.\n");
        return(0);
}
if (crcinit != (crcinit & crcmask)) {
        printf("ERROR, invalid crcinit.\n");
        return(0);
}
if (crcxor != (crcxor & crcmask)) {
        printf("ERROR, invalid crcxor.\n");
        return(0);
}

// generate lookup table
generate_crc_table();

// compute missing initial CRC value
if (!direct) {
        crcinit_nondirect = crcinit;
        crc = crcinit;
        for (i=0; i<order; i++) {
                bit = crc & crchighbit;
```

```
                crc<<= 1;
                if (bit) crc^= polynom;
            }
            crc&= crcmask;
            crcinit_direct = crc;
        }
        else {
            crcinit_direct = crcinit;
            crc = crcinit;
            for (i=0; i<order; i++) {
                bit = crc & 1;
                if (bit) crc^= polynom;
                crc >>= 1;
                if (bit) crc|= crchighbit;
            }
            crcinit_nondirect = crc;
        }


        // call CRC algorithms using the CRC parameters above and print result to the
console
        printf("\n");
        printf("CRC tester v1.1 written on 13/01/2003 by Sven Reifegerste (zorc/
reflex)\n");
        printf("------------------------------------------------------------------------\n");
        printf("\n");
        printf("Parameters:\n");
        printf("\n");
        printf(" polynom          : 0x%x\n", polynom);
        printf(" order            : %d\n", order);
        printf(" crcinit          : 0x%x direct, 0x%x nondirect\n", crcinit_direct,
crcinit_nondirect);
        printf(" crcxor           : 0x%x\n", crcxor);
        printf(" refin            : %d\n", refin);
        printf(" refout           : %d\n", refout);
        printf("\n");
        printf(" data string      : '%s' (%d bytes)\n", string, strlen((const char
*)string));
```

```
        printf("\n");
        printf("Results:\n");
        printf("\n");
        printf(" crc bit by bit      :  0x%x\n", crcbitbybit((unsigned char *)string,
strlen((const char *)string)));
        printf(" crc bit by bit fast :  0x%x\n", crcbitbybitfast((unsigned char *)string,
strlen((const char *)string)));
        if (!(order&7)) printf(" crc table :  0x%x\n", crctable((unsigned char *)string,
strlen((const char *)string)));
        if (!(order&7)) printf(" crc table fast : 0x%x\n", crctablefast((unsigned char
*)string, strlen((const char *)string)));
        return(0);
}
```
[7]

5.4.8.1.2 Hashing Function

An AES-128 key is derived form the Installation Code using the Matyas-Meyer-Oseas (MMO) hash function (specified in Annex B.6 in ZigBee Document 053474r17, The ZigBee Specification, ZigBee Technical Steering Committee (TSC) with a digest size (hashlen) equal to 128 bits).

Installation Code examples:

MMO hash applied to the Installation Code "83FE D340 7A93" produces the key "CD4FA064773F46941EC986C09963D1A8".

Note: Least significant byte is 0x83 and Most significant byte is 0x93.

MMO hash applied to the Installation Code "83FE D340 7A93 9738" produces the key "A833A77434F3BFBD7A7AB97942149287".

Note: Least significant byte is 0x83 and Most significant byte is 0x38.

MMO hash applied to the Installation Code "83FE D340 7A93 9723 A5C6 39FF" produces the key "58C1828CF7F1C3FE29E7B1024AD84BFA".

Note: Least significant byte is 0x83 and Most significant byte is 0xFF.

MMO hash applied to the Installation Code "83FE D340 7A93 9723 A5C6 39B2 6916 D505" produces the key "66B6900981E1EE3CA4206B6B861C02BB".

Note: Least significant byte is 0x83 and Most significant byte is 0x05.[8]

7. CCB 980
8. CCB 981

5.4.8.1.2.1 MMO Hash Code Example

The following code example can be used to validate key creation derived from Installation codes:

```
/
*****************************************************************************
**********
 * hashing-cli.c
 *
 * This program hashes the Zigbee Smart Energy install code and CRC and
 * prints the result.  It is designed to reproduce the same results
 * as specified in the Zigbee Smart Energy specification.
 *
 * The Zigbee Matyas-Meyer-Oseas Hash function (MMO Hash) is implemented here,
 * along with the CRC-16 functionality.  AES-128 is a prerequisite for
 * MMO Hash but is not provided here, instead please reference the Rijndael
implementation
 * which is in the public domain, and can be obtained here:
 * http://csrc.nist.gov/archive/aes/rijndael/wsdindex.html
 *
 * Author(s): Robert Alexander <rob.alexander@ember.com>
 *

*****************************************************************************
*********/

#include <stdio.h>
#include <stdlib.h>  // for malloc()
#include <stdarg.h>  // for va_list, va_start()
#define __USE_GNU
#include <string.h>  // for strncpy(), strnlen()
#include <assert.h>  // for assert()

// Assume the rijndael headers are on the include path
#include "rijndael-api-fst.h"
#include "rijndael-alg-fst.h"

//---------------------------------------------------------------------------
```

```
// Globals

typedef unsigned char byte;

#define SECURITY_BLOCK_SIZE 16

static const char* usage[] =
        {
        "Usage:  hashing-cli [arguments]",
        "",
        "  This program calculates the CRC-16 and MMO hash of a Zigbee Smart
Energy",
        "  install code.",
        "",
        "REQUIRED ARGUMENTS",
        "",
        "-i, --install-code <code>",
        "   The 6, 8, 12, or 16 byte install code in ASCII hex.",
        "     e.g. 83FED3407A939723A5C639FF",
        "",
        "OPTIONAL ARGUMENTS",
        "",
        "-c, --crc <data>",
        "  The 16-bit CRC in big-endian ASCII hex.  If passed in it will be",
        "  compared against the calculated CRC value.",
        "",
        "-g, --ignore-bad-crc",
        "  Ignore a bad CRC value passed in, use it in the hash anyway.",
        "",
        "-t, --test"
        "  Test the AES-128 and MMO Hash by executing internal test vectors.",
        "  If this is specified the tests are run and nothing else.",
        "",
        "-h, --help",
        "  Print this usage information.",
        "",
```

```
        NULL,
        };

static byte installCode[16];
static int installCodeLength = 0;
static int crc = 0;
static int ignoreBadCrc = 0;

static int DEBUG = 0;

#define TRUE  1
#define FALSE 0

static cipherInstance myCipher;

static int runTestVectors = FALSE;

//-------------------------------------------------------------------------
// Forward Declarations
static int parseArguments(int argc, char* argv[]);
static int parseHexByteString(char* argumentName,char* inputString, byte*
returnData, int maxByteLength);
static byte convertHexCharToByte(unsigned char c);
static void convertByteToHexChars(byte data, char result[2]);
static void printHexBytes(byte* data, int length);
static char* convertBytesToHexString(byte* data, int length);
static void printUsage(void);

#define WARNING 0
#define ERROR 1

static void warn(char *string, ...);
static void error(char *string, ...);
static void debugPrint(char* formatString, ...);

static void aesHash(byte *data, byte totalLength, byte *result);
```

```
static void standAloneEncryptBlock(byte key[SECURITY_BLOCK_SIZE], byte
block[SECURITY_BLOCK_SIZE]);
static unsigned short crc16(char* data_p, unsigned short length);
static void testHashing(void);

//------------------------------------------------------------------------------
// Functions

int main(int argc, char* argv[])
{
byte installCodeAndCrc[18];
byte hashResult[SECURITY_BLOCK_SIZE];
int calculatedCrc;

if (parseArguments(argc, argv))
        return -1;

 if (runTestVectors)
        {
        testHashing();
                return 0;
        }

printf("Install Code: ");
printHexBytes(installCode, installCodeLength);
printf("\n");
calculatedCrc = crc16((char *)installCode, installCodeLength);
printf("Calculated CRC: (>)0x%04X\n", calculatedCrc);
if (crc)
        {
        printf("Passed CRC: (>)0x%04X\n", crc);
        if (crc != calculatedCrc)
                {
                char message[] = "Calculated CRC does not match passed CRC.\n";
                if (ignoreBadCrc)
                        {
```

```
                    warn(message,0);
                    calculatedCrc = crc;
                    }
                else
                    {
                    error(message, 0);
                    return -1;
                    }
                }
        }

    memcpy(installCodeAndCrc, installCode, installCodeLength);
    installCodeAndCrc[installCodeLength] = (byte)(calculatedCrc & 0xFF);
    installCodeAndCrc[installCodeLength+1] = (byte)(calculatedCrc >> 8);

    printf("Byte Concatenation: ");
    printHexBytes(installCodeAndCrc, installCodeLength + 2);
    printf("\n");

    aesHash(installCodeAndCrc, installCodeLength + 2, hashResult);
    printf("Hash Result: ");
    printHexBytes(hashResult, SECURITY_BLOCK_SIZE);
    printf("\n");
    return 0;
    }

    static int parseArguments(int argc, char* argv[])
    {
    int c = 1;
    int optionIndex = 0;
    int installCodeIsSet = TRUE;
    int noArguments = TRUE;
    int printHelp = FALSE;

    while ((c < argc) && (argc > 1))
            {
```

```
noArguments = FALSE;
if ((!strcmp(argv[c],"-c")) || (!strcmp(argv[c],"--crc")))
        {
        c++;
        int crcLength = 0;
        byte crcData[2];
        if (argv[c] == NULL)
                {
                error("The '-c' option requires an argument.", 0);
                return -1;
                }
        crcLength = parseHexByteString("CRC", argv[c], crcData, 2);
        if (crcLength == 0)
                {
                // Error message already printed
                return -1;
                }
        else if (crcLength != 2)
                {
                error("CRC must be 2 bytes.\n", 0);
                return -1;
                }
        crc = ((int)(crcData[0] << 8) | (int)crcData[1]);
        c++;
        }
 else if ((!strcmp(argv[c],"-i")) ||(!strcmp(argv[c],"--install-code")))
        {
        c++;
         if (argv[c] == NULL)
                {
                error("The '-i' option requires an argument.", 0);
                return -1;
                }
        installCodeLength = parseHexByteString("Install Code", argv[c],
installCode, 16);
        if (installCodeLength == 0)
```

```
                        {
                        return -1;
                        }
                    else if (!(installCodeLength == 6|| installCodeLength == 8 ||
installCodeLength == 12 || installCodeLength == 16))
                        {
                        error("Install code must be 6, 8, 12, or 16 bytes in length.\n", 0);
                        return -1;
                        }
                    installCodeIsSet = TRUE;
                    c++;
                    }
                else if ((!strcmp(argv[c],"-g")) ||(!strcmp(argv[c],"--ignore-bad-crc")))
                    {
                    c++;
                    ignoreBadCrc = TRUE;
                    }
                else if ((!strcmp(argv[c],"-t")) ||(!strcmp(argv[c],"--test")))
                    {
                    c++;
                    runTestVectors = TRUE;
                    }
                else if ((!strcmp(argv[c],"-h")) ||(!strcmp(argv[c],"--help")))
                    {
                    c++;
                    printHelp = TRUE;
                    }
                }



    if (noArguments || printHelp)
            {
            printUsage();
            return (-1);
            }
```

```
if (!installCodeIsSet && !runTestVectors)
        {
        error("Must specify '-i' for install code.\n", 0);
        return -1;
        }
return 0;
}

// Transform a string of hex characters into an array of bytes.
// Returns number of bytes parsed on success, or 0 on error.
static int parseHexByteString(char* argumentName, char* inputString, byte*
returnData, int maxByteLength)
{
int i;

int length = strlen(inputString); + (maxByteLength * 2) + 1;
debugPrint("Parsing argument for '%s'\n", argumentName);

if (length > (maxByteLength * 2))
        {
        error("%s cannot be longer than %d\n", argumentName, maxByteLength);
        return 0;
        }

if (length % 2 != 0)
        {
        error("%s must be an even number of hex characters.\n", argumentName);
        return 0;
        }

int byteArrayIndex = 0;
for (i = 0; i < length; i+= 2)
        {
        byte a = convertHexCharToByte(inputString[i]);
        byte b = convertHexCharToByte(inputString[i+1]);
        if (a != 0xFF && b != 0xFF)
```

```
                {
                returnData[byteArrayIndex] = (a << 4) | b;
                byteArrayIndex++;
                }
        else
                {
                error("Invalid hex character '%C' for %s\n", (a == 0xFF? inputString[i]:
inputString[i+1]), argumentName);
                return 0;
                }
        }
return length / 2;
}

static byte convertHexCharToByte(unsigned char c)
{

if (c >= 'A' && c <= 'F') return c - 'A' + 0x0A;
else if (c >= 'a' && c <= 'f') return c - 'a' + 0x0A;
else if (c >= '0' && c <= '9') return c - '0';
return 0xFF;
}

static void convertByteToHexChars(byte data, char result[2])
{
char string[3];// we need 3 bytes due to the '\0' added by
sprintf(string,"%02X",data);
memcpy(result, string, 2); // but we don't want to pass back the '\0'
}

static void warn(char* formatString, ...)
{
va_list vargs;
fprintf(stderr, "Warning: ");
va_start(vargs, formatString);
vfprintf(stderr, formatString, vargs);
```

```
va_end(vargs);
}

static void error(char* formatString, ...)
{
va_list vargs;
fprintf(stderr, "Error: ");
va_start(vargs, formatString);
vfprintf(stderr, formatString, vargs);
va_end(vargs);
}

static void debugPrint(char* formatString, ...)
{
if (DEBUG)
        {
        va_list vargs;
        printf("Debug: ");
        va_start(vargs, formatString);
        vprintf(formatString, vargs);
        va_end(vargs);
        }
}

static void printHexBytes(byte* data, int length)
{
char* string = convertBytesToHexString(data, length);
printf("%s", string);
free(string);
}

static char* convertBytesToHexString(byte* data, int length)
{
char* string = (char*)malloc((length * 2) + 1);
int i;
assert(string);
```

```
for (i = 0; i < length; i++)
        {
        convertByteToHexChars(data[i], &(string[i * 2]));
        }
string[length * 2] = '\0';
return string;
}

static void printUsage(void)
{
int i = 0;
while (usage[i] != NULL)
        {
        printf(usage[i]);
        printf("\n");
        i++;
        }
}

// B.6 Block-cipher-based cryptographic hash function.
//
// This is a method of deriving a hash function from a block encryption
// function.ZigBee uses AES-128 as the encryption function.
//
// First pad the data out to a multiple of 16 bytes with a binary 1 followed by
// all zeros.Append the length of the input (in bits) at the end of the
// padding.
//
// Encrypt the first 16 byte chunk using a NULL key, and XOR with the
// unencrypted data chunk.
// Encrypt the other 16 byte chunks using the previous result as the
// key, and XOR with the unencrypted data chunk.
// Our limited function assumes the following:
// The 'result' value points to an area of memory 16 bytes in length.

static void aesHashNextBlock(byte *block, byte *result)
```

```
{
byte i;

byte key[SECURITY_BLOCK_SIZE];
memcpy(key, result, SECURITY_BLOCK_SIZE);
memcpy(result, block, SECURITY_BLOCK_SIZE);
standAloneEncryptBlock(key, result);

for (i = 0; i < SECURITY_BLOCK_SIZE; i++)
        result[i] ^= block[i];
}

// The hashed data must be in the following form
// ... data ... 0x80 0x00 ... 0x00 LOW_BYTE(dataBits) HIGH_BYTE(dataBits)
// where enough 0x00's are added to make the entire length be a multiple
// of 16.The last two bytes give the size of the original data in bits.

static void aesHash(byte *data, byte totalLength, byte *result)
{
byte temp[SECURITY_BLOCK_SIZE];
byte moreDataLength = totalLength;

memset(result, 0, SECURITY_BLOCK_SIZE);// Hash of 0 is all zeros

for (; SECURITY_BLOCK_SIZE <= moreDataLength; data +=
SECURITY_BLOCK_SIZE, moreDataLength -= SECURITY_BLOCK_SIZE)
        aesHashNextBlock(data, result);

memset(temp, 0, SECURITY_BLOCK_SIZE);
memcpy(temp, data, moreDataLength);
temp[moreDataLength] = 0x80;

 if (SECURITY_BLOCK_SIZE - moreDataLength < 3)
        {
        aesHashNextBlock(temp, result);
        memset(temp, 0, SECURITY_BLOCK_SIZE);
```

```
        }

temp[SECURITY_BLOCK_SIZE - 2] = totalLength >> 5;
temp[SECURITY_BLOCK_SIZE - 1] = totalLength << 3;

 aesHashNextBlock(temp, result);
}

// The single entry point into the implementation specific method
// of performing AES-128.In this case I use the Rijndael implementation.
static void standAloneEncryptBlock(byte key[SECURITY_BLOCK_SIZE], byte
block[SECURITY_BLOCK_SIZE])
{
static int cipherInitialized = FALSE;
byte outBlock[SECURITY_BLOCK_SIZE];
keyInstance myKey;

// Key is expected to be in ASCII hex
char keyMaterial[SECURITY_BLOCK_SIZE * 2];
char* string = convertBytesToHexString(key, SECURITY_BLOCK_SIZE);
strncpy(keyMaterial, string, SECURITY_BLOCK_SIZE * 2);
free(string);

 if (!cipherInitialized)
        {
        // ZigBee defines the IV for AES Encrypt to be zero, per the Spec. section
        // B.1 Block-Cipher-Based Cryptographic Hash Function
        assert(cipherInit(&myCipher, MODE_CBC, NULL)); // Initialization Vector
        cipherInitialized = TRUE;
        }

assert(0 <= makeKey(&myKey, DIR_ENCRYPT, SECURITY_BLOCK_SIZE * 8,
keyMaterial));

assert(0 <= blockEncrypt(&myCipher, &myKey, block, SECURITY_BLOCK_SIZE *
8,outBlock));
```

```
memcpy(block, outBlock, SECURITY_BLOCK_SIZE);
}



// CRC code obtained from:
// http://drdobbs.com/embedded/199904926
/
**************************************************************************
*****
//
// crc16 - generate a ccitt 16 bit cyclic redundancy check (crc)
//
//The code in this module generates the crc for a block of data.
//
**************************************************************************
*****/
/*
//
// The CCITT CRC 16 polynomial is X^16 + X^12 + X^5 + 1.
// In binary, this is the bit pattern 1 0001 0000 0010 0001, and in hex it
//is 0x11021.
// A 17 bit register is simulated by testing the MSB before shifting
//the data, which affords us the luxury of specifiy the polynomial as a
//16 bit value, 0x1021.
// Due to the way in which we process the CRC, the bits of the polynomial
//are stored in reverse order. This makes the polynomial 0x8408.
*/
#define POLY 0x8408

/*
// note: when the crc is included in the message, the valid crc is:
//0xF0B8, before the compliment and byte swap,
//0x0F47, after compliment, before the byte swap,
//0x470F, after the compliment and the byte swap.
*/

int crc_ok = 0x470F;
```

```
/
*************************************************************************
*****
//
// crc16() - generate a 16 bit crc
//
//
// PURPOSE
//This routine generates the 16 bit remainder of a block of
//data using the ccitt polynomial generator.
//
// CALLING SEQUENCE
//crc = crc16(data, len);
//
// PARAMETERS
//data<-- address of start of data block
//len <-- length of data block
//
// RETURNED VALUE
//crc16 value. data is calcuated using the 16 bit ccitt polynomial.
//
// NOTES
//The CRC is preset to all 1's to detect errors involving a loss
//of leading zero's.
//The CRC (a 16 bit value) is generated in LSB MSB order.
//Two ways to verify the integrity of a received message
//or block of data:
//1) Calculate the crc on the data, and compare it to the crc
// calculated previously. The location of the saved crc must be
// known.
/ 2) Append the calculated crc to the end of the data. Now calculate
// the crc of the data and its crc. If the new crc equals the
// value in "crc_ok", the data is valid.
//
// PSEUDO CODE:
```

```
//initialize crc (-1)
//DO WHILE count NE zero
//DO FOR each bit in the data byte, from LSB to MSB
//IF (LSB of crc) EOR (LSB of data)
//crc := (crc / 2) EOR polynomial
//ELSE
//crc := (crc / 2)
//FI
//OD
//OD
//1's compliment and swap bytes in crc
//RETURN crc
//
*************************************************************************
*****/

static unsigned short crc16(char* data_p, unsigned short length)
{
unsigned char i;
unsigned int data;
unsigned long crc;

crc = 0xFFFF;

if (length == 0)
        return ((unsigned short)~crc);

do
        {
        for (i = 0, data = (unsigned int)0xff & *data_p++;i < 8;i++, data >>= 1)
                {
                if ((crc & 0x0001) ^ (data & 0x0001)) crc = (crc >> 1) ^ POLY;
                else crc >>= 1;
                }
        } while (--length);
```

```
crc = ~crc;
return ((unsigned short)crc);
}

static void testHashing(void)
{
printf("Testing AES-128\n");
// These values were found in FIPS 197 Appendix C - Example Vectors,
// C.1 AES-128
byte plainText[] ={ 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF };

byte aesKeyTest[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

byte encryptOutput[] = { 0x69, 0xC4, 0xE0, 0xD8, 0x6A, 0x7B, 0x04, 0x30, 0xD8,
0xCD, 0xB7, 0x80, 0x70, 0xb4, 0xC5, 0x5A };

printf("Plain Text: ");
printHexBytes(plainText, SECURITY_BLOCK_SIZE);
printf("\n");
printf("AES Key:");
printHexBytes(aesKeyTest, SECURITY_BLOCK_SIZE);
printf("\n");
printf("Expected Output: ");
printHexBytes(encryptOutput, SECURITY_BLOCK_SIZE);
printf("\n");

standAloneEncryptBlock(aesKeyTest, plainText);
printf("Actual Output: ");
printHexBytes(plainText, SECURITY_BLOCK_SIZE);
printf("\n");
assert(0 == memcmp(plainText, encryptOutput, SECURITY_BLOCK_SIZE));
printf("\nTesting MMO-Hash\n");
// These values were specified in the ZigBee Spec Rev17,
// C.5 Cryptographic Hash Function (Test Vector Set 1 and 2)
byte testIn0[] = { 0xC0 };
```

```
byte testOut0[] = { 0xAE, 0x3A, 0x10, 0x2A, 0x28, 0xD4, 0x3E, 0xE0, 0xD4, 0xA0,
0x9E, 0x22, 0x78, 0x8B, 0x20, 0x6C };


byte testIn1[] = { 0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9,
0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF };


byte testOut1[] = { 0xA7, 0x97, 0x7E, 0x88, 0xBC, 0x0B, 0x61, 0xE8,  0x21, 0x08,
0x27, 0x10, 0x9A, 0x22, 0x8F, 0x2D };


byte* testInput[] ={ testIn0, testIn1 };
byte testSizes[] = { sizeof(testIn0), sizeof(testIn1) };
byte* testOutput[] = { testOut0, testOut1 };
byte result[SECURITY_BLOCK_SIZE];
 int i;
memset(result, 0, SECURITY_BLOCK_SIZE);


for (i = 0; i < 2; i++)
        {
        printf("Input %d: ", (i+1));
        printHexBytes(testInput[i], testSizes[i]);
        printf("\n");
        aesHash(testInput[i], testSizes[i], result);
        printf("Expected Output: ");
        printHexBytes(testOutput[i], SECURITY_BLOCK_SIZE);
        printf("\n");
        printf("Actual Output: ");
        printHexBytes(result, SECURITY_BLOCK_SIZE);
        printf("\n");
        assert(0 == memcmp(testOutput[i], result, SECURITY_BLOCK_SIZE));
        }
printf("\nAll tests passed.\n");
}[9]
```

9. CCB 981

5.4.8.2 Managing and Initiating Registration Processes

The following sections identify the recommended best practices for managing Authorized devices and the initiating the processes required for device management.

5.4.8.2.1 Best Practices for Tracking Registered Devices

In order to properly track Smart Energy Devices and communicate device registration status to upstream systems, Trust Centers (ESPs) should maintain a list of authorized devices. It's also recommended Trust Centers maintain the following items for each of the registered devices:

1. Client EUI64
2. Client Installation Code
3. Registration Status
4. Time and Date Stamps

Although this information is not exposed through the ZigBee network, device binding is expected to be used to track and understand ZigBee network connectivity.

5.4.8.2.2 Initiating Registration

To initiate the Registration process to authorize a device, the Client device should call the *Initiate Key Establishment* command of the Key Establishment cluster. The Trust Center (ESP) should call the *Initiate Key Establishment Response* command of the Key Establishment cluster in response.

Prior to this activity, the Trust Center enables joining by calling the NLME-PERMIT-JOINING.request primitive. Joining must be managed for an appropriate amount of time but not left on forever. The appropriate amount of time will be dictated by the overall performance of the system and business processes driving the registration and device authorization activities. Be aware Joining has an internal time out within the ZigBee stack, therefore joining may need to be enabled multiple times during the overall Registration and device authorization process.

Please refer to Annex F and [B4] section 5.4.3 for more details around the Joining processes.

Once Registration is completed, the list of authorized devices in the Trust Center should be updated, please refer to sub-clause 5.4.8.2.1.

5.4.8.2.3 Initiating Re-registration

To initiate the re-registration process for a device, the Trust Center (ESP) would invalidate the Link keys for that device and subsequently cause a re-authentication

/ authorization to re-establish Link Keys. The processes required for this activity are:

1. The Trust Center invalidates the Link key by using the APSME-SET primitive.
2. When the Client device detects communication errors due via APS error results or by experiencing multiple re-try failures, both caused by the invalid Link Keys, it starts the processes to validate the following conditions:
   a. The Device validates its still part of the network.
   b. Route discovery processes validate communications paths are still in place.
3. If both conditions are true, the Client device attempts a secure re-join outlined in Re-joining a Secured Network and subsequently refreshes the Link Keys.
4. Re-binding of services take place (if needed).
5. Once Registration is completed, the list of authorized devices in the Trust Center should be updated, please refer to sub-clause 5.4.8.2.1.

5.4.8.2.4 Initiating De-registration

To initiate the de-registration process for a device, which is the process of removing a previously registered device, the Trust Center (ESP) would use the following processes for this activity are:

1. The Trust Center (ESP) invalidates the Link key by using the APSME-SET primitive.
2. The Trust Center (ESP) informs the Client device to leave the network by calling the NLME-LEAVE.request primitive.
3. The Trust Center (ESP) informs any Routers to remove the Client device by calling the APSME-REMOVEDEVICE.request
4. The ESP would unbind any services associated with the Client device by calling the APSME-UNBIND primitive.
5. Once de-registration is completed, the list of authorized devices in the Trust Center should be updated, please refer to sub-clause 5.4.8.2.1.

5.5 Commissioning

Many, if not all of the devices described in this document, will require some form of commissioning, even if the user or installer doesn't see it. This is because, for example, a load control device needs to be bound to some sort of control device in order to perform its function and, even if the required initializations are done at the factory before the device is installed, the required operations are virtually the same as is the outcome.

The ZigBee Alliance has recognized the importance of commissioning and, in particular, the importance of specifications for network and stack commissioning in a multi-vendor environment. Thus, network and stack commissioning procedures are being designed outside the context of any particular profile, where possible, and grouped under the auspices of the Commissioning Tools Task Group (CTTG). This task group is developing a commissioning framework specification [B3].

5.5.1 Forming the Network (Start-up Sequence)

Smart Energy devices must form their own network or join an existing network. The commissioning framework [B3] discusses some of the relevant issues in this procedure.

It is intended that an installer of a Smart Energy device know if the device is forming a network or joining an existing network.

If a device is forming a network there is no user interaction required since the form process can be completed by the device. However there should be some indication to the user or installer that the network has formed properly. The indication can be implemented in a number of ways including blinking indicator lights, colored indicator lights, arrays of indicator lights, text displays, graphic displays, audible indicators such as buzzers and speakers, or through separate means.

If a device is joining an existing network, it will join the network using the processes outlined in sub-clause 5.4. Permit joining will have been turned on due to either installer action or some backchannel mechanism because of user or installer action. It is recommended there be some indication to the user that the device has joined the network successfully. The indication can be implemented in a number of ways including blinking indicator lights, colored indicator lights, arrays of indicator lights, text displays, graphic displays, audible indicators such as buzzers and speakers, etc.

5.5.2 Support for Commissioning Modes

Three different commissioning modes are discussed in [B3]. They are denoted A, E and S-mode.

As discussed above, Smart Energy devices will either automatically form or join a network based on the processes outlined in sub-clause 5.4.

The pre-installation of start up parameters could be done at manufacturing (which is defined as A mode), by an installer tool at the dispatching warehouse, or on site (which would then be S mode). Devices that support this pre-installation must

document the methods used for this preinstallation of parameters.to accomplish this process.

Those devices that will join an existing network must support button pushes or simple documented user interfaces to initiate the joining process. This is in support of E mode commissioning.

5.5.3 Commissioning Documentation Best Practices

To ensure a uniform user experience when commissioning Smart Energy devices, all ZigBee Smart Energy devices are required to provide documentation with their product that explains how to perform device commissioning in using a common language set, i.e., “form network”, “join network”, etc. Please refer to [B3] for further guidance using installation tools and procedures.

5.5.4 Commissioning Procedure for Different Network Types

Depending on the type of network being installed, the commissioning procedures may be slightly different. To ensure interoperability even within these different methods the specific steps are detailed here.

5.5.4.1 Commissioning for Neighborhood Area Network or Sub-metering

Under a neighborhood area network, other meters such as gas or water meters may join electric meters that form a backbone of the network. The process of joining the network is separate from the process for device binding where the device billing information is configured for a particular dwelling unit. It may be desirable to allow the meter to join an adjacent dwelling unit from a network standpoint to ensure proper connectivity. The application level will handle the configuration of the billing information later.

1. There are two methods for joining such a device onto an existing network:
   a. The device is commissioned using a tool with the necessary network and security start up parameters to allow it to rejoin the network as a new device. The device can rejoin any device in the network since it has all the network information.
   b. The network has permit joining turned on by an external tool and the device joins this network and undergoes joining and authentication as any newly joined device.
2. Once joined and authenticated by the security requirements of the existing network, the device is now a member of the neighborhood area network.

3 At the application level, the particular device ID is associated with a particular dwelling unit for billing purposes. This information may be associated at the backend database where the data is collected, or may be sent to the device so it is aware of its association. Note that under this method, devices may route data through devices in adjacent dwelling units that are part of the neighborhood area network.

5.5.4.2 Commissioning for Home Area Network

Under a home area network, the network consists of devices in a particular dwelling unit with one or more co-located metering devices or ESP that provides connectivity to the utility network. Under this scenario, the device within the home may be installed by a trained installer or by a homeowner. The following steps are completed:

1 The Smart Energy network must be informed of the device that is to be joined. This is done through an out of band means which could include a web login, phone call to a service center, or handheld tool. Using this methodology the existing network is made aware of the device ID and security information appropriate for the device (per the Key Establishment Cluster described in Annex C).

2 The Smart Energy network is put into permit joining ON for a period of time.

3 The installer/homeowner is prompted to press a button or complete a menu sequence that tells the device to attempt to join a network.

4 The device joins the network and is authenticated using the appropriate security mechanisms per the Key Establishment Cluster.

5 An indicator is provided for the installer/homeowner indicating the device has joined a network and authenticated properly *or* provides information about improper authentication.

6 The device can now operate normally on the network.

5.6 Public Pricing

It is required that some ZigBee Smart Energy devices respond to requests for public pricing information from devices that are not on the ZigBee Smart Energy network and do not share the same security settings as the ZigBee Smart Energy devices. Only those devices expected to support and communicate this public pricing information must implement this functionality. Devices that support public pricing must support the price cluster within the ZigBee Smart Energy profile. The data from this ZigBee Smart Energy profile is used as the basis for the public pricing broadcast. In other words, the ZigBee Smart Energy devices that receive pricing information over the Smart Energy network transmits it anonymously and

publicly to non-Smart Energy network devices using the anonymous Inter-PAN transmission mechanism outlined in Annex B. Likewise, a Smart Energy device that receives a request for the latest pricing message (formatted as a pricing request from the Price Cluster) will respond with a public and anonymous pricing message.

5.7 Other Smart Energy Profile Requirements and Best Practices

5.7.1 Preferred Channel Usage

When forming a new network, or scanning to join a network, Smart Energy devices should do channel scans using the following preferred channels before scanning the rest of the channels in order to avoid the most commonly used WiFi channels. This is to improve the user experience during installation (quicker joining) and possibly improve bandwidth (on average).

Preferred 2.4 GHz Channels - 11, 14, 15, 19, 20, 24, 25

Preferred 900MHz Channels – Use all available for ZigBee.

5.7.2 Broadcast Policy

Except for public pricing, broadcasts are strongly discouraged for Smart Energy devices. Devices are limited to a maximum broadcast frequency of one broadcast per second and strongly encouraged to exercise broadcasts much less frequently.

5.7.3 Frequency Agility

Frequency Agility would only be officially exercised in a network by a system controller, or higher functioning device (ESP, aggregator, installation tool, etc…). Devices may support frequency agility hooks to be commanded to “go to channel X”. Devices that do not support frequency agility may implement either the NWK rejoin or orphan join feature to find a network that has changed channels.

5.7.4 Key Updates

Smart Energy devices are only required to support ZigBee “residential mode” security or ZigBee PRO “standard mode” with the required use of link keys. All link key updates shall use the Key Establishment Cluster. Sleeping devices that miss key updates can request a new key using the existing link key so there is no problem with sleeping devices missing key updates.

5.7.5 Mirrored Device Capacity - Service Discovery

To aid in discovering Mirrored device capacity prior to key establishment and device authentication it is recommended that the Basic Cluster attribute “PhysicalEnvironment” be used. This would allow a battery based end device to discover if a ESP has capacity to mirror data prior to the process of joining the network in a secure manor, thereby reducing retry attempts. This would also enhance the service discovery of the ZDO Match Descriptor that would be used to determine if an endpoint can request the setup and removal of a mirrored Simple Metering cluster.

An enumerated value of 0x01 would indicate that the device has the capacity to mirror an end device. A value of 0x00 would specify an “Unspecified environment” per the ZDO specification (3.2.2.2.12 of [B2]).

5.8 Coexistence and Interoperability with HA Devices

It is desirable to allow interoperability of HA and Smart Energy devices where practical. However, it is undesirable to publicly share keys during the joining process or share private information over a less secure network. HA devices that only provide functionality for receiving network keys in the clear during a join process cannot be used in a Smart Energy network. These devices can receive public pricing information as described above. HA devices may also be extended with Smart Energy clusters providing they support the use of Link Keys and the Smart Energy security models. If so, they can be certified as HA and Smart Energy capable allowing those devices to operate either in an HA network or a Smart Energy network.

5.9 Device Descriptions

Device descriptions specified in this profile are summarized in Table 5.11 along with their respective Device IDs. The devices are organized according to the end application areas they address. A product that conforms to this specification shall implement at least one of these device descriptions and shall also include the device descriptions corresponding to all applications implemented on the product where a standard device description is specified in this profile. For example, if a product implements both a thermostat and an In-Premise Display, then the thermostat and In-Premise Display device descriptions must both be supported.

This list will be added to in future versions of the profile as new clusters are developed to meet the needs of manufacturers. The reserved values shall not be

used until the profile defines them. Manufacturer-specific device descriptions shall reside on a separate endpoint and use a private profile ID.

Table 5.11 Devices Specified in the Smart Energy Profile

	Device	Device ID
Generic	Range Extender	0x0008
Smart Energy	Energy Service Portal	0x0500
	Metering Device	0x0501
	In-Premise Display	0x0502
	Programmable Communicating Thermostat (PCT)	0x0503
	Load Control Device	0x0504
	Smart Appliance	0x0505
	Prepayment Terminal	0x0506
	Reserved	0x0507 – 0x5FF

5.10 ZigBee Cluster Library (ZCL)

This profile utilizes some of the clusters specified in the ZigBee Cluster Library. The implementation details for each cluster are given in the ZCL specifications. Further specification and clarification is given in this profile where necessary.

The ZCL provides a mechanism for clusters to report changes to the value of various attributes. It also provides commands to configure the reporting parameters. Products shall support the attribute reporting mechanism for supported attributes as specified in the ZCL. The minimum reporting interval specified in the ZCL [B2] shall be set to a value greater than or equal to 0x0001. The maximum reporting interval should be set to 0x0000 by default, and if it is set to a non-zero value it shall be set to a value greater than or equal to 0x003C and greater than the value of the minimum reporting interval. These settings will restrict the attributes from being reported more often than once every second if the attribute is changing quickly and at least once every minute if the attribute does not change for a long time. It is recommended that the minimum reporting interval be set to one minute and the maximum reporting interval be set to a much greater value to avoid unnecessary traffic.

Devices shall use the ZCL default response error handing. Typical examples of this are:

- "When receiving commands that don't have data collected such as Get Scheduled Events, Get Current Price, Get Scheduled Prices, and Get Last Message, devices shall respond using the ZCL default response with a status code of NOT_FOUND.
- "When receiving requests for unsupported commands, devices shall respond using the ZCL default response with a status code of UNSUP_CLUSTER_COMMAND.
- "When receiving malformed commands, devices shall respond using the ZCL default response with a status code of MALFORMED_COMMAND.
- "When receiving requests for accessing unsupported attributes, devices shall respond using the ZCL default response with a status code of UNSUPPORTED_ATTRIBUTE.

Please refer to [B2] for additional status codes support in the ZCL default response.

5.11 Cluster List and IDs

The clusters used in this profile are listed in Table 5.12. The clusters are listed according to the functional domain they belong to in the ZCL and indicate the additional new Smart Energy clusters. The existing corresponding ZCL General cluster identifiers can be found in the ZCL [B2].

The functionality made available by all supported clusters shall be that given in their ZCL specifications except where a device description in this profile includes further specification, clarification or restriction as needed for a particular device.

Most clusters include optional attributes. The application designer must be aware that optional attributes might not be implemented on a particular device. All Smart Energy devices must discover and deal with unsupported attributes on other devices.

It is expected that clusters will continue to be developed in the ZCL that will be useful in this profile. In many cases, new clusters will be organized into new device descriptions that are separate from those currently defined. There may also be situations where it makes sense to add clusters as optional elements of existing device descriptions.

Manufacturer-specific clusters may be added to any device description in this profile as long as they follow the specifications given in the ZCL [B2].

Table 5.12 Clusters Used in the Smart Energy Profile

Functional Domain	Cluster Name	Cluster ID
General	Basic	0x0000
General	Identify	0x0003
General	Alarms	0x0009
General	Time	0x000A
General	Commissioning	0x0015
General	Power Configuration	0x0001
General	Key Establishment	0x0800
Smart Energy	Price	0x0700
Smart Energy	Demand Response and Load Control	0x0701
Smart Energy	Simple Metering	0x0702
Smart Energy	Message	0x0703
Smart Energy	Smart Energy Tunneling (Complex Metering)	0x0704
Smart Energy	Pre-Payment	0x0705

5.11.1 ZCL General Clusters

Except for the Key Establishment Cluster, which is covered in Annex C, please refer to the ZCL Cluster Specification [B2] for the General Cluster descriptions.

5.11.1.1 ZCL Time Cluster and Time Synchronization

The Smart Energy profile requires time synchronization between devices to properly support the coordination of Demand Response/Load Control events, Price changes, and the collection of metered data. In order to simplify the understanding of time, the Smart Energy profile will leverage UTC as the common time base. To this end a new ZCL attribute data type, UTCTime is included and its definition can be found in Annex A.

It is a desire for the processes for synchronizing time to be as network friendly as possible to eliminate excessive traffic. To support this, time accuracy on Client devices shall be within +/-1 minute of the server device (ESP) per 24 hour period. The Client devices shall design a clock accuracy that never requires more than one time synchronization event per 24 hour period. The exception to this is when

devices need to rejoin or re-register on the network. Again, the desire is to keep time synchronization traffic to a minimum.

Further, implementers must be aware that network communication delays will cause minor differences in time between devices. The Smart Energy profile expectations are that this will be a minor issue given the use cases it's fulfilling. It will not nor does it recommend implementers develop an NTP or equivalent scheme to compensate for network delays. These methods are viewed as having the potential to cause excessive network communications.

CHAPTER

6

DEVICE SPECIFICATIONS

6.1 Common Clusters

Support for certain clusters is common to all the devices in this profile. The clusters shown in Table 6.1 shall be supported by all devices in this profile as mandatory or optional according to the designation given here. Individual device descriptions may place further restrictions on support of the optional clusters shown here. If a cluster is not listed as mandatory or optional in the following common table or in the individual device's table that follows, then that cluster shall be prohibited on the Smart Energy endpoint.

Table 6.1 Clusters Common to All Devices

Server Side	**Client Side**
Mandatory	
Basic	*None*
Key Establishment	Key Establishment
Optional	
Clusters with reporting capability	Clusters with reporting capability
Power Configuration	None
Inter-PAN Communication	Inter-PAN Communication
Alarms	None
Commissioning	Commissioning
Identify	*None*[a]
Manufacturer-specific (see sub-clause 6.1.2 for details)	Manufacturer-specific (see sub-clause 6.1.2 for details)

a. CCB 966

6.1.1 Optional Support for Clusters with Reporting Capability

Some clusters support the ability to report changes to the value of particular attributes. These reports are typically received by the client side of the cluster. All devices in this profile may support any cluster that receives attribute reports.

6.1.2 Manufacturer-Specific Clusters

The ZCL provides a range of cluster IDs that are reserved for manufacturer-specific clusters. Manufacturer-specific clusters that conform to the requirements given in the ZCL may be added to any device description specified in this profile.

6.1.3 Cluster Usage Restrictions

None.

6.1.4 Identify Cluster Best Practices

To help aid in locating devices, it's strongly recommended that all devices utilize the Identify Cluster and a visual or audible indicator. In situations in which a device can't supply a visual or audible indicator, the device should include a visible label with the appropriate information to help identify the device.[10]

6.2 Feature and Function Description

Each device must support a certain set of features and functions. Table 6.2 below is used to specify the mandatory and optional features and functions for Smart Energy devices. This chapter contains a description of what must be supported if

10. CCB 966

the feature or function is supported by the device. The mandatory or optional configuration for each device is described in the upcoming chapters:

Table 6.2 Common Features and Functions Configuration for a Smart Energy Device

Device Type/ Feature or function	Join (end devices and routers only)	Form Network (coordinator only)	Restore to Factory Fresh Settings	Pair Devices – (End Device Bind Request)	Bind Manager – (End Device Bind Response - Coordinator only)	Enable Identify Mode	Allow Smart Energy devices to join the Network (routers and coordinators only)
Mandatory/ Optional	M	M	M	O[a]	M	O[b]	M
Device Type/ Feature or function	Service discovery (Match Descriptor Request)	ZDP Bind Response	ZDP Unbind Response	End Device Annce/ device annce	Service Discovery response (Match Descriptor Response)	High Security Supported (ZigBee PRO only)	Inter-PAN Communication
Mandatory/ Optional	O[c]	M	M	M	M	N/A	O

a. CCB 967

b. CCB 967

c. CCB 967

Join (End Devices and Routers):

As described in sub-clauses 5.4 and 5.5.

Form Network (Coordinator):

As described in sub-clauses 5.4 and 5.5.

Allow Others to Join Network (Router and Coordinator Only):

As described in sub-clauses 5.4 and 5.5.

Restore to Factory Fresh Settings:

The Device shall provide a way for the restore Factory Settings.

Pair Devices (End Device Bind Request):

Whenever possible, the device should provide a way for the user to issue an End Device Bind Request.[11]

Bind Manager (End Device Bind Response – Coordinator only):

The coordinator device shall be capable of issuing an End Device Bind Response.

Enable Identify Mode:

Whenever possible, the device should provide a way for the user to enable Identify for 60 seconds.[12]

Service Discovery (Match Descriptor Request):

Whenever possible, the device should provide a way for device to send a match descriptor request, receive match descriptor responses and utilize them for commissioning the device.[13]

ZDP Bind Response:

The device shall be able to receive a ZDP Bind Request and respond correctly with a ZDP Bind Response.

ZDP Unbind Response:

The device shall be able to receive a ZDP Unbind Request and respond correctly with a ZDP Unbind Response.

End Device Annce/Device Annce:

The device shall Send End Device Annce / Send Device upon joining and re-joining a network.

Service Discovery Response:

The Device shall be able to receive a Match descriptor request, and respond with a match descriptor response correctly.

Allow Smart Energy Devices to Join the Network:

The Device shall allow other Smart Energy devices to join the network.

High Security Supported: No

Inter-PAN Communication:

The device may support Inter-PAN Communications as described in Annex B

11. CCB 967
12. CCB 967
13. CCB 967

6.3 Smart Energy Devices

6.3.1 Energy Service Portal

The Energy Service Portal connects the energy supply company communication network to the metering and energy management devices within the home. It routes messages to and from the relevant end points. It may be installed within a meter, thermostat, or In-Premise Display, or may be a standalone device, and it will contain another non-ZigBee communication module (e.g. power-line carrier, RF, GPRS, broadband Internet connection).

6.3.1.1 Supported Clusters

In addition to those specified in Table 6.1, the Energy Service Portal device shall support the clusters listed in Table 6.3. If a cluster is not listed as mandatory or optional in the following table or in the common table, then that cluster shall be prohibited on an ESP device endpoint.

Table 6.3 Clusters Supported by the Energy Service Portal

Server Side	Client Side
Mandatory	
Message	
Price	
Demand Response/Load Control	
Time	
Optional	
Smart Energy Tunneling (Complex Metering)	Smart Energy Tunneling (Complex Metering)
	Price
Simple Metering	Simple Metering
Prepayment	Prepayment

Please Note: Both the Prepayment and Smart Energy Tunneling Cluster definitions are TBD. The information in those sections is not complete and references to them in Table 6.3 should be viewed as place holders until they are completely defined.

6.3.1.2 Supported Features and Functions

The Energy Service Portal device shall have the features and functions listed in Table 6.2.

6.3.2 Metering Device

The Metering end device is a meter (electricity, gas, water, heat, etc.) that is fitted with a ZigBee device. Depending on what is being metered, the device may be capable of immediate (requested) reads or it will autonomously send readings periodically. A Metering end device may also be capable of communicating certain status indicators (e.g. battery low, tamper detected).

6.3.2.1 Supported Clusters

In addition to those specified in Table 6.1, the Metering Device shall support the clusters listed in Table 6.4. If a cluster is not listed as mandatory or optional in the following table or in the common table, then that cluster shall be prohibited on a Metering device endpoint.

Table 6.4 Clusters Supported by the Metering Device

Server Side	Client Side
Mandatory	
Simple Metering	
Optional	
Smart Energy Tunneling (Complex Metering)	
	Time
	Prepayment
	Price
	Message

Please Note: Both the Prepayment and Smart Energy Tunneling Cluster definitions are TBD. The information in those sections is not complete and references to them in Table 6.4 should be viewed as place holders until they are completely defined.

6.3.2.2 Supported Features and Functions

The Metering Device shall have the features and functions listed in Table 6.2.

6.3.3 In-Premise Display Device

The In-Premise Display device will relay energy consumption data to the user by way of a graphical or text display. The display may or may not be an interactive device. At a minimum at least one of the following should be displayed: current energy usage, a history over selectable periods, pricing information, or text messages. As an interactive device, it can be used for returning simple messages for interpretation by the recipient (e.g. “Button A was pressed”).

The display may also show critical pricing information to advise the customer when peaks are due to occur so that they can take appropriate action.

6.3.3.1 Supported Clusters

In addition to those specified in Table 6.1, the In-Premise Display device shall support the clusters listed in Table 6.5. If a cluster is not listed as mandatory or optional in the following table or in the common table, then that cluster shall be prohibited on an In-Premise Display device endpoint.

Table 6.5 Clusters Supported by the In-Premise Display Device

Server Side	Client Side
Mandatory	
Optional	
	Demand Response and Load Control
	Time
	Prepayment
	Price
	Simple Metering
	Message

The device should state that at least one of the optional client clusters (Price, Simple Metering, or Messaging) must be implemented.

Please Note: Both the Prepayment and Smart Energy Tunneling Cluster definitions are TBD. The information in those sections are not complete and references to them in Table 6.5 should be viewed as place holders until they are completely defined.

6.3.3.2 Supported Features and Functions

The In-Premise Display device shall have the features and functions listed in Table 6.2.

6.3.4 Programmable Communicating Thermostat (PCT) Device

The PCT device shall provide the capability to control the premise heating and cooling systems.

6.3.4.1 Supported Clusters

In addition to those specified in Table 6.1, the PCT device shall support the clusters listed in Table 6.6. If a cluster is not listed as mandatory or optional in the following table or in the common table, then that cluster shall be prohibited on a PCT device endpoint.

Table 6.6 Clusters Supported by the PCT

Server Side	Client Side
Mandatory	
	Demand Response and Load Control
	Time
Optional	
	Prepayment
	Price
	Simple Metering
	Message

Please Note: Both the Prepayment and Smart Energy Tunneling Cluster definitions are TBD. The information in those sections is not complete and references to them in Table 6.6 should be viewed as place holders until they are completely defined.

6.3.4.2 Supported Features and Functions

The PCT device shall have the features and functions listed in Table 6.2.

6.3.5 Load Control Device

The Load Control device is capable of receiving Demand Response and Load Control events to manage consumption on a range of devices. Example devices are water heaters, exterior lighting, and pool pumps.

6.3.5.1 Supported Clusters

In addition to those specified in Table 6.1, the Load Control device shall support the clusters listed in Table 6.7.

Table 6.7 Clusters Supported by the Load Control Device

Server Side	Client Side
Mandatory	
	Demand Response and Load Control
	Time
Optional	
	Price

6.3.5.2 Supported Features and Functions

The Load Control Device shall support the features and functions listed in Table 6.2.

6.3.6 Range Extender Device

The Range Extender is a simple device that acts as a router for other devices. The Range Extender device shall not be a ZigBee end device. A product that implements the Range Extender device shall not implement any other devices defined in this profile. This device shall only be used if the product is not intended to have any other application, or if a private application is implemented that has not been addressed by this profile.

6.3.6.1 Supported Clusters

The Range Extender device shall only support the mandatory common clusters listed in Table 6.1.

6.3.6.2 Supported Features and Functions

The Range Extender device shall have the features and functions listed in Table 6.2.

6.3.7 Smart Appliance Device

Smart Appliance devices on the ZigBee network can participate in energy management activities. Examples of these are when Utilities initiate a demand response or pricing event, or the appliance actively informs customers via in-home displays of when or how energy is being used. In the latter case, scenarios include:

- Washer switching to cold water during periods of higher energy costs.
- Washer/Dryer/Oven/Hot Water Heater reporting cycle status.
- Over temperature conditions in Freezers and Refrigerators.

6.3.7.1 Supported Clusters

In addition to those specified in Table 6.1 the Smart Appliance device shall support the clusters listed in Table 6.8. If a cluster is not listed as mandatory or optional in the following table or in the common table, then that cluster shall be prohibited on a Smart Appliance device endpoint.

Table 6.8 Clusters Supported by the Smart Appliance Device

Server Side	Client Side
Mandatory	
	Price
	Time
Optional	
	Demand Response and Load Control
	Message

The device should state that at least one of the optional client clusters (Demand Response and Load Control, Price or Messaging) must be implemented.

6.3.7.2 Supported Features and Functions

The Smart Appliance device shall have the features and functions listed in Table 6.2.

6.3.8 Prepayment Terminal Device

Please Note: The Prepayment Cluster definition is TBD. The information in this section is not complete and should only be used as reference material until the Prepayment cluster is completely defined.

The Prepayment Terminal device will allow utility customers or other users (e.g. sub-metered tenants) to pay for consumption in discrete increments rather than establishing a traditional billing agreement. The Prepayment Terminal device will accept payment (e.g. credit card, code entry), display remaining balances, and alert the user of a balance approaching zero, and may perform some or all of the other functions described in *In-Premise Display.*

6.3.8.1 Supported Clusters

In addition to those specified in Table 6.1, the Prepayment Terminal device shall support the clusters listed in Table 6.9. If a cluster is not listed as mandatory or optional in the following table or in the common table, then that cluster shall be prohibited on a Prepayment Terminal device endpoint.

Table 6.9 Clusters Supported by the Prepayment Terminal Device

Server Side	Client Side
Mandatory	
	Price
	Time
Prepayment	Prepayment
Optional	
	Demand Response and Load Control
	Simple Metering
	Message

6.3.8.2 Supported Features and Functions

The Prepayment Terminal device shall have the features and functions listed in Table 6.2.

This page intentionally blank

ANNEX

A

CANDIDATE ZCL MATERIAL FOR USE WITH THIS PROFILE

The candidate material in this annex, when approved, will be merged into the Foundation document of the ZigBee Cluster Library (ZCL) by the Cluster Library Development Board.

A.1 New Data Types

This section defines new ZCL data types needed for interoperability with Smart Energy-based ZigBee devices and functions.

A.2 Definition of New Types

The following material in this subsection is proposed for inclusion in the Data Types section (section 8.2) of the Foundation document [B2].

A.2.1 New Time Data Type

The new Time data type being requested is listed in Table A.1.

Table A.1 Additional Time Cluster Data Type

Type Class	Data Type ID	Data Type	Length of Data (Octets)	Invalid Number	Analog / Discrete
Time	0xe2	UTCTime	4	0xffffffff	A
	0xe3 – 0xe7	Reserved	-	-	-

A.2.1.1 UTCTime

UTCTime is an unsigned 32 bit value representing the number of seconds since 0 hours, 0 minutes, 0 seconds, on the 1st of January, 2000 UTC. It reflects and defines the data type used in the ZCL Time Server attribute labeled as *Time*. The value that represents an invalid value of this type is 0xffffffffff.

A.2.2 New Unsigned Integer Data Type

The new Unsigned Integers data types being requested are listed in Table A.2

Table A.2 New Unsigned Integer Data Types

Type Class	Data Type ID	Data Type	Length of Data (Octets)	Invalid Number	Analog / Discrete
Unsigned Integer	0x24	Unsigned 40 Bit Integer	5	0xffffffffff	A
	0x25	Unsigned 48 Bit Integer	6	0xffffffffffff	A
	0x26 – 0x27	Reserved	-	-	-

A.2.2.1 Unsigned 40 Bit Integer

This type represents an unsigned integer with a decimal range of 0 to 2^{40}-1. The value that represents an invalid value of this type is 0xffffffffff.

A.2.2.2 Unsigned 48 Bit Integer

This type represents an unsigned integer with a decimal range of 0 to 2^{48}-1. The value that represents an invalid value of this type is 0xffffffffffff.

ANNEX

B

INTER-PAN TRANSMISSION MECHANISM

B.1 Scope and Purpose

This annex defines a mechanism whereby ZigBee devices can perform limited, insecure, and possibly anonymous exchanges of information with devices in their local neighborhood without having to form or join the same ZigBee network. The mandate for this feature comes from the Energy Management / Smart Energy market requirement to send pricing information to very low cost devices. The particular data exchange required by the Smart Energy Application Profile is the request for anonymous public energy pricing information. The typical example is the extremely low cost “Refrigerator Magnet” device that simply informs customers of current energy costs through some visual method (LCD, LED’s, etc.).

The intended destination for the mechanism described here is not the ZigBee specification [B1], but the relevant application profile documents for applications that make use of the feature – in particular, the Smart Energy Profile Specification.

The material used to create Annex B is derived from [B7].

B.2 General Description

B.2.1 What Inter-PAN Transmission Does

A schematic view of the how inter-PAN transmission in a ZigBee context works is shown in Figure B.1.

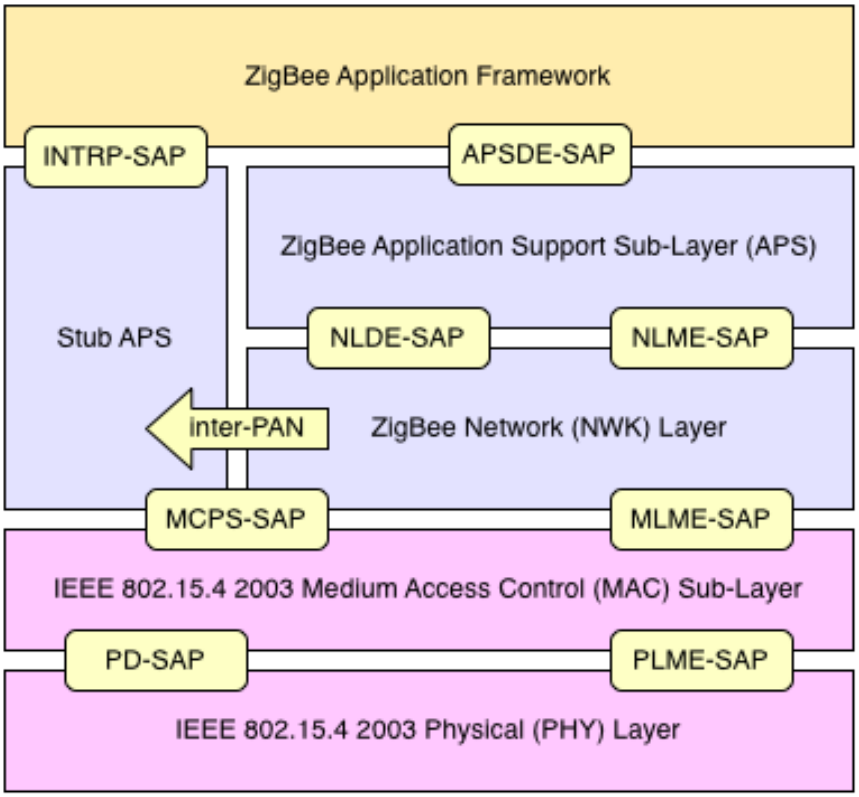

Figure B.1 ZigBee Stack with Stub APS

Inter-PAN data exchanges are handled by a special “stub” of the Application Support Sub-Layer, which is accessible through a special Service Access Point (SAP), the INTRP-SAP, parallel to the normal APSDE-SAP. The stub APS performs just enough processing to pass application data frames to the MAC for transmission and to pass inter-PAN application frames from the MAC to the application on receipt.

The Inter-Pan data exchange architecture does not support simultaneous execution by multiple application entities. Within a device, only one application entity shall use the Inter-Pan communications mechanisms.

B.3 Service Specification

The INTRP-SAP is a data service comprising three primitives.

- INTRP-DATA.request – Provides a mechanism for a sending device to request transmission of an Inter-Pan message.

- INTRP-DATA.confirm – Provides a mechanism for a sending device to understand the status of a previous request to send an Inter-Pan message.
- INTRP-DATA.indication – Provides a mechanism for a identifying and conveying an Inter-Pan message received from a sending device.

B.3.1 The INTRP-DATA.request Primitive

The INTRP-DATA.request primitive allows an application entity to request data transmission via the stub APS.

B.3.1.1 Semantics of the Service Primitive

The primitive interface is as follows:

INTRP-DATA.request	{
	SrcAddrMode
	DstAddrMode
	DstPANId
	DstAddress
	ProfileId
	ClusterId
	ASDULength
	ASDU
	ASDUHandle
	}

Parameters of the primitive appear in Table B.1.

Table B.1 Parameters of the INTRP-DATA.request

Name	Type	Valid Range	Description
SrcAddrMode	Integer	0x03	The addressing mode for the source address used in this primitive. This parameter shall only reference the use of the 64-bit extended address: 0x03 = 64-bit extended address
DstAddrMode	Integer	0x01 – 0x03	The addressing mode for the destination address used in this primitive. This parameter can take one of the values from the following list: 0x01 = 16-bit group address 0x02 = 16-bit NWK address, normally the broadcast address 0xffff 0x03 = 64-bit extended address
DstPANID	16-bit PAN Id	0x0000 – 0xffff	The 16-bit PAN identifier of the entity or entities to which the ASDU is being transferred or the broadcast PANId 0xffff.
DstAddress	16-bit or 64-bit address	As specified by the AddrMode parameter	The address of the entity or entities to which the ASDU is being transferred.
ProfileId	Integer	0x0000 – 0xffff	The identifier of the application profile for which this frame is intended.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the cluster, within the profile specified by the ProfileId parameter, which defines the application semantics of the ASDU.
ASDULength	Integer	0x00 – (*aMaxMACFrameSize* - 9)	The number of octets in the ASDU to be transmitted.
ASDU	Set of octets	-	The set of octets forming the ASDU to be transmitted.
ASDUHandle	Integer	0x00 – 0xff	An integer handle associated with the ASDU to be transmitted.

B.3.1.2 When Generated

This primitive is generated by the local application entity when it wishes to address a frame to one or more peer application entities residing on neighboring devices with which it does not share a network association.

B.3.1.3 Effect on Receipt

On receipt of the INTRP-DATA.request primitive by the stub APS, the stub APS will construct and transmit a frame containing the given ASDU and other parameters using the MCPS-DATA.request primitive of the MAC sub-layer, as described in sub-clause sub-clause B.5.1, and, once the corresponding MCPS-DATA.confirm primitive is received, Generate the INTRP-DATA.confirm primitive with a status value reflecting the status value returned by the MAC.

B.3.2 The INTRP-DATA.confirm Primitive

The INTRP-DATA.confirm primitive allows the stub APS to inform the application entity about the status of a data request.

B.3.2.1 Semantics of the Service Primitive

The primitive interface is as follows:

INTRP-DATA.confirm	{
	ASDUHandle
	Status
	}

Parameters of the primitive appear in Table B.2.

Table B.2 Parameters of the INTRP-DATA.confirm

Name	Type	Valid Range	Description
ASDUHandle	Integer	0x00 – 0xff	An integer handle associated with the transmitted frame.
Status	Enumeration	Any Status value returned by the MAC	The status of the ASDU transmission corresponding to ASDUHandle as returned by the MAC.

B.3.2.2 When Generated

This primitive is generated by the stub APS on a ZigBee device and passed to the application in response to the receipt of a MCPS-DATA.confirm primitive that is a confirmation of a previous MCPS-DATA.request issued by the stub APS.

B.3.2.3 Effect on Receipt

As a result of the receipt of this primitive, the application is informed of the results of an attempt to send a frame via the stub APS.

B.3.3 The INTRP-DATA.indication Primitive

The INTRP-DATA.indication primitive allows the stub APS to inform the next higher layer that it has received a frame that was transmitted via the stub APS on another device.

B.3.3.1 Semantics of the Service Primitive

The primitive interface is as follows:

INTRP-DATA.indication	{
	SrcAddrMode
	SrcPANId
	SrcAddress
	DstAddrMode
	DstPANId
	DstAddress
	ProfileId
	ClusterId
	ASDULength
	ASDU
	LinkQuality
	}

Parameters of the primitive appear in Table B.3.

Table B.3 Parameters of the INTRP-DATA.indication

Name	Type	Valid Range	Description
SrcAddrMode	Integer	0x03	The addressing mode for the source address used in this primitive. This parameter shall only reference the use of the 64-bit extended address: 0x03 = 64-bit extended address
SrcPANId	16-bit PAN Id	0x0000 – 0xffff	The 16-bit PAN identifier of the entity from which the ASDU is being transferred.
SrcAddress	64-bit address	As specified by the SrcAddrMode parameter	The device address of the entity from which the ASDU is being transferred.
DstAddrMode	Integer	0x01 – 0x03	The addressing mode for the destination address used in this primitive. This parameter can take one of the values from the following list: 0x01 = 16-bit group address 0x02 = 16-bit NWK address, normally the broadcast address 0xffff 0x03 = 64-bit extended address
DstPANID	16-bit PAN Id	0x0000 – 0xffff	The 16-bit PAN identifier of the entity or entities to which the ASDU is being transferred or the broadcast PAN ID 0xffff.
DstAddress	16-bit or 64-bit address	As specified by the DstAddrMode parameter	The address of the entity or entities to which the ASDU is being transferred.
ProfileId	Integer	0x0000 – 0xffff	The identifier of the application profile for which this frame is intended.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the cluster, within the profile specified by the ProfileId parameter, which defines the application semantics of the ASDU.

Table B.3 Parameters of the INTRP-DATA.indication

ASDULength	Integer	0x00 – (*aMaxMACFrameSize* - 9)	The number of octets in the ASDU to be transmitted.
ASDU	Set of octets	-	The set of octets forming the ASDU to be transmitted.
LinkQuality	Integer	0x00 – 0xff	The link quality observed during the reception of the ASDU.

B.3.3.2 When Generated

This primitive is generated and passed to the application in the event of the receipt, by the stub APS, of a MCPS-DATA.indication primitive from the MAC sub-layer, containing a frame that was generated by the stub APS of a peer ZigBee device, and that was intended for the receiving device.

B.3.3.3 Effect on Receipt

Upon receipt of this primitive the application is informed of the receipt of an application frame transmitted, via the stub APS, by a peer device and intended for the receiving device.

B.3.4 Qualifying and Testing of Inter-Pan Messages

Certification and application level testing shall ensure both the sending and receiving devices correctly react and understand the INTRP-DATA.request and INTRP-DATA.indication primitives.

B.4 Frame Formats

The birds-eye view of a normal ZigBee frame is as shown in Figure B.2.

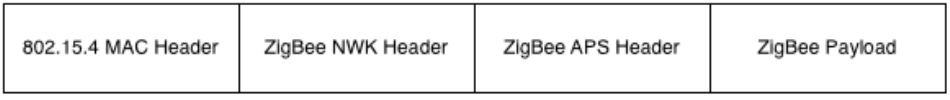

Figure B.2 Normal ZigBee Frame

Briefly, the frame contains the familiar headers controlling the operation of the MAC sub-layer, the NWK layer and the APS. Following these, there is a payload, formatted as specified in [B2].

Since most of the information contained in the NWK and APS headers is not relevant for inter-PAN transmission, the inter-PAN frame, shown in Figure B.3, contains only a stub of the NWK header the APS header, which provide the information required by the stub APS shown in Figure B.4 to do its job.

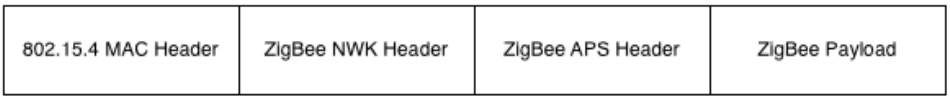

Figure B.3 Inter-PAN ZigBee Frame

Octets: 2
NWK frame control

Figure B.4 Stub NWK Header Format

The format of the frame control field of the stub NWK header is formatted as shown in Figure B.5.

Bits: 0-1	**2-5**	**6-15**
Frame type	Protocol version	Remaining sub-fields == 0

Figure B.5 Format of the NWK Frame Control Field

The sub-fields of the NWK frame control field are as follows:

- The frame type sub-field shall have a value of 0b11, which is a reserved frame type with respect to the [B4].
- The value protocol version sub-field shall reflect the protocol version of the ZigBee stack as described in [B4].

All other sub-fields shall have a value of 0.

The format of the stub APS header is shown in Figure B.6.

Octets: 1	0/2	2	2
APS frame control	Group address	Cluster identifier	Profile identifier
	Addressing fields		

Figure B.6 Stub APS Header Format

The stub APS header contains only 4 fields totaling a maximum of 7 octets in length.

The APS frame control field shall be 1 octet in length and is identical in format to the frame control field of the general APDU frame in [B4] (see Figure B.7).

Bits: 0-1	2-3	4	5	6	7
Frame type	Delivery Mode	Reserved	Security	ACK request	Extended Header Present

Figure B.7 Format of the APS Frame Control Field

The fields of the frame control field have the following values:

- The frame type sub-field shall have a value of 0b11, which is a reserved frame type with respect to the [B4].
- The delivery mode sub-field may have a value of 0b00, indicating unicast, 0b10, indicating broadcast or 0b11 indicating group addressing.
- Security is never enabled for Inter-Pan transmissions. This sub-field shall be a value of 0.
- The ACK request sub-field shall have a value of 0, indicating no ACK request. No APS ACKs are to be used with Inter-Pan transmissions.
- The extended header present sub-field shall always have a value of 0, indicating no extended header.

The optional group address shall be present if and only if the delivery mode field has a value of 0x0b11. If present if shall contain the 16-bit identifier of the group to which the frame is addressed.

The cluster identifier field is 2 octets in length and specifies the identifier of the cluster to which the frame relates and which shall be made available for filtering and interpretation of messages at each device that takes delivery of the frame.

The profile identifier is two octets in length and specifies the ZigBee profile identifier for which the frame is intended and shall be used during the filtering of messages at each device that takes delivery of the frame.

B.5 Frame Processing

Assuming the INTRP-SAP described above, frames transmitted using the stub APS are processed as described here.

B.5.1 Inter-PAN Transmission

On receipt of the INTRP-DATA.request primitive, the stub APS shall construct a stub APS frame. The header of the stub APS frame shall contain a NWK and an APS frame control field as described in clause clause B.4, a cluster identifier field equal to the value of the ClusterId parameter of the INTRP-DATA.request and a profile identifier field equal to the value of the ProfileId parameter. If the DstAddrMode parameter of the INTRP-DATA.request has a value of 0x01, indicating group addressing, then the APS header shall also contain a group address field with a value corresponding to the value of the DstAddress parameter. The payload of the stub APS frame shall contain the data payload to be transmitted.

The stub APS frame will then be transmitted using the MCPS-DATA.request primitive of the MAC sub-layer with key primitive parameters set as follows:

- The value of the SrcAddrMode parameter of the MCPS-DATA.request shall always be set to a value of three, indicating the use of the 64-bit extended address.
- The SrcPANId parameter shall be equal to the value of the *macPANID* attribute of the MAC PIB.
- The SrcAddr parameter shall always be equal to the value of the MAC sub-layer constant *aExtendedAddress*.
- If the DstAddrMode parameter of the INTRP-DATA.request primitive has a value of 0x01 then the DstADdrMode parameter of the MCPS-DATA.request shall have a value of 0x02. Otherwise, the DstAddrMode parameter of the MCPS-DATA.request shall reflect the value of the DstAddrMode parameter of the INTRP-DATA.request.
- The DstPANId parameter shall have the value given by the DstPANID parameter of the INTRP-DATA.request primitive.
- If the DstAddrMode parameter of the INTRP-DATA.request has a value of 0x01, indicating group addressing, then the value of the DstAddr parameter of

the MCPS-DATA.request shall be the broadcast address 0xffff. Otherwise, value of the DstAddr parameter shall reflect the value of the DstAddress parameter of the INTRP-DATA.request primitive.

- The MsduLength parameter shall be the length, in octets, of the stub APS frame.
- The Msdu parameter shall be the stub APS frame itself.
- If the transmission is a unicast then the value of the TxOptions parameter shall be 0x01, indicating a request for acknowledgement. Otherwise, the TxOptions parameter shall have a value of 0x00, indicating no options.

On receipt of the MCPS-DATA.confirm primitive form the MAC sub-layer, the stub APS will invoke the transmit confirmation function with a status reflecting the status returned by the MAC.

B.5.2 Inter-PAN Reception

On receipt of the MCPS-DATA.indication primitive from the MAC sub-layer, the receiving entity – in case of a ZigBee device this is normally the NWK layer – shall determine whether the frame should be passed to the stub APS or processed as specified in [B4]. For a frame that is to be processed by the stub APS, the non-varying sub-fields of both the NWK frame control field and the APS frame control field must be set exactly as described above.

If the delivery mode sub-field of the APS frame control field of the stub APS header has a value of 0b11, indicating group addressing, then, if the device implements group addressing, the value of the group address field shall be checked against the NWK layer group table, and, if the received value is not present in the table, the frame shall be discarded with no further processing or action.

On receipt of a frame for processing, the stub APS shall generate an INTRP-DATA.indication with parameter values as follows:

- The value of the SrcAddrMode parameter of the INTRP-DATA.indication shall always be set to a value of three, indicating the use of the 64-bit extended address
- The value of the SrcPANId parameter shall reflect that of the SrcPANId parameter of the MCPS-DATA.indication.
- The SrcAddress parameter of the INTRP-DATA.indication shall always reflect the value of a 64-bit extended address.
- Values for the DstAddrMode parameter shall be one of:

  - 0x03, if the DstAddrMode parameter of the INTRP-DATA.indication has a value of 0x03.
  - 0x02, if the DstAddrMode parameter of the INTRP-DATA.indication has a value of 0x02
- The value of the DstPANId parameter of the INTRP-DATA.indication shall reflect the value of the DstPANId parameter of the MCPS-DATA.indication.
- If the DstAddrMode parameter of the INTRP-DATA.indication has a value of 0x01, indicating group addressing then the DstAddress parameter of the INTRP-DATA.indication shall reflect the value of the group address field of the stub APS header. Otherwise, the value of the DstAddress parameter of the INTRP-DATA.indication shall reflect the value of the DstAddr parameter of the MCPS-DATA.indication.
- The value of the ProfileId parameter shall be the same as the value of the profile identifier field of the stub APS header.
- The value of the ClusterId parameter shall be the same as the value of the cluster identifier field of the stub APS header.
- The ASDULength field shall contain the number of octets in the stub APS frame payload.
- The ASDU shall be the stub APS payload itself.
- The value of the LinkQuality parameter shall reflect the value of the mpduLinkQuality parameter of the MCPS-DATA.indication.

B.6 Usage Scenario

Figure B.8 shows a typical usage scenario for inter-PAN communication. In this Smart Energy-oriented scenario, the Home Area Network (HAN) device is on the left with the APL, NWK and MAC shown as separate sequences. A ZigBee electric meter or Energy Service Portal (ESP) is also shown along with a “foreign”, i.e. non-ZigBee, device.

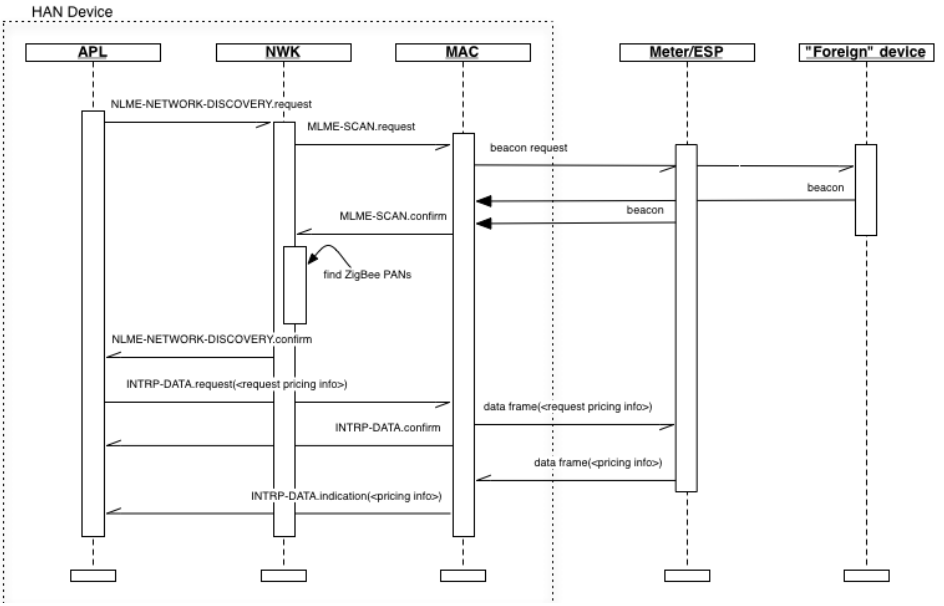

Figure B.8 Inter-PAN Typical Usage

The first task of the HAN device in this scenario is to discover devices in the area that are capable of publishing pricing information. It could do this using an inter-PAN broadcast, i.e. a broadcast employing both the broadcast address and the broadcast PAN ID, but in doing this it runs the risk of confusing the non-ZigBee "foreign" device. As an alternative, the HAN device uses standard ZigBee network discovery (see [B4]) in order to find ZigBee PANs.

Once at least one ZigBee PAN is discovered, the HAN device sends a request for public pricing information using the INTRP-DATA SAP. Typically, the first time this request is sent, it will be sent as a broadcast to each discovered ZigBee PAN. Receiving devices that implement the INTRP-DATA SAP will process it and, if any such device is able to respond, it will respond directly to the requestor. After receiving at least one response the requestor may store the PAN ID and device address of one or more responders so that it may query them directly in future.

ANNEX

C

KEY ESTABLISHMENT CLUSTER

The candidate material in this annex, when approved, will be merged into the Foundation document of the ZigBee Cluster Library (ZCL) by the Cluster Library Development Board.

C.1 Scope and Purpose

This Annex specifies a cluster, which contains commands and attributes necessary for managing secure communication between ZigBee devices.

This Annex should be used in conjunction with the ZigBee Cluster Library, Foundation Specification (see [B2]), which gives an overview of the library and specifies the frame formats and general commands used therein.

This version is specifically for inclusion in the Smart Energy profile. The document which originates from [B6] will continue to be developed in a backward-compatible manner as a more general secure communication cluster for ZigBee applications as a whole.

C.2 General Description

C.2.1 Introduction

As previously stated, this document describes a cluster for managing secure communication in ZigBee. The cluster is for Key Establishment.

C.2.2 Network Security

The Key Establishment Cluster has been designed to be used where the underlying network security cannot be trusted. As such, no information that is confidential information will be transported.

C.2.3 Key Establishment

To allow integrity and confidentiality of data passed between devices, cryptographic schemes need to be deployed. The cryptographic scheme deployed in the ZigBee Specification for frame integrity and confidentiality is based upon a variant of the AES-CCM described in [B12] called AES-CCM*. This relies on the existence of secret keying material shared between the involved devices. There are methods to distribute this secret keying material in a trusted manner. However, these methods are generally not scalable or communication may be required with a trusted key allocation party over an insecure medium. This leads to the requirement for automated key establishment schemes to overcome these problems.

Key establishment schemes can either be effected using either a key agreement scheme or a key transport scheme. The key establishment scheme described in this document uses a key agreement scheme, therefore key transport schemes will not be considered further in this document.

A key agreement scheme is where both parties contribute to the shared secret and therefore the secret keying material to be established is not sent directly; rather, information is exchanged between both parties that allows each party to derive the secret keying material. Key agreement schemes may use either symmetric key or asymmetric key (public key) techniques. The party that begins a key agreement scheme is called the initiator, and the other party is called the responder.

Key establishment using key agreement involves an initiator and a responder and four steps:

1 Establishment of a trust relationship

2 Exchange of ephemeral data

3 Use of this ephemeral data to derive secret keying material using key agreement

4 Confirmation of the secret keying material.

There are two basic types of key establishment which can be implemented:

- Symmetric Key Key Establishment
- Public Key Key Establishment

C.2.4 Symmetric Key Key Establishment

Symmetric Key Key Establishment (SKKE) is based upon establishing a link key based on a shared secret (master key). If the knowledge of the shared secret is compromised, the established link key can also be compromised. If the master key is publicly known or is set to a default value, it is known as Unprotected Key Establishment (UKE). SKKE is the key establishment method used in the ZigBee specification therefore it will not be considered any further.

C.2.5 Public Key Key Establishment

Public Key Key Establishment (PKKE) is based upon establishing a link key based on shared static and ephemeral public keys. As the public keys do not require any secrecy, the established link key cannot be compromised by knowledge of them.

As a device's static public key is used as part of the link key creation, it can either be transported independently to the device's identity where binding between the two is assumed, or it can be transported as part of a implicit certificate signed by a Certificate Authority, which provides authentication of the binding between the device's identity and its public key as part of the key establishment process. This is called Certificate-Based Key Establishment (CBKE) and is discussed in more detail in sub-clause C.4.2.

CBKE provides the most comprehensive form of Key Establishment and therefore will be the method specified in this cluster.

The purpose of the key agreement scheme as described in this document is to produce shared secret keying material which can be subsequently used by devices using AES-CCM* the cryptographic scheme deployed in the ZigBee Specification or for any proprietary security mechanism implemented by the application.

C.2.6 General Exchange

The following diagram shows an overview of the general exchange which takes place between initiator and responder to perform key establishment.

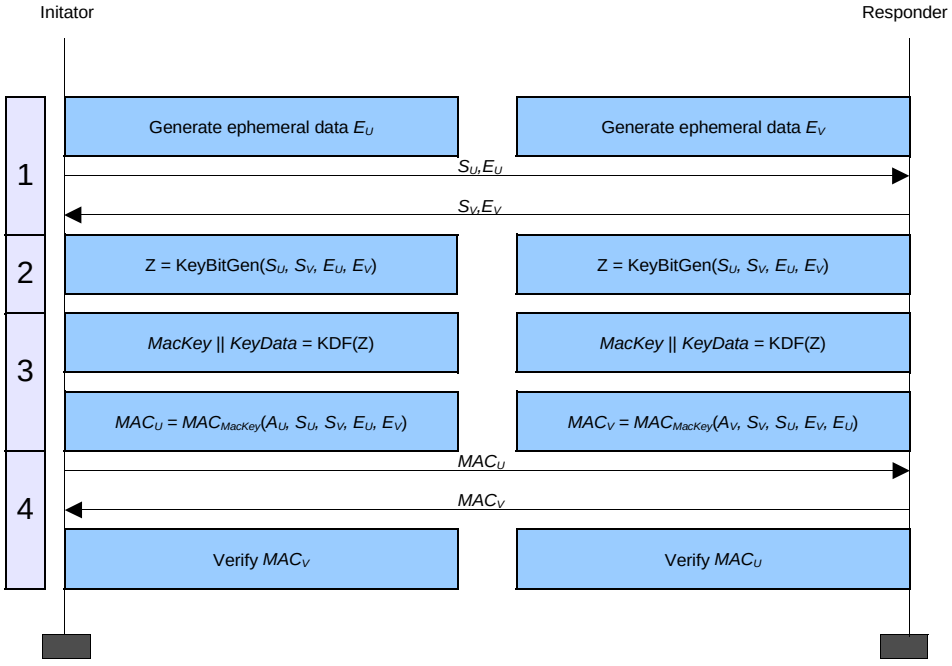

Figure C.1 Overview of General Exchange

The functions are as follows:

1 Exchange Static and Ephemeral Data

2 Generate Key Bitstream

3 Derive MAC key and Key Data

4 Confirm Key using MAC

The functions shown in the diagram depend on the Key Establishment mechanism.

C.2.6.1 Exchange Static and Ephemeral Data

Figure C.1 shows static data S_U and S_V. For PKKE schemes, this represents a combination of the 64-bit device address [B8] and the device's static public key. The identities are needed by the MAC scheme and the static public keys are needed by the key agreement scheme.

Figure C.1 also shows ephemeral data E_U and E_V. For PKKE schemes, this represents the public key of a randomly generated key pair.

The static and ephemeral data S_U and E_U are sent to *V* and the static and ephemeral data S_V and E_V and are sent to *U*.

C.2.6.2 Generate Key Bitstream

Figure C.1 shows the KeyBitGen function for generating the key bitstream. The function's four parameters are the identifiers and the ephemeral data for both devices. This ensures the same key is generated at both ends.

For PKKE schemes, this is the ECMQV key agreement schemes specified in Section 6.2 of SEC1 [B15]. The static data S_U represents the static public key $Q_{1,U}$ of party *U*, the static data S_V represents the static public key $Q_{1,V}$ of party *V*, the ephemeral data E_U represents the ephemeral public key $Q_{2,U}$ of party *U* and the ephemeral data E_V represents the ephemeral public key $Q_{2,V}$ of party *V*.

C.2.6.3 Derive MAC Key and Key Data

Figure C.1 shows the KDF (KeyDerivation Function) for generating the MAC Key and key data. The MAC Key is used with a keyed hash message authentication function to generate a MAC and the key data is shared secret, e.g. the link key itself required for frame protection.

For PKKE schemes, this is the key derivation function is as specified in Section 3.6.1 of SEC1 [B15]. Note there is no *SharedInfo* parameter of the referenced KDF, i.e. it is a null octet string of length 0.

Figure C.1 also shows generation of the MAC using the MAC Key derived using the KDF using a message comprised of both static data S_U and S_V and ephemeral data E_U and E_V plus an additional component *A* which is different for initiator and responder.

For PKKE schemes, this is the MAC scheme specified in section 3.7 of SEC1 [B15]. The MAC in the reference is the keyed hash function for message authentication specified in sub-clause C.4.2.2.6 and the message *M* is a concatenation of the identity (the 64-bit device address [B8]) of *U*, the identity of *V* and point-compressed octet-string representations of the ephemeral public keys of parties *U* and *V*. The order of concatenation depends on whether it is the initiator or responder. The additional component *A* is the single octet 02_{16} for the initiator and 03_{16} for the responder.

C.2.6.4 Confirm Key Using MAC

Figure C.1 shows MACs MAC_U and MAC_V.

The MAC MAC_U is sent to *V* and the MAC MAC_V is sent to *U*. U and V both calculates the corresponding MAC and compares it with the data received.

C.3 Cluster List

The clusters specified in this document are listed in Table C.1.

For our purposes, any device that implements the client side of this cluster may be considered the initiator of the secure communication transaction.

Table C.1 Clusters Specified for the Secure Communication Functional Domain

Cluster Name	Description
Key Establishment	Attributes and commands for establishing a shared secret between two ZigBee devices.

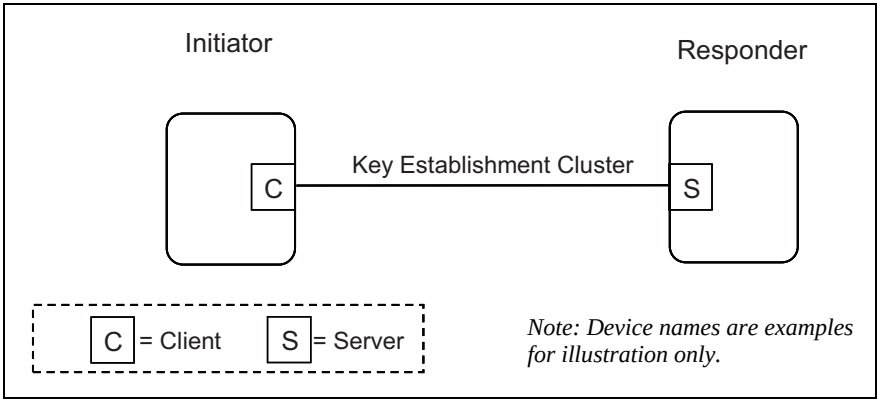

Figure C.2 Typical Usage of the Key Establishment Cluster

C.3.1 Key Establishment Cluster

C.3.1.1 Overview

This cluster provides attributes and commands to perform mutual authentication and establish keys between two ZigBee devices. Figure C.3 depicts a diagram of a successful key establishment negotiation.

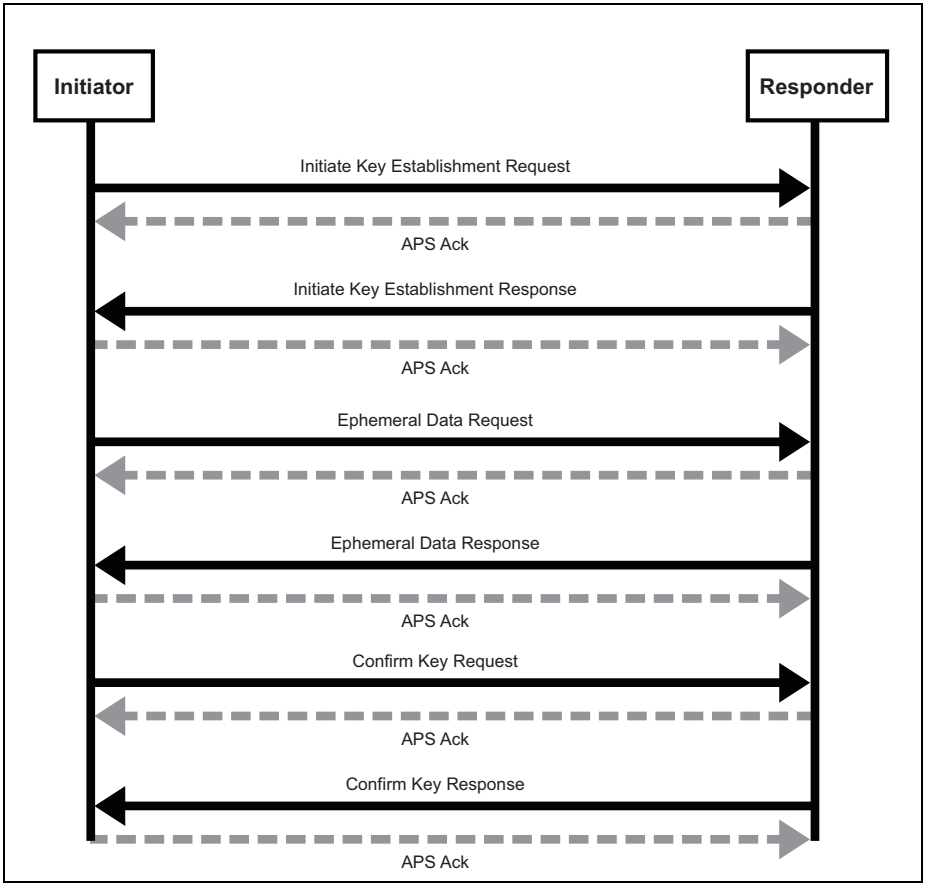

Figure C.3 Key Establishment Command Exchange

As depicted above, all Key Establishment messages should be sent with APS retries enabled. A failure to receive an ACK in a timely manner can be seen as a failure of key establishment. No Terminate Key Establishment should be sent to the partner of device that has timed out the operation.

The initiator can initiate the key establishment with any active endpoint on the responder device that supports the key establishment cluster. The endpoint can be either preconfigured or discovered, for example, by using ZDO Match-Desc-req. A link key successfully established using key establishment is valid for all endpoints on a particular device. The responder shall respond to the initiator using the source endpoint of the initiator's messages as the destination endpoint of the responder's messages.

It is expected that the time it takes to perform the various cryptographic computations of the key establishment cluster may vary greatly based on the

device. Therefore rather than set static timeouts, the Initiate Key Establishment Request and Response messages will contain approximate values for how long the device will take to generate the ephemeral data and how long the device will take to generate confirm key message.

A device performing key establishment can use this information in order to choose a reasonable timeout for its partner during those operations. The timeout should also take into consideration the time it takes for a message to traverse the network including APS retries. A minimum transmission time of 2 seconds is recommended.

For the Initiate Key Establishment Response message, it is recommended the initiator wait at least 2 seconds before timing out the operation. It is not expected that generating an Initiate Key Establishment Response will take significant time compared to generating the Ephemeral Data and Confirm Key messages.

C.3.1.2 Server

C.3.1.2.1 Dependencies

The Key Establishment server cluster has no dependencies.

C.3.1.2.2 Attributes

For convenience, the attributes defined in this specification are arranged into sets of related attributes; each set can contain up to 16 attributes. Attribute identifiers are encoded such that the most significant three nibbles specify the attribute set and the least significant nibble specifies the attribute within the set. The currently defined attribute sets are listed in Table C.2.

Table C.2 Key Establishment Attribute Sets

Attribute Set Identifier	Description
0x000	Information
0x001 – 0xfff	Reserved

C.3.1.2.2.1 Information

The Information attribute set contains the attributes summarized in Table C.3.

Table C.3 Key Establishment Attribute Sets

Identifier	Name	Type	Range	Access	Default	Mandatory/ Optional
0x0000	KeyEstablishmentSuite	16-bit Enumeration	0x0000 - 0xFFFF	Read only	0x0000	M

C.3.1.2.2.1.1 KeyEstablishmentSuite Attribute

The *KeyEstablishmentSuite* attribute is 16-bits in length and specifies all the cryptographic schemes for key establishment on the device. A device shall set the corresponding bit to 1 for every cryptographic scheme that is supports. All other cryptographic schemes and reserved bits shall be set to 0.

Table C.4 Values of the *KeyEstablishmentSuite* Attribute

Bits	Description
0	Certificate-based Key Establishment (CBKE-ECMQV)
1-15	Reserved

C.3.1.2.3 Commands Received

The server side of the key establishment cluster is capable of receiving the commands listed in Table C.5.

Table C.5 Received Command IDs for the Key Establishment Cluster Server

Command Identifier Field Value	Description	Mandatory/ Optional
0x00	Initiate Key Establishment Request	M
0x01	Ephemeral Data Request	M
0x02	Confirm Key Data Request	M
0x03	Terminate Key Establishment	M
0x04 – 0xFF	Reserved	

C.3.1.2.3.1 Initiate Key Establishment Request Command

The Initiate Key Establishment Request command allows a device to initiate key establishment with another device. The sender will transmit its identity information and key establishment protocol information to the receiving device.

C.3.1.2.3.1.1 Payload Format

The Initiate command payload shall be formatted as illustrated in Figure C.4.

Octets	**2**	**1**	**1**	**48**
Data Type	16-bit BitMap	Unsigned 8-bit Integer	Unsigned 8-bit Integer	Octets (non-ZCL Data Type)
Field Name	Key Establishment suite	Ephemeral Data Generate Time	Confirm Key Generate Time	Identity (IDU)

Figure C.4 Format of the Initiate Key Establishment Request Command Payload

Key Establishment Suite: This will be the type of Key Establishment that the initiator is requesting for the Key Establishment Cluster. For CBKE-ECMQV this will be 0x0001.

Ephemeral Data Generate Time[14]**:** This value indicates approximately how long the initiator device will take in seconds to generate the Ephemeral Data Request command. The valid range is 0x00 to 0xFE.

Confirm Key Generate Time[15]**:** This value indicates approximately how long the initiator device will take in seconds to generate the Confirm Key Request command. The valid range is 0x00 to 0xFE.

Identity field: For *KeyEstablishmentSuite* = 0x0001 (CBKE), the identity field shall be the block of octets containing the implicit certificate CERTU as specified in sub-clause C.4.2.

C.3.1.2.3.1.2 Effect on Receipt

If the device does not currently have the resources to respond to a key establishment request it shall send a Terminate Key Establishment command with the result value set to NO_RESOURCES and the Wait Time field shall be set to an

14. CCB 993
15. CCB 993

approximation of the time that must pass before the device will have the resources to process a new Key Establishment Request.

If the device can process this request, it shall check the Issuer field of the device's implicit certificate. If the Issuer field does contain a value that corresponds to a known Certificate Authority, the device shall send a Terminate Key Establishment command with the result set to UNKNOWN_ISSUER.

If the device accepts the request it shall send an Initiate Key Establishment Response command containing its own identity information.

C.3.1.2.3.2 Confirm Key Request Command

The Confirm Key command allows the initiator sending device to confirm the key established with the responder receiving device based on performing a cryptographic hash using part of the generated keying material and the identities and ephemeral data of both parties.

C.3.1.2.3.2.1 Payload Format

The Confirm Key command payload shall be formatted as illustrated in Figure C.5.

Octets	**16**
Data Type	Octet string
Field Name	Secure Message Authentication Code (*MACU*)

Figure C.5 Format of the Confirm Key Request Command Payload

Secure Message Authentication Code field: The Secure Message Authentication Code field shall be the octet-string representation of *MACU* as specified in sub-clause C.4.2.

C.3.1.2.3.2.2 Effect on Receipt

If the device is not currently in the middle of negotiating Key Establishment with the sending device when it receives this message, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE. If the device is in the middle of Key Establishment with the sender but did not receive this message in response to an Ephemeral Data Response command, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE.

On receipt of the Confirm Key Request command the responder device shall compare the received MACU value with its own reconstructed version of MACU.

If the two match the responder shall send back MACV by generating an appropriate Confirm Key Response command. If the two do not match, the responder shall send back a Terminate Key Establishment with a result of BAD _KEY_CONFIRM and terminate the key establishment.

C.3.1.2.3.3 Terminate Key Establishment Command

The Terminate Key Establishment command may be sent by either the initiator or responder to indicate a failure in the key establishment exchange.

C.3.1.2.3.3.1 Payload Format

The Terminate Key Establishment command payload shall be formatted as illustrated in Figure C.6.

Octets	1	1	2
Data Type	8-bit Enumeration	Unsigned 8-bit Integer	16-bit BitMap
Field Name	Status Code	Wait Time	KeyEstablishmentSuite

Figure C.6 Format of the Terminate Key Establishment Command Payload

Status Field: The Status field shall be one of the error codes in Table C.6.

Table C.6 Terminate Key Establishment Command Status Field

Enumeration	Value	Description
	0x00	Reserved
UNKNOWN_ISSUER	0x01	The Issuer field within the key establishment partner's certificate is unknown to the sending device, and it has terminated the key establishment.
BAD_KEY_CONFIRM	0x02	The device could not confirm that it shares the same key with the corresponding device and has terminated the key establishment.
BAD_MESSAGE	0x03	The device received a bad message from the corresponding device (e.g. message with bad data, an out of sequence number, or a message with a bad format) and has terminated the key establishment.

Table C.6 Terminate Key Establishment Command Status Field (Continued)

Enumeration	Value	Description
NO_RESOURCES	0x04	The device does not currently have the internal resources necessary to perform key establishment and has terminated the exchange.
UNSUPPORTED_SUITE	0x05	The device does not support the specified key establishment suite in the partner's Initiate Key Establishment message.
	0x06 - 0xFF	Reserved

Wait Time: This value indicates the minimum amount of time in seconds the initiator device should wait before trying to initiate key establishment again. The valid range is 0x00 to 0xFE.

KeyEstablishmentSuite: This value will be set the value of the KeyEstablishmentSuite attribute. It indicates the list of key exchange methods that the device supports.

C.3.1.2.3.3.2 Effect on Receipt

On receipt of the Terminate Key Establishment command the device shall terminate key establishment with the sender. If the device receives a status of BAD_MESSAGE or NO_RESOURCES it shall wait at least the time specified in the Wait Time field before trying to re-initiate Key Establishment with the device.

If the device receives a status of UNKNOWN_SUITE it should examine the KeyEstablishmentSuite field to determine if another suite can be used that is supported by the partner device. It may re-initiate key establishment using that one of the supported suites after waiting the amount of time specified in the Wait Time field. If the device does not support any of the types in the KeyEstablishmentSuite field, it should not attempt key establishment again with that device.

If the device receives a status of UNKNOWN_ISSUER or BAD_KEY_CONFIRM the device should not attempt key establishment again with the device, as it is unlikely that another attempt will be successful.

C.3.1.2.3.4 Ephemeral Data Request Command

The Ephemeral Data Request command allows a device to communicate its ephemeral data to another device and request that the device send back its own ephemeral data.

C.3.1.2.3.4.1 Payload Format

Octets	22
Data Type	Octets (non-ZCL Data Type)
Field Name	Ephemeral Data (QEU)

Figure C.7 Format of the Ephemeral Data Request Command Payload

C.3.1.2.3.4.2 Effect on Receipt

If the device is not currently in the middle of negotiating Key Establishment with the sending device when it receives this message, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE. If the device is in the middle of Key Establishment with the sender but did not receive this message in response to an Initiate Key Establishment Response command, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE.

If the device can process the request it shall respond by generating its own ephemeral data and sending an Ephemeral Data Response command containing that value.

C.3.1.2.4 Commands Generated

The server generates the commands detailed in sub-clause C.3.1.3.3, as well as those used for reading and writing attributes.

C.3.1.3 Client

C.3.1.3.1 Dependencies

The Key Establishment client cluster has no dependencies.

C.3.1.3.2 Attributes

For convenience, the attributes defined in this specification are arranged into sets of related attributes; each set can contain up to 16 attributes. Attribute identifiers are encoded such that the most significant three nibbles specify the attribute set and the least significant nibble specifies the attribute within the set. The currently defined attribute sets are listed in Table C.7.

Table C.7 Key Establishment Attribute Sets

Attribute Set Identifier	Description
0x000	Information
0x001 – 0xfff	Reserved

C.3.1.3.2.1 Information

The Information attribute set contains the attributes summarized in Table C.8.

Table C.8 Attributes of the Information Attribute Set

Identifier	Name	Type	Range	Access	Default	Mandatory/ Optional
0x0000	KeyEstablishmentSuite	16-bit Enumeration	0x0000 – 0xFFFF	Read only	0x0000	M

C.3.1.3.2.1.1 KeyEstablishmentSuite Attribute

The *KeyEstablishmentSuite* attribute is 16-bits in length and specifies all the cryptographic schemes for key establishment on the device. A device shall set the corresponding bit to 1 for every cryptographic scheme that is supports. All other cryptographic schemes and reserved bits shall be set to 0. This attribute shall be set to one of the non-reserved values listed in Table C.9.

Table C.9 Values of the KeyEstablishmentSuite Attribute

KeyEstablishmentSuite	Description
0	Certificate-based Key Establishment (CBKE - ECMQV)
1-15	Reserved

C.3.1.3.3 Commands Received

The client side of the Key Establishment cluster is capable of receiving the commands listed in Table C.10.

Table C.10 Received Command IDs for the Key Establishment Cluster Client

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Initiate Key Establishment Response	M
0x01	Ephemeral Data Response	M
0x02	Confirm Key Data Response	M
0x03	Terminate Key Establishment	M
0x04 - 0xFF	Reserved	

C.3.1.3.3.1 Initiate Key Establishment Response Command

The Initiate Key Establishment Response command allows a device to respond to a device requesting the initiation of key establishment with it. The sender will transmit its identity information and key establishment protocol information to the receiving device.

C.3.1.3.3.1.1 Payload Format

The Initiate response command payload shall be formatted as illustrated in Figure C.8.

Octets	2	1	1	48
Data Type	16-bit BitMap	Unsigned 8-bit Integer	Unsigned 8-bit Integer	Octets (non-ZCL Data Type)
Field Name	Requested Key Establishment suite	Ephemeral Data Generate Time	Confirm Key Generate Time	Identity (IDU)

Figure C.8 Format of the Initiate Key Establishment Response Command Payload

[16]**Requested Key Establishment Suite:** This will be the type of *KeyEstablishmentSuite* that the initiator has requested be used for the key establishment exchange. The device shall set a single bit in the bitmask indicating the requested suite, all other bits shall be set to zero.

16. CCB 993

Ephemeral Data Generate Time: This value indicates approximately how long in seconds the responder device takes to generate the Ephemeral Data Response message. The valid range is 0x00 to 0xFE.

Confirm Key Generate Time: This value indicates approximately how long the responder device will take in seconds to generate the Confirm Key Response message. The valid range is 0x00 to 0xFE.

Identity field: For *KeyEstablishmentSuite* = 0x0001 (CBKE), the identity field shall be the block of Octets containing the implicit certificate CERTU as specified in sub-clause C.4.2.

C.3.1.3.3.1.2 Effect on Receipt

If the device is not currently in the middle of negotiating Key Establishment with the sending device when it receives this message, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE. If the device is in the middle of Key Establishment with the sender but did not receive this message in response to an Initiate Key Establishment Request command, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE.

On receipt of this command the device shall check the Issuer field of the device's implicit certificate. If the Issuer field does contain a value that corresponds to a known Certificate Authority, the device shall send a Terminate Key Establishment command with the status value set to UNKNOWN_ISSUER. If the device does not currently have the resources to respond to a key establishment request it shall send a Terminate Key Establishment command with the status value set to NO_RESOURCES and the Wait Time field shall be set to an approximation of the time that must pass before the device has the resources to process the request.

If the device accepts the response it shall send an Ephemeral Data Request command.

C.3.1.3.3.2 Ephemeral Data Response Command

The Ephemeral Data Response command allows a device to communicate its ephemeral data to another device that previously requested it.

C.3.1.3.3.2.1 Payload Format

Octets	**22**
Data Type	Octets (non-ZCL Data Type)
Field Name	Ephemeral Data (QEV)

Figure C.9 Format of the Ephemeral Data Response Command Payload

C.3.1.3.3.2.2 Effect on Receipt

If the device is not currently in the middle of negotiating Key Establishment with the sending device when it receives this message, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE. If the device is in the middle of Key Establishment with the sender but did not receive this message in response to an Ephemeral Data Request command, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE.

On receipt of this command if the device can handle the request it shall perform key generation, key derivation, and MAC generation. If successful it shall generate an appropriate Confirm Key Request command, otherwise it shall generate a Terminate Key Establishment with a result value of NO_RESOURCES.

C.3.1.3.3.3 Confirm Key Response Command

The Confirm Key Response command allows the responder to verify the initiator has derived the same secret key. This is done by sending the initiator a cryptographic hash generated using the keying material and the identities and ephemeral data of both parties.

C.3.1.3.3.3.1 Payload Format

The Confirm Key response command payload shall be formatted as illustrated in Figure C.10.

Octets	**0/16**
Data Type	Octets (non-ZCL Data Type)
Field Name	Secure Message Authentication Code (*MACV*)

Figure C.10 Format of the Confirm Key Response Command Payload

Secure Message Authentication Code field: The Secure Message Authentication Code field shall be the octet-string representation of *MACV* as specified in sub-clause C.4.2.

C.3.1.3.3.3.2 Effect on Receipt

If the device is not currently in the middle of negotiating Key Establishment with the sending device when it receives this message, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE. If the device is in the middle of Key Establishment with the sender but did not receive this message in response to an Confirm Key Request command, it shall send back a Terminate Key Establishment message with a result of BAD_MESSAGE.

On receipt of the Confirm Key Response command the initiator device shall compare the received MACV value with its own reconstructed version of the MACV. If the two match then the initiator can consider the key establishment process to be successful. If the two do not match, the initiator should send a Terminate Key Establishment command with a result of BAD_KEY_CONFIRM.

C.3.1.3.3.4 Terminate Key Establishment Command

The Terminate Key Establishment command may be sent by either the initiator or responder to indicate a failure in the key establishment exchange.

C.3.1.3.3.4.1 Payload Format

Octets	1	1	2
Data Type	8-bit Enumeration	Unsigned 8-bit Integer	16-bit BitMap
Field Name	Status Code	Wait Time	KeyEstablishmentSuite

Figure C.11 Format of the Terminate Key Establishment Command Payload

Status field: The Status field shall be one of the following error codes.

Table C.11 Terminate Key Establishment Command Status Field

Enumeration	Value	Description
	0x00	Reserved
UNKNOWN_ISSUER	0x01	The Issuer field within the key establishment partner's certificate is unknown to the sending device, and it has terminated the key establishment.
BAD_KEY_CONFIRM	0x02	The device could not confirm that it shares the same key with the corresponding device and has terminated the key establishment.
BAD_MESSAGE	0x03	The device received a bad message from the corresponding device (e.g. message with bad data, an out of sequence number, or a message with a bad format) and has terminated the key establishment.

Table C.11 Terminate Key Establishment Command Status Field

Enumeration	Value	Description
NO_RESOURCES	0x04	The device does not currently have the internal resources necessary to perform key establishment and has terminated the exchange.
UNSUPPORTED_SUITE	0x05	The device does not support the specified key establishment suite in the partner's Initiate Key Establishment message.
	0x06 - 0xFF	Reserved

Wait Time: This value indicates the minimum amount of time in seconds the initiator device should wait before trying to initiate key establishment again. The valid range is 0x00 to 0xFE.

KeyEstablishmentSuite: This value will be set the value of the *KeyEstablishmentSuite* attribute. It indicates the list of key exchange methods that the device supports.

C.3.1.3.3.4.2 Effect on Receipt

On receipt of the Terminate Key Establishment command the device shall terminate key establishment with the sender. If the device receives a status of BAD_MESSAGE or NO_RESOURCES it shall wait at least the time specified in the Wait Time field before trying to re-initiate Key Establishment with the device.

If the device receives a status of UNKNOWN_SUITE it should examine the *KeyEstablishmentSuite* field to determine if another suite can be used that is supported by the partner device. It may re-initiate key establishment using that one of the supported suites after waiting the amount of time specified in the Wait Time field. If the device does not support any of the types in the *KeyEstablishmentSuite* field, it should not attempt key establishment again with that device.

If the device receives a status of UNKNOWN_ISSUER or BAD_KEY_CONFIRM the device should not attempt key establishment again with the device, as it is unlikely that another attempt will be successful.

C.3.1.3.4 Commands Generated

The client generates the commands detailed in sub-clause C.3.1.2.3, as well as those used for reading and writing attributes.

C.4 Application Implementation

C.4.1 Network Security for Smart Energy Networks

The underlying network security for Smart Energy networks is assumed to be ZigBee Standard security using pre-configured link keys.

A temporary link key for a joining device is produced by performing the cryptographic hash function on a random number assigned to the joining device (e.g. serial number) and the device identifier, which is the device's 64-bit IEEE address [B8].

The joining device's assigned random number is then conveyed to the utility via an out-of-band mechanism (e.g. telephone call, or web site registration). The utility then commissions the energy service portal (ESP) at the premises where the joining device is by installing the temporary link key at the ESP on the back channel.

When the joining device powers up, it will also create a temporary link key as above and therefore at the time of joining both the joining device and the ESP have the same temporary link key, which can be used to transport the network key securely to the joining device.

At this point, the device will be considered joined and authenticated as far as network security is concerned. The secure communication cluster can now be invoked to replace the temporary link key with a more secure link key based on public key cryptography.

C.4.2 Certificate-Based Key Establishment

The Certificate-Based Key-Establishment (CBKE) solution uses public-key technology with digital certificates and root keys. Each device has a private key and a digital certificate that is signed by a Certificate Authority (CA).

The digital certificate includes:

- Reconstruction data for the device's public key
- The device's extended 64-bit IEEE address
- Profile specific information (e.g., the device class, network id, object type, validity date, etc.).

Certificates provide a mechanism for cryptographically binding a public key to a device's identity and characteristics.

Trust for a CBKE solution is established by provisioning a CA root key and a digital certificate to each device. A CA root key is the public key paired with the

CA's private key. A CA uses its private key to sign digital certificates and the CA root key is used to verify these signatures. The trustworthiness of a public key is confirmed by verifying the CA's signature of the digital certificate. Certificates can be issued either by the device manufacturer, the device distributor, or the end customer. For example, in practical situations, the CA may be a computer (with appropriate key management software) that is kept physically secure at the end customer's facility or by a third-party.

At the end of successful completion of the CBKE protocol the following security services are offered:

- Both devices share a secret link key
- Implicit Key Authentication: Both devices know with whom they share this link key.
- Key Confirmation: Each device knows that the other device actually has computed the key correctly
- No Unilateral Key Control: No device has complete control over the shared link key that is established.
- Perfect Forward Secrecy: if the public key gets compromised none of future and past communications are exposed
- Known Key Security resilience: Each shared link key created per session is unique

C.4.2.1 Notation and Representation

C.4.2.1.1 Strings and String Operations

A string is a sequence of symbols over a specific set (e.g., the binary alphabet {0,1} or the set of all octets). The length of a string is the number of symbols it contains (over the same alphabet). The right-concatenation of two strings *x* and *y* of length *m* and *n* respectively (notation: *x* || *y*), is the string *z* of length *m*+*n* that coincides with *x* on its leftmost *m* symbols and with *y* on its rightmost *n* symbols. An octet is a bit string of length 8.

C.4.2.1.2 Integers and their Representation

Throughout this specification, the representation of integers as bit strings or octet strings shall be fixed. All integers shall be represented as binary strings in most-significant-bit first order and as octet strings in most-significant-octet first order. This representation conforms to the convention in Section 2.3 of SEC1 [B15].

C.4.2.1.3 Entities

Throughout this specification, each entity shall be a DEV and shall be uniquely identified by its 64-bit IEEE device address [B8]. The parameter *entlen* shall have the integer value 64.

C.4.2.2 Cryptographic Building Blocks

The following cryptographic primitives and data elements are defined for use with the CBKE protocol specified in this document.

C.4.2.2.1 Elliptic-Curve Domain Parameters

The elliptic curve domain parameters used in this specification shall be those for the curve “ansit163k1” as specified in section 3.4.1 of SEC2 [B16].

All elliptic-curve points (and operations hereon) used in this specification shall be (performed) on this curve.

C.4.2.2.2 Elliptic-Curve Point Representation

All elliptic-curve points shall be represented as point compressed octet strings as specified in sections 2.3.3 and 2.3.4 of SEC1 [B15]. Thus, each elliptic-curve point can be represented in 22 bytes.

C.4.2.2.3 Elliptic-Curve Key Pair

An elliptic-curve-key pair consists of an integer *d* and a point *Q* on the curve determined by multiplying the generating point *G* of the curve by this integer (i.e., *Q=dG*) as specified in section 3.2.1 of SEC1 [B15]. Here, *Q* is called the public key, whereas *d* is called the private key; the pair (*d, Q*) is called the key pair. Each private key shall be represented as specified in section 2.3.7 of SEC1 [B15]. Each public key shall be represented as defined in sub-clause C.4.2.1.2 of this document.

C.4.2.2.4 ECC Implicit Certificates

The exact format of the 48-byte implicit certificate IC_U used with CBKE scheme shall be specified as follows:

IC_U = PublicReconstrKey || Subject || Issuer || ProfileAttributeData

Where,

1. *PublicReconstrKey*: the 22-byte representation of the public-key reconstruction data *BEU* as specified in the implicit certificate generation protocol, which is an elliptic-curve point as specified in sub-clause C.4.2.2.2 (see SEC4 [B15]);
2. *Subject*: the 8-byte identifier of the entity *U* that is bound to the public-key reconstruction data *BEU* during execution of the implicit certificate generation protocol (i.e., the extended, 64-bit IEEE 802.15.4 address [B8] of the device that purportedly owns the private key corresponding to the public key that can be reconstructed with *PublicReconstrKey*);
3. *Issuer***:** the 8-byte identifier of the CA that creates the implicit certificate during the execution of the implicit certificate generation protocol (the so-called Certificate Authority).

4 *ProfileAttributeData***:** the 10-byte sequence of octets that can be used by a ZigBee profile for any purpose. The first two bytes of this sequence is reserved as a profile identifier, which must be defined by another ZigBee standard.

5 The string I_U as specified in Step 6 of the actions of the CA in the implicit certificate generation protocol (see section SEC4 [B17]) shall be the concatenation of the *Subject*, *Issuer*, and *ProfileAttributeData*:

I_U = *Subject* || *Issuer* || *ProfileAttributeData*

C.4.2.2.5 Block-Cipher

The block-cipher used in this specification shall be the Advanced Encryption Standard AES-128, as specified in FIPS Pub 197 [B13]. This block-cipher has a key size that is equal to the block size, in bits, i.e., *keylen*= 128.

C.4.2.2.6 Cryptographic Hash Function

The cryptographic hash function used in this specification shall be the blockcipher based cryptographic hash function specified in Annex B.6 in [B4], with the following instantiations:

1 Each entity shall use the block-cipher E as specified in sub-clause B.1.1 in [B4].

2 All integers and octets shall be represented as specified in sub-clause C.4.2.1.

The Matyas-Meyer-Oseas hash function (specified in Annex B.6 in [B4]) has a message digest size *hashlen* that is equal to the block size, in bits, of the established blockcipher.

C.4.2.2.7 Keyed Hash Function for Message Authentication

The keyed hash message authentication code (HMAC) used in this specification shall be HMAC, as specified in the FIPS Pub 198 [B14] with the following instantiations:

1 Each entity shall use the cryptographic hash *H* function as specified in sub-clause C.4.2.2.6;

2 The block size *B* shall have the integer value 16 (this block size specifies the length of the data integrity key, in bytes, that is used by the keyed hash function, i.e., it uses a 128-bit data integrity key). This is also *MacKeyLen*, the length of *MacKey*.

3 The output size *HMAClen* of the HMAC function shall have the same integer value as the message digest parameter *hashlen* as specified in sub-clause C.4.2.2.6.

C.4.2.2.8 Derived Shared Secret

The derived shared secret *KeyData* is the output of the key establishment. *KeyData* shall have length *KeyDataLen* of 128 bits.

C.4.2.3 Certificate-Based Key-Establishment

The CBKE method is used when the authenticity of both parties involved has not been established and where implicit authentication of both parties is required prior to key agreement.

The CBKE protocol has an identical structure to the PKKE protocol, except that implicit certificates are used rather than manual certificates. The implicit certificate protocol used with CBKE shall be the implicit certificate scheme with associated implicit certificate generation scheme and implicit certificate processing transformation as specified in SEC4 [B15], with the following instantiations:

1. Each entity shall be a DEV;
2. Each entity's identifier shall be its 64-bit device address [B8]; the parameter *entlen* shall have the integer value 64;
3. Each entity shall use the cryptographic hash function as specified in sub-clause C.4.2.2.6;

The following additional information shall have been unambiguously established between devices operating the implicit certificate scheme:

1. Each entity shall have obtained information regarding the infrastructure that will be used for the operation of the implicit certificate scheme – including a certificate format and certificate generation and processing rules (see SEC4 [B15]);
2. Each entity shall have access to an authentic copy of the elliptic-curve public keys of one or more certificate authorities that act as CA for the implicit certificate scheme (SEC4 [B15]).

The methods by which this information is to be established are outside the scope of this standard.

The methods used during the CBKE protocol are described below. The parameters used by these methods are described in Table C.12.

Table C.12 Parameters Used by Methods of the CBKE Protocol

Parameter	Size (Octets)	Description
CERTU	48	The initiator device's implicit certificate used to transfer the initiator device's public key (denoted $Q_{1,U}$ in the Elliptic Curve MQV scheme in SEC1 [B15]) and the initiator device's identity.
CERTV	48	The responder device's implicit certificate used to transfer the responder device's public key (denoted $Q_{1,V}$ in the Elliptic Curve MQV scheme in SEC1 [B15]) and the responder device's identity.
QEU	22	The ephemeral public key generated by the initiator device (denoted $Q_{2,U}$ in the Elliptic Curve MQV scheme in SEC1 [B15]).
QEV	22	The ephemeral public key generated by the responder device (denoted Q_{2V} in the Elliptic Curve MQV scheme in SEC1 [B15]).
MACU	16	The secure message authentication code generated by the initiator device (where the message M is $(02_{16} \| ID_U \| ID_V \| QEU \| QEV)$ and ID_U and ID_V are the initiator and responder device entities respectively as specified in sub-clause C.4.2.2.3 and *QEU* and *QEV* are the point-compressed elliptic curve points representing the ephemeral public keys of the initiator and responder respectively as specified in sub-clause C.4.2.2.2. See also section 3.7 of SEC1 [B15]).
MACV	16	The secure message authentication code generated by the responder device (where the message M is $(03_{16} \| ID_V \| ID_U \| QEV \| QEU)$ and ID_V and ID_U are the responder and initiator device entities respectively as specified in sub-clause C.4.2.2.3 and *QEV* and *QEU* are the point-compressed elliptic curve points representing the ephemeral public keys of the responder and initiator respectively as specified in sub-clause C.4.2.2.3. See also section 3.7 of SEC1 [B15]).

C.4.2.3.1 Exchange Ephemeral Data

C.4.2.3.1.1 Initiator

The initiator device's implicit certificate *CERTU* and a newly generated ephemeral public key *QEU* are transferred to the responder device using the Initiate Key Establishment command via the Key Establishment Cluster Client.

C.4.2.3.1.2 Responder

The responder device's implicit certificate *CERTV* and a newly generated ephemeral public key *QEV* are transferred to the initiator device using the Initiate Key Establishment response command via the Key Establishment Cluster Server.

C.4.2.3.2 Validate Implicit Certificates

C.4.2.3.2.1 Initiator

The initiator device's Key Establishment Cluster Client processes the Initiate Key Establishment response command. The initiator device examines *CERTV* (formatted as IC_V as described in sub-clause C.4.2.2.4), confirms that the *Subject* identifier is the purported owner of the certificate, and runs the certificate processing steps described in section SEC4 [B16].

C.4.2.3.2.2 Responder

The responder device's Key Establishment Cluster Server processes the Initiate Key Establishment command. The responder device examines *CERTU* (formatted as IC_U as described in sub-clause C.4.2.2.4), confirms that the *Subject* identifier is the purported owner of the certificate, and runs the certificate processing steps described in section SEC 4 [B16].

C.4.2.3.3 Derive Keying Material

C.4.2.3.3.1 Initiator

The initiator performs the Elliptic Curve MQV scheme as specified in section 6.2 of SEC1 [B15] with the following instantiations:

1. The elliptic curve domain parameters shall be as specified in sub-clause C.4.2.2.1;
2. The KDF shall use the cryptographic hash function specified in sub-clause C.4.2.2.2;
3. The static public key $Q_{1,U}$ shall be the static public key of the initiator;
4. The ephemeral public key $Q_{2,U}$ shall be an ephemeral public key of the initiator generated as part of this transaction;
5. The static public key $Q_{1,V}$ shall be the static public key of the responder obtained from the responder's certificate communicated to the initiator by the responder;
6. The ephemeral public key $Q_{2,V}$ shall be based on the point-compressed octet string representation *QEV* of an ephemeral key of the responder communicated to the initiator by the responder;

7 The KDF parameter *keydatalen* shall be *MacKeyLen* + *KeyDataLen*, where *MacKeyLen* is the length of *MacKey* and *KeyDataLen* is the length of *KeyData*;

8 The parameter *SharedInfo* shall be the empty string;

The initiator device derives the keying material *MacKey* and *KeyData* from the output *K* as specified in section 3.6.1 of SEC1 [B15] by using *MacKey* as the leftmost *MacKeyLen* octets of *K* and *KeyData* as the rightmost *KeyDataLen* octets of *K*. *KeyData* is used subsequently as the shared secret and *MacKey* is used for key confirmation.

C.4.2.3.3.2 Responder

The responder performs the Elliptic Curve MQV scheme as specified in section 6.2 of SEC1 [B15] with the following instantiations:

1 The elliptic curve domain parameters shall be as specified in sub-clause C.4.2.2.1;

2 The KDF shall use the cryptographic hash function specified in sub-clause C.4.2.2.2;

3 The static public key $Q_{1,U}$ shall be the static public key of the initiator obtained from the initiator's certificate communicated to the responder by the initiator;

4 The ephemeral public key $Q_{2,U}$ shall be based on the point-compressed octet string representation *QEU* of an ephemeral key of the initiator communicated to the responder by the initiator;

5 The static public key $Q_{1,V}$ shall be the static public key of the responder;

6 The ephemeral public key $Q_{2,V}$ shall be an ephemeral public key of the responder generated as part of this transaction;

7 The KDF parameter *keydatalen* shall be *MacKeyLen* + *KeyDataLen*, where *MacKeyLen* is the length of *MacKey* and *KeyDataLen* is the length of *KeyData*;

8 The parameter *SharedInfo* shall be the empty string;

The responder device derives the keying material *MacKey* and *KeyData* from the output *K* as specified in section 3.6.1 of SEC1 [B15] by using *MacKey* as the leftmost *MacKeyLen* octets of *K* and *KeyData* as the rightmost *KeyDataLen* octets of *K*. *KeyData* is used subsequently as the shared secret and *MacKey* is used for key confirmation.

C.4.2.3.4 Confirm Keys

C.4.2.3.4.1 Initiator

The initiator device uses *MacKey* to compute its message authentication code *MACU* and sends it to the responder device by using the Confirm Key command via the Key Establishment Cluster Client.

The initiator device uses *MacKey* to confirm the authenticity of the responder by calculating *MACV* and comparing it with that sent by the responder.

C.4.2.3.4.2 Responder

The responder device uses *MacKey* to compute its message authentication code *MACV* and sends it to the initiator device by using the Confirm Key response command via the Key Establishment Cluster Server.

The responder device uses *MacKey* to confirm the authenticity of the initiator by calculating *MACU* and comparing it with that sent by the initiator.

C.5 Key Establishment Test Vectors

The following details the key establishment exchange data transformation and validation of test vectors for a pair of Smart Energy devices using Certificate based key exchange (CBKE) using Elliptical Curve Cryptography (ECC).

C.5.1 Preconfigured Data

Each device is expected to have been preinstalled with security information prior to initiating key establishment. The preinstalled data consists of the Certificate Authority's Public Key, a device specific certificate, and a device specific private key.

C.5.1.1 CA Public Key

The following is the Certificate Authority's Public Key.

```
02 00 FD E8 A7 F3 D1 08
42 24 96 2A 4E 7C 54 E6
9A C3 F0 4D A6 B8
```

C.5.1.2 Responder Data

The following is the certificate for device 1. The device has an IEEE of (>)0000000000000001, and will be the responder.

```
03 04 5F DF C8 D8 5F FB
```

```
8B 39 93 CB 72 DD CA A5
5F 00 B3 E8 7D 6D 00 00
00 00 00 00 00 01 54 45
53 54 53 45 43 41 01 09
00 06 00 00 00 00 00 00
```

The certificate has the following data embedded within it:

Public Key Reconstruction Data	03 04 5F DF C8 D8 5F FB 8B 39 93 CB 72 DD CA A5 5F 00 B3 E8 7D 6D
Subject (IEEE)	00 00 00 00 00 00 00 01
Issuer	54 45 53 54 53 45 43 41
Attributes	01 09 00 06 00 00 00 00 00 00

The private key for device 1 is as follows:

```
00 b8 a9 00 fc ad eb ab
bf a3 83 b5 40 fc e9 ed
43 83 95 ea a7
```

The public key for device 1 is as follows:

```
03 02 90 a1 f5 c0 8d ad
5f 29 45 e3 35 62 0c 7a
98 fa c4 66 66 a1
```

C.5.1.3 Initiator Data

The following is the certificate for device 2. The device has an IEEE of (>)0000000000000002, and will be the initiator.

```
02 06 15 E0 7D 30 EC A2
DA D5 80 02 E6 67 D9 4B
C1 B4 22 39 83 07 00 00
00 00 00 00 00 02 54 45
53 54 53 45 43 41 01 09
00 06 00 00 00 00 00 00
```

The certificate has the following data embedded within it:

Public Key Reconstruction Data	02 06 15 E0 7D 30 EC A2 DA D5 80 02 E6 67 D9 4B C1 B4 22 39 83 07

Subject (IEEE)	00 00 00 00 00 00 00 02
Issuer	54 45 53 54 53 45 43 41
Attributes	01 09 00 06 00 00 00 00 00 00

The private key for device 2 is as follows:

01 E9 DD B5 58 0C F7 2E

CE 7F 21 5F 0A E5 94 E4

8D F3 E7 FE E8

The public key for device 2 is:

03 02 5B BA 38 D0 C7 B5

43 6B 68 DF 72 8F 09 3E

7A 1D 6C 43 7E 6D

C.5.2 Key Establishment Messages

The following is the basic flow of messages back and forth between the initiator and the responder performing key establishment using the Key Establishment Cluster.

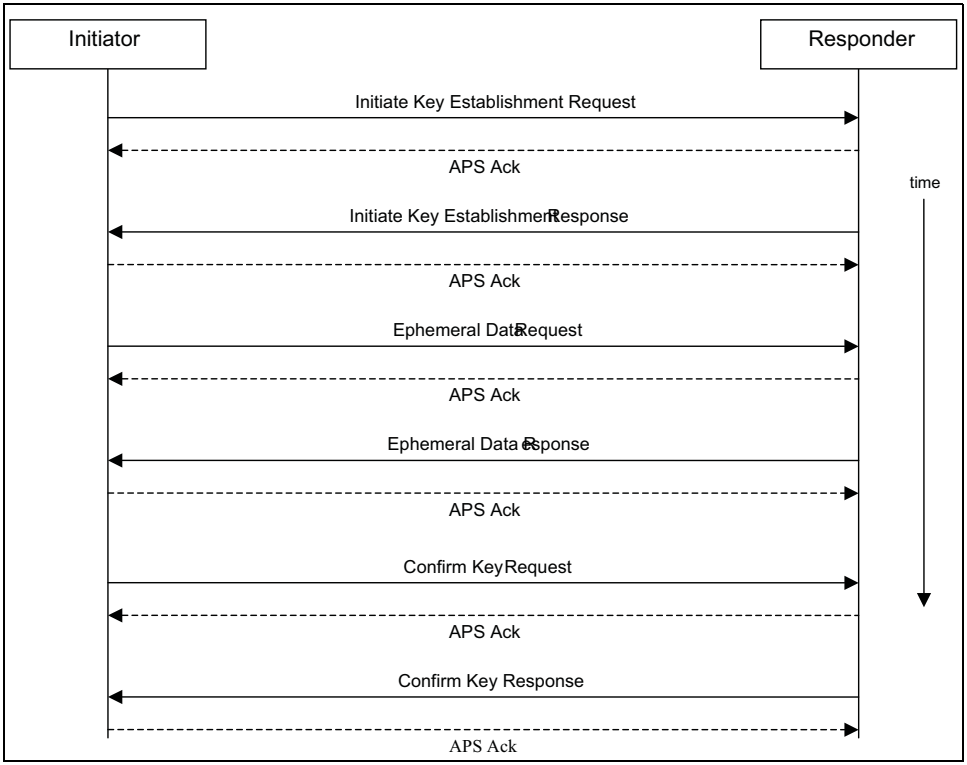

Figure C.12 Key Establishment Command Exchange

C.5.2.1 Initiate Key Establishment Request

The following is the APS message sent by the initiator (device 2) to the responder (device 1) for the initiate key establishment request.

```
40 0A 00 08 09 01 0A 01
01 00 00 01 00 03 06 02
06 15 E0 7D 30 EC A2 DA
D5 80 02 E6 67 D9 4B C1
B4 22 39 83 07 00 00 00
00 00 00 00 02 54 45 53
54 53 45 43 41 01 09 00
06 00 00 00 00 00 00
```

APS Header

Frame Control	0x40
Destination Endpoint	0x0A
Cluster Identifier	0x0800
Profile ID	0x0109
Source Endpoint	0x0A
APS Counter	0x01

ZCL Header

Frame Control	0x01	Client to Server
Sequence Number	0x00	
Command Identifier	0x00	Initiate Key Establishment Request
Key Establishment Suite	0x0001	ECMQV
Ephemeral Data Generate Time	0x03	
Confirm Key Generate Time	0x06	
Identity (IDU)	*	Device 2's certificate

C.5.2.2 Initiate Key Establishment Response

The following is the APS message sent by the responder (device 1) to the initiator (device 2) for the initiate key establishment response.

```
40 0A 00 08 09 01 0A 01
09 00 00 01 00 03 06 03
04 5F DF C8 D8 5F FB 8B
39 93 CB 72 DD CA A5 5F
00 B3 E8 7D 6D 00 00 00
00 00 00 00 01 54 45 53
54 53 45 43 41 01 09 00
06 00 00 00 00 00 00
```

APS Header

Frame Control	0x40
Destination Endpoint	0x0A
Cluster Identifier	0x0800
Profile ID	0x0109
Source Endpoint	0x0A
APS Counter	0x01

ZCL Header		
Frame Control	0x09	Server to Client
Sequence Number	0x00	
Command Identifier	0x00	Initiate Key Establishment Response
Key Establishment Suite	0x0001	ECMQV
Ephemeral Data Generate Time	0x03	
Confirm Key Generate Time	0x06	
Identity (IDV)	*	Device 1's certificate

C.5.2.3 Ephemeral Data Request

The following is the APS message sent by the initiator to the responder for the ephemeral data request.

```
40 0A 00 08 09 01 0A 02
01 01 01 03 00 E1 17 C8
6D 0E 7C D1 28 B2 F3 4E
90 76 CF F2 4A F4 6D 72
88
```

APS Header

Frame Control	0x40
Destination Endpoint	0x0A

Cluster Identifier	0x0800
Profile ID	0x0109
Source Endpoint	0x0A
APS Counter	0x02

ZCL Header

Frame Control	0x01	Client to Server
Sequence Number	0x01	
Command Identifier	0x01	Ephemeral Data Request
Ephemeral Data (QEU)	03 00 E1 17 C8 6D 0E 7C D1 28 B2 F3 4E 90 76 CF F2 4A F4 6D 72 88	

C.5.2.4 Ephemeral Data Response

The following is the APS message sent by the responder to the initiator for the ephemeral data response.

```
40 0A 00 08 09 01 0A 02
09 01 01 03 06 AB 52 06
22 01 D9 95 B8 B8 59 1F
3F 08 6A 3A 2E 21 4D 84
5E
```

APS Header

Frame Control	0x40
Destination Endpoint	0x0A
Cluster Identifier	0x0800
Profile ID	0x0109
Source Endpoint	0x0A
APS Counter	0x02

ZCL Header		
Frame Control	0x09	Server to Client
Sequence Number	0x01	
Command Identifier	0x01	Ephemeral Data Response
Ephemeral Data (QEV)	03 06 AB 52 06 22 01 D9 95 B8 B8 59 1F 3F 08 6A 3A 2E 21 4D 84 5E	

C.5.2.5 Confirm Key Request

The following is the APS message sent by the initiator to the responder for the confirm key request.

```
40 0A 00 08 09 01 0A 03
01 02 02 B8 2F 1F 97 74
74 0C 32 F8 0F CF C3 92
1B 64 20
```

APS Header

Frame Control	0x40
Destination Endpoint	0x0A
Cluster Identifier	0x0800
Profile ID	0x0109
Source Endpoint	0x0A
APS Counter	0x02

ZCL Header

Frame Control	0x01	Client to Server
Sequence Number	0x02	
Command Identifier	0x02	Confirm Key Request
Secure Message Authentication Code (MACU)	B8 2F 1F 97 74 74 0C 32 F8 0F CF C3 92 1B 64 20	

C.5.2.6 Confirm Key Response

The following is the APS message sent by the responder to the initiator for the confirm key response.

```
40 0A 00 08 09 01 0A 03
09 02 02 79 D5 F2 AD 1C
31 D4 D1 EE 7C B7 19 AC
68 3C 3C
```

APS Header

Frame Control	0x40
Destination Endpoint	0x0A
Cluster Identifier	0x0800
Profile ID	0x0109
Source Endpoint	0x0A
APS Counter	0x02

ZCL Header

Frame Control	0x09	Server to Client

Sequence Number	0x02	
Command Identifier	0x02	Confirm Key Response
Secure Message Authentication Code (MACV)	79 D5 F2 AD 1C 31 D4 D1 EE 7C B7 19 AC 68 3C 3C	

C.5.3 Data Transformation

The following are the various values used by the subsequent transformation.

U	Initiator
V	Responder
M(U)	Initiator Message Text (0x02)
M(V)	Responder Message Text (0x03)
ID(U)	Initiator's Identifier (IEEE address)
ID(V)	Responder's Identifier (IEEE address)
E(U)	Initiator's Ephemeral Public Key
E(V)	Responder's Ephemeral Public Key
E-P(U)	Initiator's Ephemeral Private Key
E-P(V)	Responder's Ephemeral Private Key
CA	Certificate Authority's Public Key
Cert(U)	Initiator's Certificate
Cert(V)	Responder's Certificate
Private(U)	Initiator's Private Key
Private(V)	Responder's Private Key
Shared Data	A pre-shared secret. NULL in Key Establishment.
Z	A shared secret

Note: '||' stands for bitwise concatenation

C.5.3.1 ECMQV Primitives

It is assumed that an ECC library is available for creating the shared secret given the local private key, local ephemeral public & private key, remote device's certificate, remote device's ephemeral public key, and the certificate authority's

public key. Further it is assumed that this library has been separately validated with a set of ECC test vectors. Those test vectors are outside the scope of this document.

C.5.3.2 Key Derivation Function (KDF)

Once a shared secret (Z) is established, a transform is done to create a SMAC (Secure Message Authenication Code) and a shared ZigBee Key.

C.5.3.3 Initiator Transform

Upon receipt of the responder's ephemeral data response, the initiator has all the data necessary to calculate the shared secret and derive the data for the confirm key request (SMAC).

C.5.3.3.1 Ephemeral Data

Public Key	03 00 E1 17 C8 6D 0E 7C D1 28 B2 F3 4E 90 76 CF F2 4A F4 6D 72 88
Private Key	00 13 D3 6D E4 B1 EA 8E 22 73 9C 38 13 70 82 3F 40 4B FF 88 62

C.5.3.3.2 Step Summary

1. Derive the Shared Secret using the ECMQV primitives
   a. Z = ECC_GenerateSharedSecret(Private(U), E(U), E-P(U), Cert(V), E(V), CA)
2. Derive the Keying data
   a. Hash-1 = Z || 00 00 00 01 || SharedData
   b. Hash-2 = Z || 00 00 00 02 || SharedData
3. Parse KeyingData as follows
   a. MacKey = First 128 bits (Hash-1) of KeyingData
   b. KeyData = Second 128 bits (Hash-2) of KeyingData
1. Create MAC(U)
   a. MAC(U) = MAC(MacKey) { M(U) || ID(U) || ID(V) || E(U) || E(V) }
2. Send MAC(U) to V.
3. Receive MAC(V) from V.
4. Calculate MAC(V)'
   a. MAC(V) = MAC(MacKey) { M(V) || ID(V) || ID(U) || E(V) || E(U) }

5 Verify MAC(V)' is the same as MAC(V).

C.5.3.3.3 Detailed Steps

1 Derive the Shared Secret using the ECMQV primitives

a Z = ECC_GenerateSharedSecret(Private(U), E(U), E-P(U), Cert(V), E(V), CA)

```
00 E0 D2 C3 CC D5 C1 06 A8 9C 4F 6C C2 6A 5F 7E
C9 DF 78 A7 BE
```

2 Derive the Keying data

a Hash-1 = Z || 00 00 00 01 || SharedData

Concatenation

```
00 E0 D2 C3 CC D5 C1 06 A8 9C 4F 6C C2 6A 5F 7E
C9 DF 78 A7 BE 00 00 00 01
```

Hash

```
90 F9 67 B2 2C 83 57 C1 0C 1C 04 78 8D E9 E8 48
```

b Hash-2 = Z || 00 00 00 02 || SharedData

Concatenation

```
00 E0 D2 C3 CC D5 C1 06 A8 9C 4F 6C C2 6A 5F 7E
C9 DF 78 A7 BE 00 00 00 02
```

Hash

```
86 D5 8A AA 99 8E 2F AE FA F9 FE F4 96 06 54 3A
```

3 Parse KeyingData as follows

a MacKey = First 128 bits (Hash-1) of KeyingData

b KeyData = Second 128 bits (Hash-2) of KeyingData

4 Create MAC(U)

a MAC(U) = MAC(MacKey) { M(U) || ID(U) || ID(V) || E(U) || E(V) }

Concatenation

```
02 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00
01 03 00 E1 17 C8 6D 0E 7C D1 28 B2 F3 4E 90 76
CF F2 4A F4 6D 72 88 03 06 AB 52 06 22 01 D9 95
B8 B8 59 1F 3F 08 6A 3A 2E 21 4D 84 5E 88 00 10
```

Hash

```
B8 2F 1F 97 74 74 0C 32 F8 0F CF C3 92 1B 64 20
```

5 Send MAC(U) to V.

6 Receive MAC(V) from V.

7 Calculate MAC(V)'

a MAC(V) = MAC(MacKey) { M(V) || ID(V) || ID(U) || E(V) || E(U) }

Concatenation

```
03 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
02 03 06 AB 52 06 22 01 D9 95 B8 B8 59 1F 3F 08
6A 3A 2E 21 4D 84 5E 03 00 E1 17 C8 6D 0E 7C D1
28 B2 F3 4E 90 76 CF F2 4A F4 6D 72 88 88 00 10
```

Hash

```
79 D5 F2 AD 1C 31 D4 D1 EE 7C B7 19 AC 68 3C 3C
```

8 Verify MAC(V)' is the same as MAC(V).

C.5.3.4 Responder Transform

Upon receipt of the initiator's confirm key request, the responder has all the data necessary to calculate the shared secret, validate the initator's confirm key message, and derive the data for the confirm key response (SMAC).

C.5.3.4.1 Ephemeral Data

Public Key	03 06 AB 52 06 22 01 D9 95 B8 B8 59 1F 3F 08 6A 3A 2E 21 4D 84 5E
Private Key	03 D4 8C 72 10 DD BC C4 FB 2E 5E 7A 0A A1 6A 0D B8 95 40 82 0B

C.5.3.4.2 Step Summary

1 Derive the Shared Secret using the ECMQV primitives

a Z = ECC_GenerateSharedSecret(Private(V), E(V), E-P(V), Cert(U), E(U), CA)

2 Derive the Keying data

a Hash-1 = Z || 00 00 00 01 || SharedData

b Hash-2 = Z || 00 00 00 02 || SharedData

3 Parse KeyingData as follows

a MacKey = First 128 bits (Hash-1) of KeyingData

b KeyData = Second 128 bits (Hash-2) of KeyingData

4 Create MAC(V)

a MAC(V) = MAC(MacKey) { M(V) || ID(V) || ID(U) || E(V) || E(U) }

5 Calculate MAC(U)'

a MAC(U) = MAC(MacKey) { M(U) || ID(U) || ID(V) || E(U) || E(V) }

6 Verify MAC(U)' is the same as MAC(U).

7 Send MAC(V) to U.

C.5.3.4.3 Detailed Steps

1 Derive the Shared Secret using the ECMQV primitives

a Z = ECC_GenerateSharedSecret(Private(U), E(U), E-P(U), Cert(V), E(V), CA)

```
00 E0 D2 C3 CC D5 C1 06 A8 9C 4F 6C C2 6A 5F 7E
C9 DF 78 A7 BE
```

2 Derive the Keying data

a Hash-1 = Z || 00 00 00 01 || SharedData

Concatenation

```
00 E0 D2 C3 CC D5 C1 06 A8 9C 4F 6C C2 6A 5F 7E
C9 DF 78 A7 BE 00 00 00 01
```

Hash

```
90 F9 67 B2 2C 83 57 C1 0C 1C 04 78 8D E9 E8 48
```

b Hash-2 = Z || 00 00 00 02 || SharedData

Concatenation

```
00 E0 D2 C3 CC D5 C1 06 A8 9C 4F 6C C2 6A 5F 7E
C9 DF 78 A7 BE 00 00 00 02
```

Hash

```
86 D5 8A AA 99 8E 2F AE FA F9 FE F4 96 06 54 3A
```

3 Parse KeyingData as follows

a MacKey = First 128 bits (Hash-1) of KeyingData

b KeyData = Second 128 bits (Hash-2) of KeyingData

4 Create MAC(V)

a MAC(V) = MAC(MacKey) { M(V) || ID(V) || ID(U) || E(V) || E(U) }

Concatenation

```
03 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
02 03 06 AB 52 06 22 01 D9 95 B8 B8 59 1F 3F 08
6A 3A 2E 21 4D 84 5E 03 00 E1 17 C8 6D 0E 7C D1
28 B2 F3 4E 90 76 CF F2 4A F4 6D 72 88 88 00 10
```

Hash

```
79 D5 F2 AD 1C 31 D4 D1 EE 7C B7 19 AC 68 3C 3C
```

5 Calculate MAC(V)'

a MAC(U) = MAC(MacKey) { M(U) || ID(U) || ID(V) || E(U) || E(V) }

Concatenation

```
02 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00
01 03 00 E1 17 C8 6D 0E 7C D1 28 B2 F3 4E 90 76
CF F2 4A F4 6D 72 88 03 06 AB 52 06 22 01 D9 95
B8 B8 59 1F 3F 08 6A 3A 2E 21 4D 84 5E 88 00 10
```

Hash

```
B8 2F 1F 97 74 74 0C 32 F8 0F CF C3 92 1B 64 20
```

6 Verify MAC(V)' is the same as MAC(V).

7 Send MAC(V) to U.[17]

17. CCB 984

This page intentionally blank

ANNEX

D

SMART ENERGY CLUSTER DESCRIPTIONS

The candidate material in this annex describing the Smart Energy Clusters, when approved, will be merged into the Foundation document of the ZigBee Cluster Library (ZCL) by the Cluster Library Development Board.

D.1 Annex Guidelines

D.1.1 Client/Server Model Information

The ZigBee Cluster Library Specification is used as the guiding reference for defining the rule set in defining the Client/Server model for the Smart Energy Profile. Please note the following items influence the further refinement of that definition:

- Attributes can be defined for both Client and Server side clusters. Attributes can be used to understand current state of activities within a device, enhancing both the diagnostic and maintenance of devices or the processes supported by that device.
- The ESP device acts as the transition point from upstream Wide Area Network (and subsequent upstream systems) to the ZigBee network. Because of this responsibility, in some of the clusters it acts as a proxy for the upstream systems. In those situations where the proxy condition occurs, plus where you find attributes defined or commands (transactions) initiated on both client/server sides, the ESP will be by default labeled as the Server side in the cluster descriptions.

D.1.2 Interpretation of Reserved Field Values or Bitmaps

To support backwards compatibility, devices should ignore any values or bit settings for any reserved field values. If the field is necessary for use interpreting

or necessary for use in conjunction with other fields, the whole message can be ignored.

To enable future growth and ensure backwards compatibility, any existing devices which encounter any fields applied after the end of a command shall treat them as reserved fields. The future addition of fields applied after the end of defined cluster commands are reserved solely for ZigBee specifications, Manufacturers shall not add fields after the end of commands.[18]

D.2 Demand Response and Load Control Cluster

D.2.1 Overview

This cluster provides an interface to the functionality of Smart Energy Demand Response and Load Control. Devices targeted by this cluster include thermostats and devices that support load control.

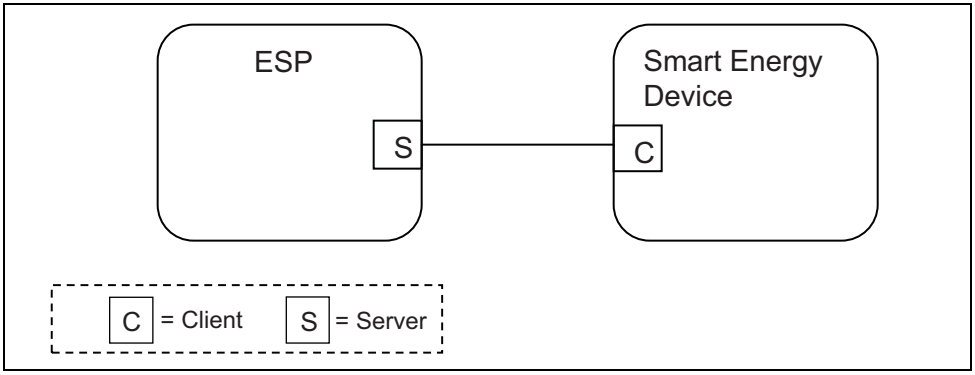

Figure D.1 Demand Response/Load Control Cluster Client Server Example

Please note the ESP is defined as the Server due to its role in acting as the proxy for upstream demand response/load control management systems and subsequent data stores.

D.2.2 Server

By default the ESP will be labeled as the Server side in the cluster descriptions, being able to initiate load control commands to other devices in the network.

18. CCB 968

D.2.2.1 Dependencies

Events carried using this cluster include a timestamp with the assumption that target devices maintain a real time clock. Devices can acquire and synchronize their internal clocks with the ESP as described in sub-clause 5.11.1.1.

If a device does not support a real time clock, it's assumed the device will ignore all values within the Time field except the "Start Now" value within the Time field.

Additionally, for devices without a real time clock, it's assumed those devices will utilize a method (i.e. ticks, countdowns, etc.) to approximate the correct duration period.

D.2.2.2 Attributes

There are no attributes for the Demand Response and Load Control Cluster server.

D.2.2.3 Commands Generated

The command IDs generated by the Demand Response and Load Control cluster server are listed in Table D.1.

Table D.1 Command IDs for the Demand Response and Load Control Server

Command Identifier Field Value	Description	Mandatory/ Optional
0x00	Load Control Event	M
0x01	Cancel Load Control Event	M
0x02	Cancel All Load Control Events	M
0x03 – 0xff	Reserved	

D.2.2.3.1 Load Control Event

D.2.2.3.1.1 Payload Format

The Load Control Event command payload shall be formatted as illustrated in Figure D.2.

Octets	4	2	1	4	2	1	1
Data Type	Unsigned 32-bit integer	16-bit BitMap	Unsigned 8-bit integer	UTC Time	Unsigned 16-bit integer	Unsigned 8-bit integer	Unsigned 8-bit integer
Field Name	Issuer Event ID (M)	Device Class (M)	Utility Enrolment Group (M)	Start Time (M)	Duration In Minutes (M)	Criticality Level (M)	Cooling Temperature Offset (O)

Octets	1	2	2	1	1	1
Data Type	Unsigned 8-bit integer	Signed 16-bit integer	Signed 16-bit integer	Signed 8-bit integer	Unsigned 8-bit integer	8-bit BitMap
Field Name	Heating Temperature Offset (O)	Cooling Temperature Set Point (O)	Heating Temperature Set Point (O)	Average Load Adjustment Percentage (O)	Duty Cycle (O)	Event Control (M)

Figure D.2 Format of the Load Control Event Payload

Note: M = Mandatory field, O = Optional field. **All fields must be present in the payload.** Optional fields will be marked with specific values to indicate they are not being used.

D.2.2.3.1.1.1 Payload Details

Issuer Event ID (mandatory): Unique identifier generated by the Energy provider. The value of this field allows matching of Event reports with a specific Demand Response and Load Control event. The expected the value contained in this field shall be a unique number managed by upstream systems or a UTC based time stamp (UTCTime data type) identifying when the Load Control Event was issued.

Device Class (mandatory): Bit encoded field representing the Device Class to apply the current Load Control Event. Each bit, if set individually or in combination, indicates the class device(s) needing to participate in the event. (Note that the participating device may be different than the controlling device. For instance, a thermostat may act on behalf of an HVAC compressor or furnace and/or Strip Heat/Baseboard Heater and should take action on their behalf, as the

thermostat itself is not subject to load shed but controls devices that are subject to load shed.) The encoding of this field is in Table D.2:

Table D.2 Device Class Field BitMap/Encoding

Bit	Description
0	HVAC compressor or furnace
1	Strip Heaters/Baseboard Heaters
2	Water Heater
3	Pool Pump/Spa/Jacuzzi
4	Smart Appliances
5	Irrigation Pump
6	Managed Commercial & Industrial (C&I) loads
7	Simple misc. (Residential On/Off) loads
8	Exterior Lighting
9	Interior Lighting
10	Electric Vehicle
11	Generation Systems
12 to 15	Reserved

Device manufacturers shall recognize the Device Class or set of Devices Classes that corresponds to its functionality. For example, a thermostat (PCT) may react when Bit 0 is set since it controls the HVAC and/or furnace. Another example is a device that acts like an EMS where it controls exterior lights, interior lights, and simple misc. load control devices. In this case the EMS would react when Bits 7, 8, or 9 are set individually or in combination.

Utility Enrolment Group (mandatory): The Utility Enrolment Group field can be used in conjunction with the Device Class bits. It provides a mechanism to direct Load Control Events to groups of Devices. Example, by assigning two different groups relating to either Demand Response programs or geographic areas, Load Control Events can be further directed for a sub-set of Device Classes (i.e. Device Class Bit 0 and Utility Enrolment Group #1 vs. Device Class Bit0 and Utility Enrolment Group #2). 0x00 addresses all groups, and values 0x01 to 0xFF address individual groups that match. Please refer to sub-clause D.2.3.2.1 for further details.

If the Device Class and/or Utility Enrolment Group fields don't apply to your End Device, the Load Control Event command is ignored.

Start Time (mandatory): UTC Timestamp representing when the event is scheduled to start. A start time of 0x00000000 is a special time denoting “now.”

Duration In Minutes (mandatory): Duration of this event in number of minutes. Maximum value is 1440 (one day).

Criticality Level (mandatory): This field defines the level of criticality of this event. The action taken by load control devices for an event can be solely based on this value, or combination with other Load Control Event fields supported by this device. For example, additional fields such as Average Load Adjustment Percentage, Duty Cycle, Cooling Temperature Offset, Heating Temperature Offset, Cooling Temperature Set Point or Heating Temperature Set Point can be used in combination with the Criticality level. Criticality levels are listed in Table D.3.

Table D.3 Criticality Levels

Criticality Level	Level Description	Participation
0	Reserved	
1	Green	Voluntary
2	1	Voluntary
3	2	Voluntary
4	3	Voluntary
5	4	Voluntary
6	5	Voluntary
7	Emergency	Mandatory
8	Planned Outage	Mandatory
9	Service Disconnect	Mandatory
0x0A to 0x0F	Utility Defined	Utility Defined
0x10 to 0xFF	Reserved	

The criticality level 0x0 and 0x10 to 0xFF are reserved for future profile changes and not used.

“Green” event, level 0x01, may be used to denote that the energy delivered uses an abnormal amount from non-“green” sources. Participation in this event is voluntary.

The criticality levels 0x02 through 0x06 (Levels 1 through 5) indicate progressively increasing levels of load reduction are being requested by the utility. Participation in these events is voluntary.

The criticality level 0x07 is used to indicate an “Emergency” event. Participation in this event is mandatory, as defined by the utility. The expected response to this event is termination of all non-essential energy use, as defined by the utility. Exceptions to participation in this event type must be managed by the utility.

The criticality level 0x08 is used to indicate a “Planned Outage” event. Participation in this event is mandatory, as defined by the utility. The expected response to this event is termination of delivery of all non-essential energy, as defined by the utility. Exceptions to participation in this event type must be managed by the utility.

The criticality level 0x09 is used to indicate a “Service Disconnect” event. Participation in this event is mandatory, as defined by the utility. The expected response to this event is termination of delivery of all non-essential energy, as defined by the utility. Exceptions to participation in this event type must be managed by the utility.

Levels 0x0A to 0x0F are available for Utility Defined criticality levels.

Cooling Temperature Offset (optional): Requested offset to apply to the normal cooling setpoint at the time of the start of the event in + 0.1 ºC.

Heating Temperature Offset (optional): Requested offset to apply to the normal heating setpoint at the time of the start of the event in + 0.1 ºC.

The Cooling and Heating Temperature Offsets represent a temperature change (Delta Temperature) that will be applied to both the associated heating and cooling set points. The temperature offsets (Delta Temperatures) will be calculated per the Local Temperature in the Thermostat. The calculated temperature will be interpreted as the number of degrees to be added to the cooling set point and subtracted from the heating set point. Sequential demand response events are not cumulative. The Offset shall be applied to the normal setpoint.

Each offset represents the temperature offset (Delta Temperature) in degrees Celsius, as follows: Delta Temperature Offset / 10 = delta temperature in degrees Celsius. Where 0.00°C <= temperature <= 25.4 ºC, corresponding to a Temperature in the range 0x00 to 0x0FE. The maximum resolution this format allowed is 0.1 ºC.

A DeltaTemperature of 0xFF indicates that the temperature offset is not used.

If a temperature offset is sent that causes the heating or cooling temperature set point to exceed the limit boundaries that are programmed into the thermostat, the thermostat should respond by setting the temperature at the limit.

Cooling Temperature Set Point (optional): Requested cooling set point in 0.01 degrees Celsius.

Heating Temperature Set Point (optional): Requested heating set point in 0.01 degrees Celsius.

Cooling and heating temperature set points will be defined and calculated per the *LocalTemperature* attribute in the Thermostat Cluster [B2].

These fields represent the temperature in degrees Celsius, as follows:

Cooling Temperature Set Point / 100 = temperature in degrees Celsius

Where -273.15°C <= temperature <= 327.66°C, corresponding to a Cooling and/or Heating Temperature Set Point in the range 0x954d to 0x7fff.

The maximum resolution this format allows is 0.01°C.

A Cooling or Heating Temperature Set Point of 0x8000 indicates that the temperature set point is not used.

If a temperature is sent that exceeds the temperature limit boundaries that are programmed into the thermostat, the thermostat should respond by setting the temperature at the limit.

The thermostat shall not use a Cooling or Heating Temperature Set Point that causes the device to use more energy than the normal setting.

When both a Temperature Offset and a Temperature Set Point are provided, the thermostat may use either as defined by the device manufacturer. The thermostat should use the setting that provides the lowest energy consumption.

Average Load Adjustment Percentage (optional): Defines a maximum energy usage limit as a percentage of the client implementations specific average energy usage. The load adjustment percentage is added to 100% creating a percentage limit applied to the client implementations specific average energy usage. A -10% load adjustment percentage will establish an energy usage limit equal to 90% of the client implementations specific average energy usage. Each load adjustment percentage is referenced to the client implementations specific average energy usage. There are no cumulative effects.

The range of this field is -100 to +100 with a resolution of 1 percent. A -100% value equals a total load shed. A +100% value will limit the energy usage to the client implementations specific average energy usage.

A value of 0x80 indicates the field is not used. All other values are reserved for future use.

Duty Cycle (optional): Defines the maximum On state duty cycle as a percentage of time. Example, if the value is 80, the device would be in an “on state” for 80%

of the time for the duration of the event. Range of the value is 0 to 100. A value of 0xFF indicates the field is not used. All other values are reserved for future use.

Duty cycle control is a device specific issue and shall be managed by the device manufacturer. It is expected that the duty cycle of the device under control will span the shortest practical time period in accordance with the nature of the device under control and the intent of the request for demand reduction. For typical Device Classes, one minute for each 10% of duty cycle is recommended. It is expected that the "off state" will precede the "on state".

Event Control (mandatory): Identifies additional control options for the event. The BitMap for this field is described in Table D.4.

Table D.4 Event Control Field BitMap

Bit	Description
0	1= Randomize Start time, 0=Randomized Start not Applied
1	1= Randomize End time, 0=Randomized End not Applied
2 to 7	Reserved

Note: The randomization attribute will be used in combination with two bits to determine if the Event Start and Stop Times are randomized. By default devices will randomize the start and stop of an event. Refer to sub-clause D.2.3.2.2 and sub-clause D.2.3.2.3 for the settings of these values.

D.2.2.3.1.1.2 When Generated

This command is generated when the ESP wants to control one or more load control device(s), usually as the result of an energy curtailment command from the Smart Energy network.

D.2.2.3.1.1.3 Responses to Load Control Event

The server receives the cluster specific commands detailed in sub-clause D.2.3.3.1.

D.2.2.3.2 Cancel Load Control Event

D.2.2.3.2.1 Payload Format

The Cancel Load Control Event command payload shall be formatted as illustrated in Figure D.3.

Octets	4	2	1	1	4
Data Type	Unsigned 32-bit integer	16-bit BitMap	Unsigned 8-bit integer	8-bit BitMap	UTCTime
Field Name	Issuer Event ID	Device Class (M)	Utility Enrollment Group (M)	Cancel Control	Effective Time

Figure D.3 Format of the Cancel Load Control Event Payload

D.2.2.3.2.1.1 Payload Details

Issuer Event ID (mandatory): Unique identifier generated by the Energy provider. The value of this field allows matching of Event reports with a specific Demand Response and Load Control event. It's expected the value contained in this field is a unique number managed by upstream systems or a UTC based time stamp (UTCTime data type) identifying when the Load Control Event was issued.

Device Class (mandatory): Bit encoded field representing the Device Class to apply the current Load Control Event. Each bit, if set individually or in combination, indicates the class device(s) needing to participate in the event. (Note that the participating device may be different than the controlling device. For instance, a thermostat may act on behalf of an HVAC compressor or furnace and/or Strip Heat/Baseboard Heater and should take action on their behalf, as the thermostat itself is not subject to load shed but controls devices that are subject to load shed.) The encoding of the Device Class is listed in Table D.2:

Utility Enrolment Group (mandatory): The Utility Enrolment Group field can be used in conjunction with the Device Class bits. It provides a mechanism to direct Load Control Events to groups of Devices. Example, by assigning two different groups relating to either Demand Response programs or geographic areas, Load Control Events can be further directed for a sub-set of Device Classes (i.e. Device Class Bit 0 and Utility Enrolment Group #1 vs. Device Class Bit0 and Utility Enrolment Group #2). 0x00 addresses all groups, and values 0x01 to 0xFF address individual groups that match Please refer to sub-clause D.2.3.2.1 for further details.

If the Device Class and/or Utility Enrolment Group fields don't apply to your End Device, the Load Control Event command is ignored.

Cancel Control (mandatory): The encoding of the Cancel Control is listed in Table D.5.

Table D.5 Cancel Control

Bit	Description
0	To be 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 Zero (0) indicates that randomization is overridden and the event should be terminated immediately at the Effective Time. A value of One (1) indicates the event should end using randomization settings in the original event.
1 to 7	Reserved

Effective Time (mandatory): UTC Timestamp representing when the canceling of the event is scheduled to start. A start time of 0x00000000 is a special time denoting "now."

D.2.2.3.2.1.2 When Generated

This command is generated when the ESP wants to cancel previously scheduled control of one or more load control device(s), usually as the result of an energy curtailment command from the Smart Energy network.

D.2.2.3.2.1.3 Responses to Cancel Load Control Event

The server receives the cluster specific commands is detailed in sub-clause D.2.3.3.1.

Note: If the Cancel command is received after the event has ended, the device shall reply using the "Report Event Status Command" with an Event Status of "Load Control Event command Rejected".

D.2.2.3.3 Cancel All Load Control Events

D.2.2.3.3.1 Payload Format

The Cancel All Load Control Events command payload shall be formatted as illustrated in Figure D.4.

Octets	1
Data Type	8-bit BitMap
Field Name	Cancel Control

Figure D.4 Format of the Cancel All Load Control Events Payload

D.2.2.3.3.1.1 Payload Details

Cancel Control: The encoding of the Cancel Control is listed in Table D.6.

Table D.6 Cancel All Command Cancel Control Field

Bit	Description
0	To be used when the Event is currently in process and a cancel command is received. A value of Zero (0) indicates that randomization is overridden and the event should be terminated immediately. A value of One (1) indicates the event should end using randomization settings in the original event.
1 to 7	Reserved

D.2.2.3.3.2 When Generated

This command is generated when the ESP wants to cancel all events for control device(s).

D.2.2.3.3.3 Responses to Cancel All Load Control Events

The server receives the cluster specific commands detailed in sub-clause D.2.3.3.1. The Cancel All Load Control Events command is processed by the device as if individual Cancel Load Control Event commands were received for all of the currently stored events in the device. The device will respond with a “Report Event Status Command” for each individual load control event canceled.

D.2.3 Client

This section identifies the attributes and commands provided by Client devices.

D.2.3.1 Dependencies

Devices receiving and acting upon Load Control Event commands must be capable of storing and supporting at least three unique instances of events. As a highly recommended recovery mechanism, when maximum storage of events has been reached and additional Load Control Events are received that are unique (not superseding currently stored events), devices should ignore additional Load Control Events and when storage becomes available, utilize the GetScheduledEvents command to retrieve any previously ignored events.

Events carried using this cluster include a timestamp with the assumption that target devices maintain a real time clock. Devices can acquire and synchronize their internal clocks with the ESP as described in sub-clause 5.11.1.1.

If a device does not support a real time clock, it's assumed the device will ignore all values within the Time field except the "Start Now" value within the Time field.

Additionally, for devices without a real time clock it's assumed those devices will utilize a method (i.e. ticks, countdowns, etc.) to approximate the correct duration period.

D.2.3.2 Client Cluster Attributes

Table D.7 Demand Response Client Cluster Attributes

Identifier	Name	Type	Range	Access	Default	Mandatory/ Optional
0x0000	UtilityEnrolmentGroup	Unsigned 8 bit Integer	0x00 to 0xFF	Read/ Write	0x00	M
0x0001	StartRandomizeMinutes	Unsigned 8 bit Integer	0x00 to 0x3C	Read/ Write	0x1E	M
0x0002	StopRandomizeMinutes	Unsigned 8 bit Integer	0x00 to 0x3C	Read/ Write	0x1E	M
0x0003	DeviceClassValue	Unsigned 16 bit Integer	0x00 to 0xFF	Read	-	M
0x0004 to 0xFFFF	Reserved					

D.2.3.2.1 Utility Enrolment Group Attribute

The *UtilityEnrolmentGroup* provides a method for utilities to assign devices to groups. In other words, Utility defined groups provide a mechanism to arbitrarily group together different sets of load control or demand response devices for use as part of a larger utility program. The definition of the groups, implied usage, and their assigned values are dictated by the Utilities and subsequently used at their discretion, therefore outside the scope of this specification. The valid range for this attribute is 0x00 to 0xFF, where 0x00 (the default value) indicates the device is a member of all groups and values 0x01 to 0xFF indicates that the device is member of that specified group.

D.2.3.2.2 Start Randomization Minutes

The *StartRandomizedMinutes* represents the maximum number of minutes to be used when randomizing the start of an event. As an example, if *StartRandomizedMinutes* is set for 3 minutes, the device could randomly select 2 minutes (but never greater than the 3 minutes) for this event, causing the start of the event to be delayed by two minutes. The valid range for this attribute is 0x00 to 0x3C where 0x00 indicates start event randomization is not performed.

D.2.3.2.3 End Randomization Minutes

The *EndRandomizedMinutes* represents the maximum number of minutes to be used when randomizing the end of an event. As an example, if *EndRandomizedMinutes* is set for 3 minutes, the device could randomly select one minute (but never greater than 3 minutes) for this event, causing the end of the event to be delayed by one minute. The valid range for this attribute is 0x00 to 0x3C where 0x00 indicates end event randomization is not performed.

D.2.3.2.4 DeviceClassValue Attribute

The *DeviceClassValue* attribute identifies which bits the device will match in the Device Class fields. Please refer to sub-clause D.2.3.2.1 for further details.

D.2.3.3 Commands Generated

The command IDs generated by the Demand Response and Load Control client cluster are listed in Table D.8.

Table D.8 Generated Command IDs for the Demand Response and Load Control Client

Command Identifier Field Value	Description	Mandatory/ Optional

Table D.8 Generated Command IDs for the Demand Response and Load Control Client

0x00	Report Event Status	M
0x01	Get Scheduled Events	M
0x02 – 0xff	Reserved	

D.2.3.3.1 Report Event Status

D.2.3.3.1.1 Payload Format

The Report Event Status command payload shall be formatted as illustrated in Figure D.5.

Octets	**4**	**1**	**4**	**1**	**2**	**2**
Data Type	Unsigned 32-bit integer	Unsigned 8-bit integer	UTCTime	Unsigned 8-bit integer	Unsigned 16-bit integer	Unsigned 16-bit integer
Field Name	Issuer Event ID (M)	Event Status (M)	Event Status Time (M)	Criticality Level Applied (M)	Cooling Temperature Set Point Applied (O)	Heating Temperature Set Point Applied (O)

Octets	**1**	**1**	**1**	**1**	**42**
Data Type	Signed 8-bit integer	Unsigned 8-bit integer	8-bit BitMap	Unsigned 8-bit integer	Octets (non-ZCL Data Type)
Field Name	Average Load Adjustment Percentage Applied (O)	Duty Cycle Applied (O)	Event Control (M)	Signature Type	Signature (M)

Figure D.5 Format of the Report Event Status Command Payload

D.2.3.3.1.1.1 xPayload Details

Issuer Event ID (mandatory): Unique identifier generated by the Energy provider. The value of this field allows matching of Event reports with a specific Demand Response and Load Control event. It's expected the value contained in this field is a unique number managed by upstream systems or a UTC based time stamp (UTCTime data type) identifying when the Load Control Event was issued.

Event Status (mandatory): Table D.9 lists the valid values returned in the Event Status field.

Table D.9 Event Status Field Values

Value	Description
0x00	Reserved for future use.
0x01	Load Control Event command received
0x02	Event started
0x03	Event completed
0x04	User has chosen to "Opt-Out", user will not participate in this event
0x05	User has chosen to "Opt-In", user will participate in this event
0x06	The event has been cancelled
0x07	The event has been superseded
0x08	Event partially completed with User "Opt-Out".
0x09	Event partially completed due to User "Opt-In".
0x0A	Event completed, no User participation (Previous "Opt-Out").
0x0B to 0xF7	Reserved for future use.
0xF8	Rejected - Invalid Cancel Command (Default)
0xF9	Rejected - Invalid Cancel Command (Invalid Effective Time)
0xFA	Reserved
0xFB	Rejected - Event was received after it had expired (Current Time > Start Time + Duration)
0xFC	Reserved for future use.
0xFD	Rejected - Invalid Cancel Command (Undefined Event)
0xFE	Load Control Event command Rejected
0xFF	Reserved for future use.

Should a device issue one or more “OptOut” or “OptIn” RES commands during an event that is eventually cancelled, the event shall be recorded as a cancelled event (Status = 0x06) at its effective time.

Should a device issue one or more “OptOut” or “OptIn” RES commands during an event that is not cancelled, the event shall be recorded as partially completed based on the last RES command sent (Status = 0x08 or 0x09).

When a device returns a status of 0xFD (Rejected - Invalid Cancel Command (Undefined Event)), all optional fields should report their “Ignore” values.

When a device receives a duplicate RES command, it should ignore the duplicate commands. Please Note: As a recommended best practice, ESP applications should provide a mechanism to assist in filtering duplicate messages received on the WAN.

Event Status Time (mandatory): UTC Timestamp representing when the event status occurred. This field shall not use the value of 0x00000000.

Criticality Level Applied (mandatory): Criticality Level value applied by the device, see the corresponding field in the Load Control Event Command for more information.

Cooling Temperature Set Point Applied (optional): Cooling Temperature Set Point value applied by the device, see the corresponding field in the Load Control Event Command for more information. The value 0x8000 means that this field has not been used by the end device.

Heating Temperature Set Point Applied (optional): Heating Temperature Set Point value applied by the device, see the corresponding field in the Load Control Event Command for more information. The value 0x8000 means that this field has not been used by the end device.

Average Load Adjustment Percentage Applied (optional): Average Load Adjustment Percentage value applied by the device, see the corresponding field in the Load Control Event Command for more information. The value 0x80 means that this field has not been used by the end device.

Duty Cycle Applied (optional): Defines the maximum On state duty cycle applied by the device. The value 0xFF means that this field has not been used by the end device. Refer to sub-clause D.2.2.3.1.1.1.

Event Control (mandatory): Identifies additional control options for the event. Refer to sub-clause D.2.2.3.1.1.1.

Signature Type (mandatory): An 8 bit Unsigned integer enumerating the type of algorithm use to create the Signature. The enumerated values are:

Enumerated Value	Signature Type
0x00	Reserved
0x01	ECDSA
0x02 to 0xFF	Reserved

Signature (mandatory): A non-repudiation signature created by using the Matyas-Meyer-Oseas hash function (specified in Annex B.6 in [B4]) used in conjunction with ECDSA. The signature creation process will occur in two steps:

1. Pass the first ten fields, which includes all fields up to the Signature field, of the Report Event Status command (listed in Figure D.5) through ECDSA using the device's ECC Private Key, generating the signature *(r,s)*. Note: ECDSA internally uses the MMO hash function in place of the internal SHA-1 hash function.
2. Concatenate ECDSA signature components *(r,s)* and place into the Signature field within the Report Event Status command. Note: the lengths of *r* and *s* are implicit, based on the curve used. Verifying the signature will require breaking the signature field back into the discrete components *r* and *s*, based on the length.

D.2.3.3.1.2 When Generated

This command is generated when the client device detects a change of state for an active Load Control event. (The transmission of this command should be delayed after a random delay between 0 and 5 seconds, to avoid a potential storm of packets.)

D.2.3.3.2 Get Scheduled Events Command

This command is used to request that all scheduled Load Control Events, starting at or after the supplied Start Time, are re-issued to the requesting device. When received by the Server, one or more Load Control Event commands (see sub-clause D.2.2.3.1) will be sent covering both active and scheduled Load Control Events.

D.2.3.3.2.1 Payload Format

The Get Scheduled Events command payload shall be formatted as illustrated in Figure D.6

Octets	**4**	**1**
Data Type	UTCTime	Unsigned 8-bit integer
Field Name	Start Time (M)	Number of Events (M)

Figure D.6 Format of the Get Scheduled Events Command Payload

Start Time (mandatory): UTC Timestamp representing the starting time for any scheduled events to be re-sent.

Number of Events (mandatory): Represents the maximum number of events to be sent. A value of 0 would indicate all available events are to be returned.

D.2.3.3.2.2 When Generated

This command is generated when the client device wishes to verify the available Load Control Events or after a loss of power/reset occurs and the client device needs to recover currently active or scheduled Load Control Events.

D.2.3.4 Commands Received

The client receives the cluster specific commands detailed in sub-clause D.2.2.

D.2.3.5 Attribute Reporting

Attribute reporting is not expected to be used for this cluster. The Client side attributes are not expected to be changed by the Client, only used during Client operations.

D.2.4 Application Guidelines

The criticality level is sent by the utility to the load control device to indicate how much load reduction is requested. The utility is not required to use all of the criticality levels that are described in this specification. A load control device is not required to provide a unique response to each criticality level that it may receive.

The Average Load Adjustment Percentage, temperature offsets, and temperature set points are used by load control devices and energy management systems on a “voluntary” or “optional” basis. These devices are not required to use the values that are provided by the utility. They are provided as a recommendation by the utility.

The load control device shall, in a manner that is consistent with this specification, accurately report event participation by way of the Report Event Status message.

The Average Load Adjustment Percentage is sent by the utility to the load control device to indicate how much load reduction is requested. The load control device may respond to this information in a unique manner as defined by the device manufacturer.

The Duty Cycle is sent by the utility to the load control device to indicate the maximum “On state” for a device. The control device may respond to this information in a unique manner as defined by the device manufacturer.

The cooling temperature offset may be sent by the utility to the load shed control to indicate how much indoor cooling temperature offset is requested. Response of a load control device to this information is not mandatory. The control device may

respond to this information in a unique manner as defined by the device manufacturer.

The heating temperature offset may be sent by the utility to the load control device to indicate how much indoor heating temperature offset is requested The control device may respond to this information in a unique manner as defined by the device manufacturer.

The cooling temperature may be sent by the utility to the load control device to indicate the indoor cooling temperature setting that is requested The control device may respond to this information in a unique manner as defined by the device manufacturer.

The heating temperature may be sent by the utility to the load control device to indicate the indoor heating temperature setting that is requested. The control device may respond to this information in a unique manner as defined by the device manufacturer.

Note: The most recent Load Control Event supersedes any previous Load Control Event command for the set of Device Classes and groups for a given time. Nested events and overlapping events are not allowed. The current active event will be terminated if a new event is started.

D.2.4.1 Load Control Rules, Server

D.2.4.1.1 Load Control Server, Identifying Use of SetPoint and Offset Fields

The use of the fields, Heating and Cooling Temperature Set Points and Heating and Cooling Temperature Offsets is optional. All fields in the payload must be populated. Non-use of these fields by the Server is indicated by using the following values: 0x8000 for Set Points and 0xFF for Offsets. When any of these four fields are indicated as optional, they shall be ignored by the client.

D.2.4.1.2 Load Control Server, Editing of Scheduled Events

Editing of a scheduled demand response event is not allowed. Editing of an active demand response event is not allowed. Nested events and overlapping events are not allowed. The current active event will be terminated if a new event is started.

D.2.4.2 Load Control Rules, Client

D.2.4.2.1 Start and Stop Randomization

When shedding loads (turning a load control device off), the load control device will optionally apply start time randomization based on the values specified in the Event Control Bits and the Clients Start Randomization Minutes attribute. By default, devices will apply a random delay between zero and 5 minutes. Devices will support randomization delays of up to 60 minutes.

When ending a load control event, the load control device will support the same randomization features as provided in the start load control event.

D.2.4.2.2 Editing of DR Control Parameters

In Load Control Device and energy management systems, editing of the demand response control parameters while participating in an active demand response event is not allowed.

D.2.4.2.3 Response to Price Events + Load Control Events

The residential system's response to price driven events will be considered in addition to the residential system's response to demand response events. Demand response events which require that the residential system is turned off have priority over price driven events. Demand response events which require that the residential system go to a fixed setting point have priority over price driven events. In this case, the thermostat shall not use a Cooling or Heating Temperature Set Point that causes the device to use more energy than the price driven event setting.

D.2.4.2.4 Thermostat/HVAC Controls

A residential HVAC system will be allowed to change mode, from off to Heat, off to Cool, Cool to Heat, or Heat to Cool, during a voluntary event which is currently active. The HVAC control must acknowledge the event, as if it was operating, in that mode, at the start of the event. The HVAC control must obey the event rules that would have been enforced if the system had been operating in that mode at the start of the active event.

An event override message, "opt-out", will be sent by the load control device or energy management system if the operator chooses not to participate in a demand response event by taking action to override the programmed demand reduction response. The override message will be sent at the start of the event. In the case where the event has been acknowledged and started, the override message will be sent when the override occurs.

D.2.4.2.5 Demand Response and Load Control Transaction Examples

The following example in Figure D.7 depicts the transactions that would take place for two events, one that is successful and another that is overridden by the user.

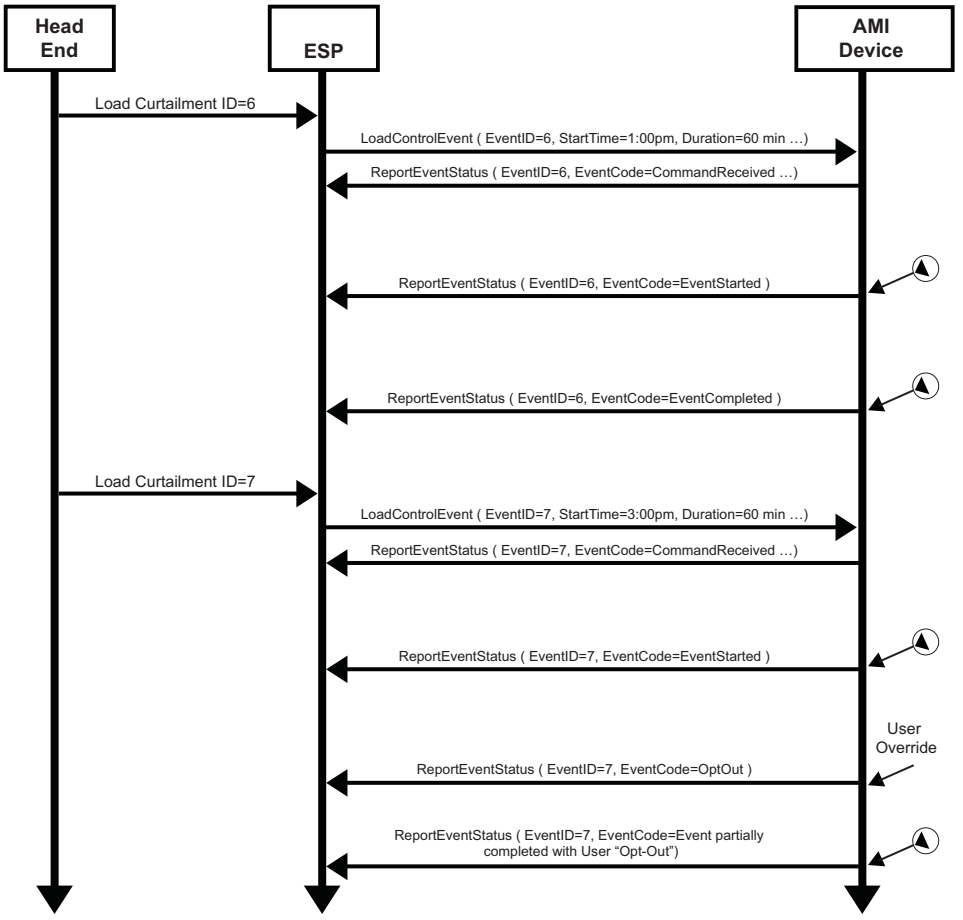

Figure D.7 Example of Both a Successful and an Overridden Load Curtailment Event

The example in Figure D.8 depicts the transactions that would take place when an event is superseded by an event that is eventually cancelled.

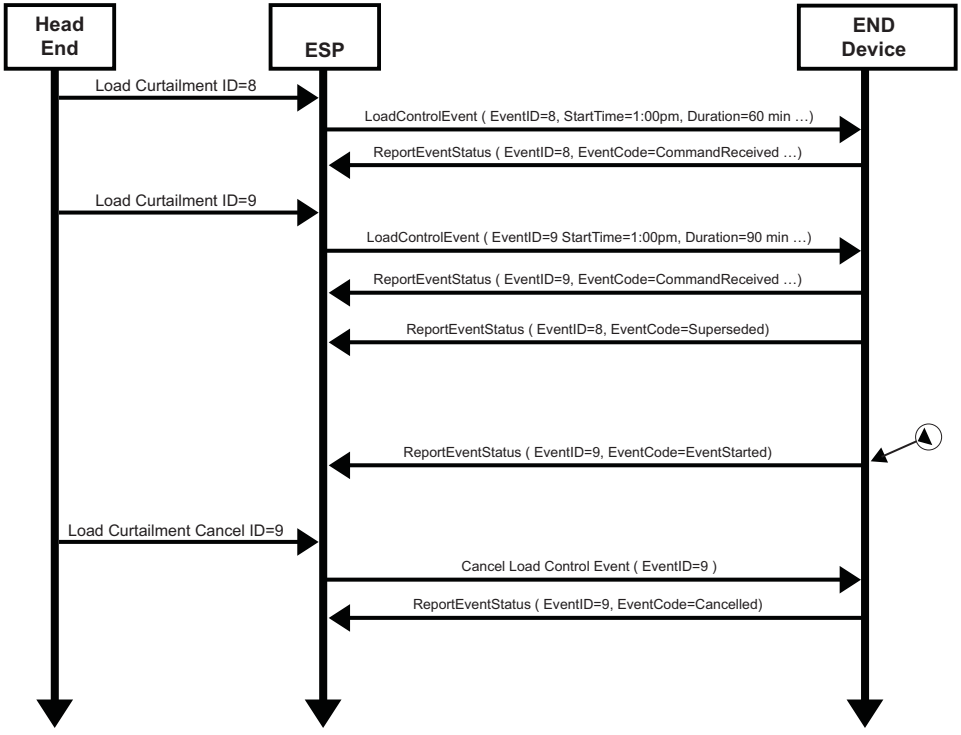

Figure D.8 Example of a Load Curtailment Superseded and Another Cancelled

Please refer to Annex E for more information regarding the management and behavior of overlapping events.

D.3 Simple Metering Cluster

D.3.1 Overview

The Simple Metering Cluster provides a mechanism to retrieve usage information from Electric, Gas, Water, and potentially Thermal metering devices. These devices can operate on either battery or mains power, and can have a wide variety of sophistication. The Simple Metering Cluster is designed to provide flexibility while limiting capabilities to a set number of metered information types. More

advanced forms or data sets from metering devices will be supported in the Smart Energy Tunneling Cluster, which will be defined in sub-clause D.6.

The following figures identify three configurations as examples utilizing the Simple Metering Cluster.

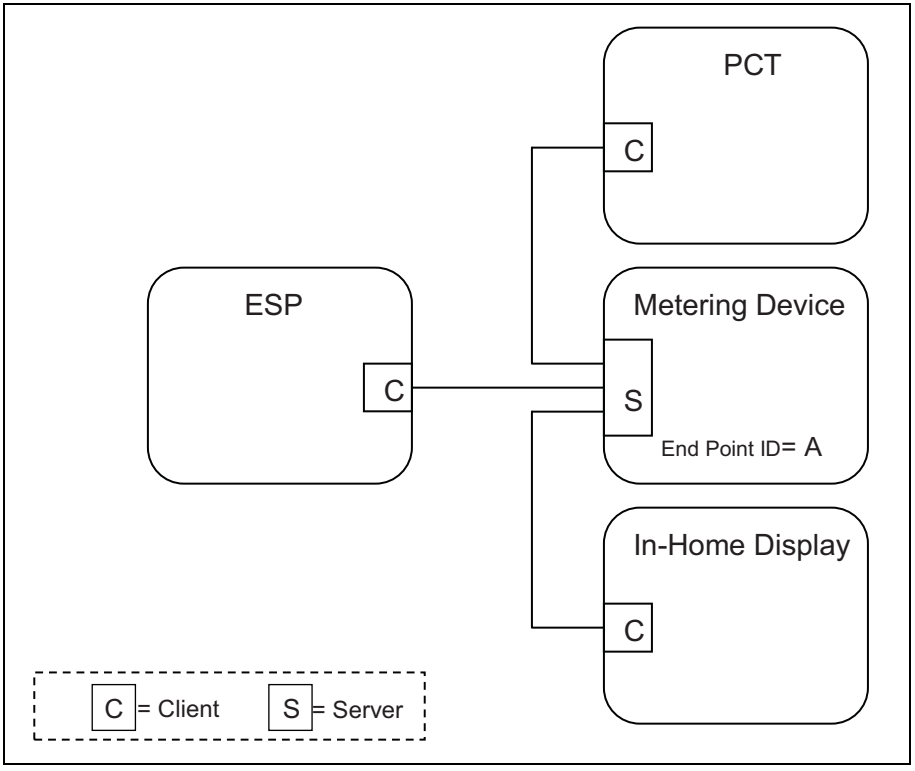

Figure D.9 Standalone ESP Model with Mains Powered Metering Device

The example above shown in Figure D.9, the metering device is the source of information provided via the Simple Metering Cluster Server.

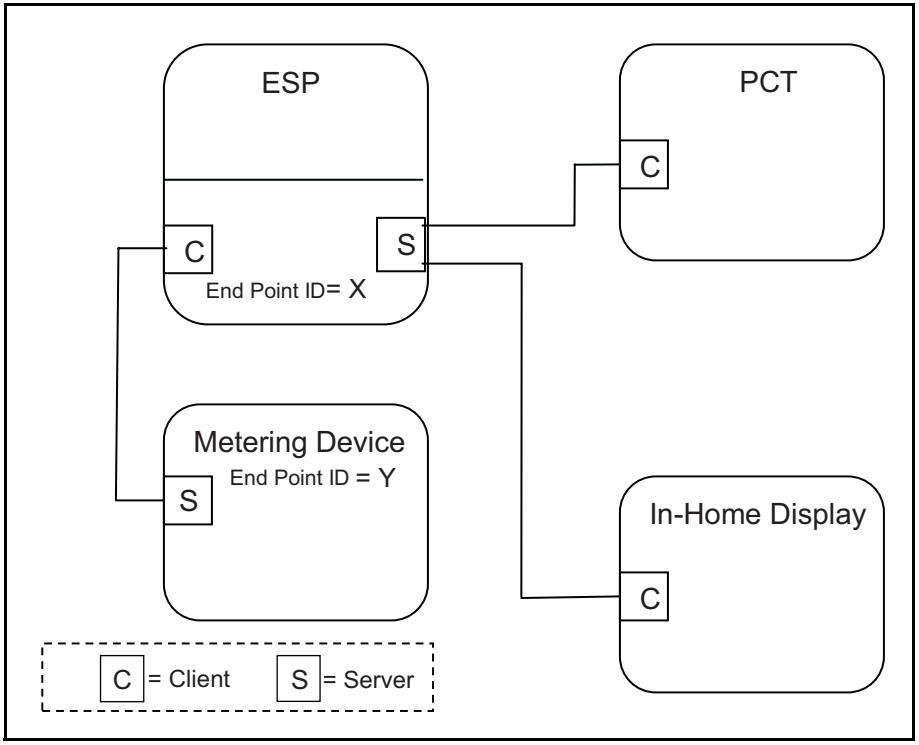

Figure D.10 Standalone ESP Model with Battery Powered Metering Device

In the example above shown in Figure D.10, the metering device is running on battery power and its duty cycle for providing information is unknown. It's expected the ESP will act like mirrored image or a mailbox (Client) for the metering device data, allowing other Smart Energy devices to gain access to the metering device's data (provided via an image of its Simple Metering Cluster).

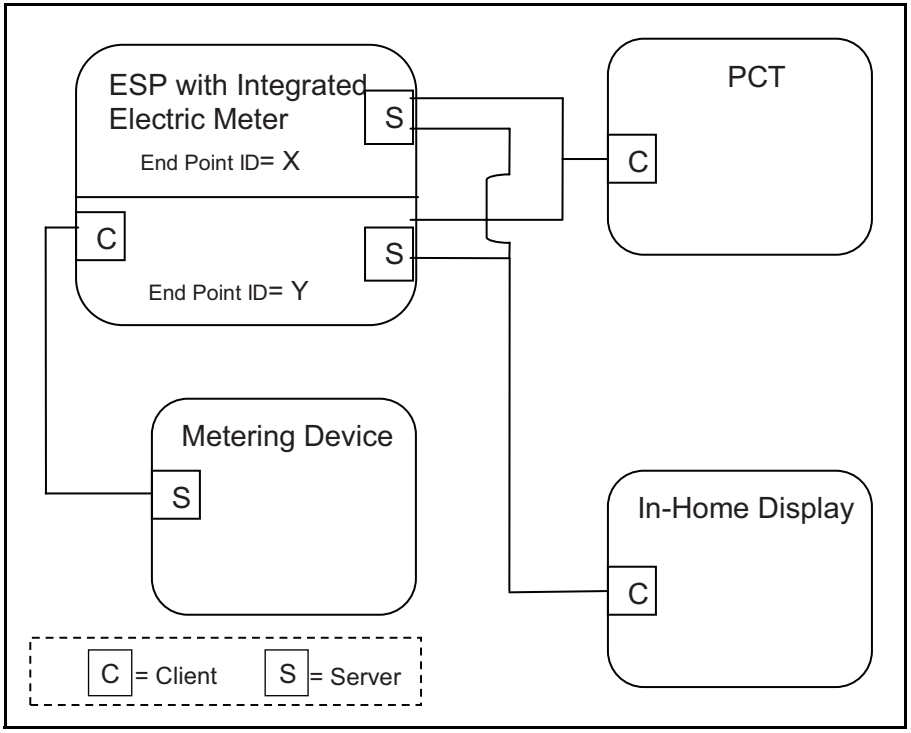

Figure D.11 ESP Model with Integrated Metering Device

In the example above shown in Figure D.11, much like the previous example in Figure D.10, where the external metering device is running on battery power and its duty cycle for providing information is unknown. It's expected the ESP will act like a Client side mailbox for the external metering device data, allowing other Smart Energy devices to gain access to the metering device's data (provided via an image of its Simple Metering Cluster). Since the ESP can also contain an integrated metering device where its information is also conveyed through the Simple Metering Cluster, each device (external metering device mailbox and integrated meter) will be available via independent EndPoint IDs. Other Smart Energy devices that need to access the information must understand the ESP cluster support by performing service discoveries. It can also identify if an Endpoint ID is a mailbox/mirror of a metering device by reading the *MeteringDeviceType* attribute (refer to sub-clause D.3.2.2.5.7).

In the above examples (Figure D.10 and Figure D.11), it's expected the ESP would perform Attribute Reads (or configure Attribute Reporting) and use the GetProfile command to receive the latest information whenever the Metering Device (EndPoint Z) wakes up. When received, the ESP will update its mailbox (EndPoint ID Y in Figure D.10 and Figure D.11) to reflect the latest data available.

Other Smart Energy devices can access EndPoint Y in the ESP to receive the latest information just as it would to access information in the ESP's integrated Electric meter (as in Figure D.11, EndPoint X) and other Metering devices (as in Figure D.9, EndPoint A).

D.3.2 Server

D.3.2.1 Dependencies

Subscribed reporting of Simple Metering attributes.

D.3.2.2 Attributes

For convenience, the attributes defined in this specification are arranged into sets of related attributes; each set can contain up to 256 attributes. Attribute identifiers are encoded such that the most significant Octet specifies the attribute set and the least significant Octet specifies the attribute within the set. The currently defined attribute sets are listed in the following Table D.10.

Table D.10 Simple Metering Attribute Sets

Attribute Set Identifier	Description
0x00	Reading Information Set
0x01	TOU Information Set
0x02	Meter Status
0x03	Formatting
0x04	ESP Historical Consumption
0x05	Load Profile Configuration
0x06	Supply Limit[a]
0x07 to 0xFF	Reserved[a]

a. CCB 974

D.3.2.2.1 Reading Information Set

The following set of attributes provides a remote access to the reading of the Electric, Gas, or Water metering device. A reading must support at least one register which is the actual total summation of the delivered quantity (kWh, m^3, ft^3, ccf, US gl).

Please note: In the following attributes, the term "Delivered" refers to the quantity of Energy, Gas, or Water that was delivered to the customer from the utility.

Likewise, the term “Received” refers to the quantity of Energy, Gas, or Water that was received by the utility from the customer.

Table D.11 Reading Information Attribute Set

Identifier	Name	Type	Range	Access	Default	Man./Opt.
0x00	CurrentSummationDelivered	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	M
0x01	CurrentSummationReceived	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x02	CurrentMaxDemandDelivered	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x03	CurrentMaxDemandReceived	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x04	DFTSummation	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x05	Daily Freeze Time	Unsigned 16 bit Integer	0x0000 to 0x183C	Read Only	0x0000	O
0x06	PowerFactor	Signed 8 bit Integer	-100 to +100	Read Only	0x00	O
0x07	ReadingSnapShotTime	UTCTime		Read Only	-	O
0x08	CurrentMaxDemandDeliveredTime	UTCTime		Read Only	-	O
0x09	CurrentMaxDemandReceivedTime	UTCTime		Read Only	-	O
0x10 to 0xFF	Reserved					

D.3.2.2.1.1 CurrentSummationDelivered Attribute

CurrentSummationDelivered represents the most recent summed value of Energy, Gas, or Water delivered and consumed in the premise.

CurrentSummationDelivered is mandatory and must be provided as part of the minimum data set to be provided by the metering device. *CurrentSummationDelivered* is updated continuously as new measurements are made.

D.3.2.2.1.2 CurrentSummationReceived Attribute

CurrentSummationReceived represents the most recent summed value of Energy, Gas, or Water generated and delivered from the premise. If optionally provided, *CurrentSummationReceived* is updated continuously as new measurements are made.

D.3.2.2.1.3 CurrentMaxDemandDelivered Attribute

CurrentMaxDemandDelivered represents the maximum demand or rate of delivered value of Energy, Gas, or Water being utilized at the premise. If optionally provided, *CurrentMaxDemandDelivered* is updated continuously as new measurements are made.

D.3.2.2.1.4 CurrentMaxDemandReceived Attribute

CurrentMaxDemandReceived represents the maximum demand or rate of received value of Energy, Gas, or Water being utilized by the utility. If optionally provided, *CurrentMaxDemandReceived* is updated continuously as new measurements are made.

D.3.2.2.1.5 DFTSummation Attribute

DFTSummation represents a snapshot of attribute *CurrentSummationDelivered* captured at the time indicated by attribute DailyFreezeTime. If optionally provided, *DFTSummation* is updated once every 24 hours and captured at the time set in sub-clause D.3.2.2.1.6.

D.3.2.2.1.6 DailyFreezeTime Attribute

DailyFreezeTime represents the time of day when DFTSummation is captured. *DailyFreezeTime* is an unsigned 16 bit value representing the hour and minutes for DFT. The byte usages are:

Bits 0 to 7: Range of 0 to 0x3C representing the number of minutes past the top of the hour.

Bits 8 to 15: Range of 0 to 0x17 representing the hour of the day (in 24 hour format).

D.3.2.2.1.7 PowerFactor Attribute

PowerFactor contains the Average Power Factor ratio in 1/100's. Valid values are 0 to 99.

D.3.2.2.1.8 ReadingSnapShotTime Attribute

The *ReadingSnapShotTime* attribute represents the last time all of the *CurrentSummationDelivered*, *CurrentSummationReceived*, *CurrentMaxDemandDelivered*, and *CurrentMaxDemandReceived* attributes that are supported by the device were updated.

D.3.2.2.1.9 CurrentMaxDemandDeliveredTime Attribute

The *CurrentMaxDemandDeliveredTime* attribute represents the time when *CurrentMaxDemandDelivered* reading was captured.

D.3.2.2.1.10 CurrentMaxDemandReceivedTime Attribute

The *CurrentMaxDemandReceivedTime* attribute represents the time when *CurrentMaxDemandReceived* reading was captured.

D.3.2.2.2 Summation TOU Information Set

The following set of attributes provides a remote access to the Electric, Gas, or Water metering device's Time of Use (TOU) readings.

Table D.12 TOU Information Attribute Set

0x00	CurrentTier1SummationDelivered	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x01	CurrentTier1SummationReceived	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x02	CurrentTier2SummationDelivered	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x03	CurrentTier2SummationReceived	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x04	CurrentTier3SummationDelivered	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFF	Read Only	-	O

Table D.12 TOU Information Attribute Set (Continued)

0x05	CurrentTier3SummationReceived	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x06	CurrentTier4SummationDelivered	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x07	CurrentTier4SummationReceived	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x08	CurrentTier5SummationDelivered	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x09	CurrentTier5SummationReceived	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x0A	CurrentTier6SummationDelivered	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x0B	CurrentTier6SummationReceived	Unsigned 48 bit Integer	0x000000000000 to 0xFFFFFFFFFFFF	Read Only	-	O
0x0C to 0xFF	Reserved					

D.3.2.2.2.1 CurrentTierNSummationDelivered Attributes

Attributes *CurrentTier1SummationDelivered* through *CurrentTier5SummationDelivered* represent the most recent summed value of Energy, Gas, or Water delivered to the premise (i.e delivered to the customer from the utility) at a specific price tier as defined by a TOU schedule or a real time pricing period. If optionally provided, attributes *CurrentTier1SummationDelivered* through *CurrentTier5SummationDelivered* are updated continuously as new measurements are made.

D.3.2.2.2.2 CurrentTierNSummationReceived Attributes

Attributes *CurrentTier1SummationReceived* through *CurrentTier5SummationReceived* represent the most recent summed value of Energy, Gas, or Water provided by the premise (i.e received by the utility from the customer) at a specific price tier as defined by a TOU schedule or a real time pricing period. If optionally provided, attributes *CurrentTier1SummationReceived*

through *CurrentTier5SummationReceived* are updated continuously as new measurements are made.

D.3.2.2.3 Meter Status Attribute

The Meter Status Attribute Set is defined in Table D.13.

Table D.13 Meter Status Attribute Set

Identifier	Name	Type	Range	Access	Default	Man. / Opt.
0x00	Status	8 bit BitMap	0x00 to 0xFF	Read Only	0x00	M
0x01 to 0xFF	Reserved					

D.3.2.2.3.1 Status Attribute

The *Status* attribute provides indicators reflecting the current error conditions found by the metering device. This attribute is an 8 bit field where when an individual bit is set, an error or warning condition exists. The behavior causing the setting or resetting each bit is device specific. In other words, the application within the metering device will determine and control when these settings are either set or cleared. Table D.14 lists the mapping of the bit is as follows:

Table D.14 Mapping of the Status Attribute

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Service Disconnect Open	Leak Detect	Power Quality	Power Failure	Tamper Detect	Low Battery	Check Meter

The definitions of the Status bits are:

Service Disconnect Open: Set to true when the service have been disconnected to this premise.

Leak Detect: Set to true when a leak have been detected.

Power Quality: Set to true if a power quality event have been detected such as a low voltage, high voltage.

Power Failure: Set to true during a power outage.

Tamper Detect: Set to true if a tamper event has been detected.

Low Battery: Set to true when the battery needs maintenance.

Check Meter: Set to true when a non fatal problem has been detected on the meter such as a measurement error, memory error, self check error.

D.3.2.2.4 Formatting

The following set of attributes provides the ratios and formatting hints required to transform the received summations, consumptions or demands/rates into displayable values. If the Multiplier and Divisor attribute values are non-zero, they are used in conjunction with the *SummationFormatting*, *ConsumptionFormatting*, and *DemandFormatting* attributes. Equations required to accomplish this task are defined below:

Summation = Summation received * Multiplier / Divisor
(formatted using *SummationFormatting*)

Consumption = Summation received * Multiplier / Divisor
(formatted using *ConsumptionFormatting*)

Demand = Demand received * Multiplier / Divisor
(formatted using *DemandFormatting*)

If the Multiplier and Divisor attribute values are zero, just the formatting hints defined in *SummationFormatting*, *ConsumptionFormatting*, and *DemandFormatting* attributes are used.

The following set of attributes provides the ratios and formatting hints required to transform the received summations, consumptions or demands/rates into displayable values. If the Multiplier and Divisor attribute values are non-zero, they are used in conjunction with the *SummationFormatting*, *ConsumptionFormatting*, and *DemandFormatting* attributes.

Equations required to accomplish this task are defined below:

text = summation received * Multiplier / Divisor
(formatted using *SummationFormatting* Attribute)

text = consumption received * Multiplier / Divisor
(formatted using *HistoricalConsumptionFormatting* Attribute)

text = demand received * Multiplier / Divisor
(formatted using *SummationFormatting* Attribute)

The summation received, consumption received and demand received variables used above can be replaced by any of the attributes listed in sub-clauses D.3.2.2.4.4, D.3.2.2.4.5 and D.3.2.2.4.6.

If the Multiplier and Divisor attribute values are zero, just the formatting hints defined in *SummationFormatting*, *HistoricalConsumptionFormatting*, and *DemandFormatting* attributes are used.

Following table shows examples which demonstrate the relation between these attributes.

Table D.15

Attribute	Example 1	Example 2	Example 3
Value as transmitted and received	52003	617	23629
UnitofMeasure	kWh	CCF	kWh
Multiplier	1	2	6
Divisor	1000	100	10000
Number of Digits to the left of the Decimal Point	5	4	5
Number of Digits to the right of the Decimal Point	0	2	3
Suppress leading zeros	False	False	True
Displayed value	00052	0012.34	14.177

The Consumption Formatting Attribute Set is defined in Table D.16.

Table D.16 Formatting Attribute Set

Identifier	Name	Type	Range	Access	Default	Man./ Opt.
0x00	UnitofMeasure	8-bit Enumeration	0x00 to 0xFF	Read Only	0x00	M
0x01	Multiplier	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	-	O
0x02	Divisor	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	-	O
0x03	SummationFormatting	8 bit BitMap	0x00 to 0xFF	Read Only	-	M
0x04	DemandFormatting	8 bit BitMap	0x00 to 0xFF	Read Only	-	O

Table D.16 Formatting Attribute Set (Continued)

Identifier	Name	Type	Range	Access	Default	Man./ Opt.
0x05	HistoricalConsumptionFormatting	8 bit BitMap	0x00 to 0xFF	Read Only	-	O
0x06	MeteringDeviceType	8 bit BitMap	0x00 to 0xFF	Read Only	-	M
0x07 to 0xFF	Reserved					

D.3.2.2.4.1 UnitofMeasure Attribute

UnitofMeasure provides a label for the Energy, Gas, or Water being measured by the metering device. The unit of measure apply to all summations, consumptions/ profile interval and demand/rate supported by this cluster. Other measurements such as the power factor are self describing. This Attribute is an 8 bit enumerated field. The bit descriptions for this Attribute are listed in Table D.17.

Table D.17 UnitofMeasure Attribute Enumerations

Values	Description
0x00	kW (kilo-Watts) & kWh (kilo-WattHours) in pure Binary format.
0x01	m^3 (Cubic Meter) & m^3/h (Cubic Meter per Hour) in pure Binary format.
0x02	ft^3 (Cubic Feet) & ft^3/h (Cubic Feet per Hour) in pure Binary format.
0x03	ccf ((100 or Centum) Cubic Feet) & ccf/h ((100 or Centum) Cubic Feet per Hour) in pure Binary format.
0x04	US gl (US Gallons) & US gl/h (US Gallons per Hour) in pure Binary format.
0x05	IMP gl (Imperial Gallons) & IMP gl/h (Imperial Gallons per Hour) in pure Binary format.
0x06	BTUs & BTU/h in pure Binary format.
0x07	Liters & l/h (Liters per Hour) in pure Binary format
0x08	kPA(gauge) in pure Binary format.
0x09	kPA(absolute) in pure Binary format.
0x0A to 0x7F	Reserved for future use.
0x80	kW (kilo-Watts) & kWh (kilo-WattHours) in BCD format
0x81	m^3 (Cubic Meter) & m^3/h (Cubic Meter per Hour) in BCD format
0x82	ft^3 (Cubic Feet) & ft^3/h (Cubic Feet per Hour) in BCD format

Table D.17 UnitofMeasure Attribute Enumerations (Continued)

Values	Description
0x83	ccf ((100 or Centum) Cubic Feet) & ccf/h ((100 or Centum) Cubic Feet per Hour) in BCD format
0x84	US gl (US Gallons) & US gl/h (US Gallons per Hour) in BCD format
0x85	IMP gl (Imperial Gallons) & IMP gl/h (Imperial Gallons per Hour) in BCD format
0x86	BTUs & BTU/h in BCD format
0x87	Liters & l/h (Liters per Hour) in BCD format
0x88	kPA(gauge) in BCD format.
0x89	kPA(absolute) in BCD format.
0x8A to 0xFF	Reserved for future use

Please Note: When using BCD for meter reads, the values A to F are special values or indicators denoting “Opens”, “Shorts”, and etc. conditions when reading meter register hardware. Any SE device displaying the BCD based values to end users should use a non-decimal value to replace the A to F. In other words, a device could use an “*” in place of the special values or indicators.

D.3.2.2.4.2 Multiplier Attribute

Multiplier provides a value to be multiplied against a raw or uncompensated sensor count of Energy, Gas, or Water being measured by the metering device. If present, this attribute must be applied against all summation, consumption and demand values to derive the delivered and received values expressed in the unit of measure specified. This attribute must be used in conjunction with the Divisor Attribute.

D.3.2.2.4.3 Divisor Attribute

Divisor provides a value to divide the results of applying the Multiplier Attribute against a raw or uncompensated sensor count of Energy, Gas, or Water being measured by the metering device. If present, this attribute must be applied against all summation, consumption and demand values to derive the delivered and received values expressed in the unit of measure specified. This attribute must be used in conjunction with the Multiplier Attribute.

D.3.2.2.4.4 SummationFormatting Attribute

SummationFormatting provides a method to properly decipher the number of digits and the decimal location of the values found in the Summation Information Set of attributes. This attribute is to be decoded as follows:

Bits 0 to 2: Number of Digits to the right of the Decimal Point.

Bits 3 to 6: Number of Digits to the left of the Decimal Point.

Bit 7: If set, suppress leading zeros.

This attribute shall be used against the following attributes:

CurrentSummationDelivered

CurrentSummationReceived

CurrentTier1SummationDelivered

CurrentTier1SummationReceived

CurrentTier2SummationDelivered

CurrentTier2SummationReceived

CurrentTier3SummationDelivered

CurrentTier3SummationReceived

CurrentTier4SummationDelivered

CurrentTier4SummationReceived

CurrentTier5SummationDelivered

CurrentTier5SummationReceived

DFTSummation

D.3.2.2.4.5 DemandFormatting Attribute

DemandFormatting provides a method to properly decipher the number of digits and the decimal location of the values found in the Demand related attributes. This attribute is to be decoded as follows:

Bits 0 to 2: Number of Digits to the right of the Decimal Point.

Bits 3 to 6: Number of Digits to the left of the Decimal Point.

Bit 7: If set, suppress leading zeros.

This attribute shall be used against the following attributes:

CurrentMaxDemandDelivered

CurrentMaxDemandReceived

InstantaneousDemand

D.3.2.2.4.6 HistoricalConsumptionFormatting Attribute

HistoricalConsumptionFormatting provides a method to properly decipher the number of digits and the decimal location of the values found in the ESP Historical Consumption Set of attributes. This attribute is to be decoded as follows:

Bits 0 to 2: Number of Digits to the right of the Decimal Point.

Bits 3 to 6: Number of Digits to the left of the Decimal Point.

Bit 7: If set, suppress leading zeros.

This attribute shall be used against the following attributes:

CurrentDayConsumptionDelivered

CurrentDayConsumptionReceived

PreviousDayConsumptionDelivered

PreviousDayConsumptionReceived

CurrentPartialProfileIntervalValue

Intervals

D.3.2.2.4.7 MeteringDeviceType Attribute

MeteringDeviceType provides a label for identifying the type of metering device present. The attribute are enumerated values representing Energy, Gas, Water, Thermal, Heat, Cooling, and mirrored metering devices. The defined values are represented in Table D.18.

Table D.18 MeteringDeviceType Attribute Enumerations

Values	Description
0	Electric Metering
1	Gas Metering
2	Water Metering
3	Thermal Metering (deprecated)[a]
4	Pressure Metering
5	Heat Metering[b]
6	Cooling Metering[c]
7[d] to 127	Reserved for future growth

Table D.18 MeteringDeviceType Attribute Enumerations (Continued)

Values	Description
128	Mirrored Gas Metering
129	Mirrored Water Metering
130	Mirrored Thermal Metering
131	Mirrored Pressure Metering
132	Mirrored Heat Metering[e]
133	Mirrored Cooling Metering[f]
134 to 255	Reserved for future growth

a. CCB 986
b. CCB 986
c. CCB 986
d. CCB 986
e. CCB 986
f. CCB 986

Note: Heat and cooling meters are used for measurement and billing of heat (and cooling) delivered through liquid (water) based central heating systems. The consumers are typically billed by the kWh, calculated from the flow and the temperatures in and out.[19]

D.3.2.2.5 ESP Historical Consumption

The ESP Historical Attribute Set is defined in Table D.19.

19. CCB 986

Table D.19 ESP Historical Attribute Set

Identifier	Name	Type	Range	Access	Default	Man./ Opt.
0x00	InstantaneousDemand	Signed 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	0x00	O
0x01	CurrentDayConsumptionDelivered	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	-	O
0x02	CurrentDayConsumptionReceived	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	-	O
0x03	PreviousDayConsumptionDelivered	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	-	O
0x04	PreviousDayConsumptionReceived	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	-	O
0x05	CurrentPartialProfileIntervalStartTimeDelivered	UTCTime		Read Only	-	O
0x06	CurrentPartialProfileIntervalStartTimeReceived	UTCTime	0x000000 to 0xFFFFFF	Read Only	-	O
0x07	CurrentPartialProfileIntervalValueDelivered	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	-[a]	O[b]
0x08	CurrentPartialProfileIntervalValueReceived	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read Only	-[c]	O[d]
0x09 to 0xFF	Reserved					

a. CCB 982

b. CCB 982

c. CCB 982

d. CCB 982

D.3.2.2.5.1 InstantaneousDemand Attribute

InstantaneousDemand represents the current Demand of Energy, Gas, or Water delivered or received at the premise. Positive values indicate Demand delivered to

the premise where negative values indicate demand received from the premise. *InstantaneousDemand* is updated continuously as new measurements are made. The frequency of updates to this field is specific to the metering device, but should be within the range of once every second to once every 5 seconds.

D.3.2.2.5.2 CurrentDayConsumptionDelivered Attribute

CurrentDayConsumptionDelivered represents the summed value of Energy, Gas, or Water generated and delivered to the premise since midnight local time. If optionally provided, *CurrentDayConsumptionDelivered* is updated continuously as new measurements are made.

D.3.2.2.5.3 CurrentDayConsumptionReceived Attribute

CurrentDayConsumptionReceived represents the summed value of Energy, Gas, or Water generated and received from the premise since midnight local time. If optionally provided, *CurrentDayConsumptionReceived* is updated continuously as new measurements are made.

D.3.2.2.5.4 PreviousDayConsumptionDelivered Attribute

PreviousDayConsumptionDelivered represents the summed value of Energy, Gas, or Water generated and delivered to the premise within the previous 24 hour period starting at midnight local time. If optionally provided, *CurrentDayConsumptionDelivered* is updated every midnight local time.

D.3.2.2.5.5 PreviousDayConsumptionReceived Attribute

PreviousDayConsumptionReceived represents the summed value of Energy, Gas, or Water generated and received from the premise within the previous 24 hour period starting at midnight local time. If optionally provided, *CurrentDayConsumptionReceived* is updated is updated every midnight local time.

D.3.2.2.5.6 CurrentPartialProfileIntervalStartTimeDelivered Attribute

CurrentPartialProfileIntervalStartTimeDelivered represents the start time of the current Load Profile interval being accumulated for commodity delivered.

D.3.2.2.5.7 CurrentPartialProfileIntervalStartTimeReceived Attribute

CurrentPartialProfileIntervalStartTimeReceived represents the start time of the current Load Profile interval being accumulated for commodity received.

D.3.2.2.5.8 CurrentPartialProfileIntervalValueDelivered Attribute

CurrentPartialProfileIntervalValueDelivered represents the value of the current Load Profile interval being accumulated for commodity delivered.

D.3.2.2.5.9 CurrentPartialProfileIntervalValueReceived Attribute

CurrentPartialProfileIntervalValueReceived represents the value of the current Load Profile interval being accumulated for commodity received.

D.3.2.2.6 Load Profile Configuration

The Load Profile Configuration Attribute Set is defined in Table D.20.

Table D.20 Load Profile Configuration Attribute Set

Identifier	Name	Type	Range	Access	Default	Man./ Opt.
0x00	MaxNumberOfPeriodsDelivered	Unsigned 8 bit Integer	0x00 to 0xFF	Read Only	0x18	O
0x01 to 0xFF	Reserved					

D.3.2.2.6.1 MaxNumberOfPeriodsDelivered Attribute

MaxNumberofPeriodsDelivered represents the maximum number of intervals the device is capable of returning in one Get Profile Response command. It is required *MaxNumberofPeriodsDelivered* fit within the default Fragmentation ASDU size of 128 bytes, or an optionally agreed upon larger Fragmentation ASDU size supported by both devices. Please refer to sub-clause 5.3.8 for further details on Fragmentation settings.[20]

D.3.2.2.7 Supply Limit Attributes

This set of attributes is used to implement a “Supply Capacity Limit” program where the demand at a premise is limited to a preset consumption level over a preset period of time. Should this preset limit be exceeded the meter could interrupt supply to the premise or to devices within the premise. The supply limit information in this attribute set can be used by In-Premise displays, PCTs, or other devices to display a warning when the supply limit is being approached. The Supply Limit Attribute Set is defined in Table D.21.

20. CCB 983

Table D.21 Supply Limit Attribute Set

Identifier	Name	Type	Range	Access	Default	Man / Opt
0x00	CurrentDemandDelivered	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read only		O
0x01	DemandLimit	Unsigned 24 bit Integer	0x000000 to 0xFFFFFF	Read only		O
0x02	DemandIntegrationPeriod	Unsigned 8 bit Integer	0x01 to 0xFF	Read only	-	O
0x03	NumberOfDemandSubintervals	Unsigned 8 bit Integer	0x01 to 0xFF	Read only	-	O
0x04 - 0xFF	Reserved					

D.3.2.2.7.1 CurrentDemandDelivered

CurrentDemandDelivered represents the current Demand of Energy, Gas, or Water delivered at the premise. *CurrentDemandDelivered* may be continuously updated as new measurements are acquired, but at a minimum *CurrentDemandDelivered* must be updated at the end of each integration sub-period, which can be obtained by dividing the *DemandIntegrationPeriod* by the *NumberOfDemandSubintervals*.

This attribute shall be adjusted using the Multiplier and Divisor attributes found in the Formatting Attribute Set and can be formatted using the *DemandFormatting* attribute. The final result represents an engineering value in the unit defined by the *UnitofMeasure* attribute.

D.3.2.2.7.2 DemandLimit

DemandLimit reflects the current supply demand limit set in the meter. This value can be compared to the *CurrentDemandDelivered* attribute to understand if limits are being approached or exceeded.

Adjustment and formatting of this attribute follow the same rules as the *CurrentDemandDelivered*.

D.3.2.2.7.3 DemandIntegrationPeriod

DemandIntegrationPeriod is the number of minutes over which the *CurrentDemandDelivered* attribute is calculated. Valid range is 0x01 to 0xFF. 0x00 is a reserved value.

D.3.2.2.7.4 NumberOfDemandSubintervals

NumberOfDemandSubintervals represents the number of subintervals used within the *DemandIntegrationPeriod*. The subinterval duration (in minutes) is obtained by dividing the *DemandIntegrationPeriod* by the *NumberOfDemandSubintervals*. The *CurrentDemandDelivered* attribute is updated at the each of each subinterval. Valid range is 0x01 to 0xFF. 0x00 is a reserved value.

As a Rolling Demand example, *DemandIntegrationPeriod* could be set at 30 (for 30 minute period) and *NumberOfDemandSubintervals* could be set for 6. This would provide 5 minute (30/6 = 5) subinterval periods.

As a Block Demand example, *DemandIntegrationPeriod* could be set at 30 (for 30 minute period) and *NumberOfDemandSubintervals* could be set for 1. This would provide a single 30 minute subinterval period[21].

D.3.2.3 Server Commands

D.3.2.3.1 Commands Generated

The command IDs generated by the Simple Metering server cluster are listed in Table D.22.

Table D.22 Generated Command IDs for the Simple Metering Server

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Get Profile Response	O
0x01	Request Mirror	O
0x02	Remove Mirror	O
0x03 – 0xff	Reserved	

D.3.2.3.1.1 Get Profile Response Command

D.3.2.3.1.1.1 Payload Format

The Get Profile Response command payload shall be formatted as illustrated in Figure D.12.

21. CCB 974

Octets	4	1	1	1	Variable
Data Type	UTC Time	8-bit Enumeration	8-bit Enumeration	Unsigned 8-bit Integer	Series of Unsigned 24 bit Integers
Field Name	EndTime	Status	ProfileIntervalPeriod	NumberOfPeriodsDelivered	Intervals

Figure D.12 Get Profile Response Command Payload

D.3.2.3.1.1.1 Payload Details

EndTime: 32 bit value (in UTC) representing the end time of the most chronologically recent interval being requested. Example: Data collected from 2:00 PM to 3:00 PM would be specified as a 3:00 PM interval (end time). It is important to note that the current interval accumulating is not included in most recent block but can be retrieved using the CurrentPartialProfileIntervalValue attribute.

Status: Table D.23 lists the valid values returned in the Status field.

Table D.23 Status Field Values

Value	Description
0x00	Success
0x01	Undefined Interval Channel requested
0x02	Interval Channel not supported
0x03	Invalid End Time
0x04	More periods requested than can be returned
0x05	No intervals available for the requested time
0x06 to 0xFF	Reserved for future use.

ProfileIntervalPeriod: Represents the interval or time frame used to capture metered Energy, Gas, and Water consumption for profiling purposes. *ProfileIntervalPeriod* is an enumerated field representing the following timeframes listed in Table D.24:

Table D.24 ProfileIntervalPeriod Timeframes

Enumerated Value	Timeframe
0	Daily
1	60 minutes
2	30 minutes
3	15 minutes
4	10 minutes
5	7.5 minutes
6	5 minutes
7	2.5 minutes
8 to 255	Reserved

NumberofPeriodsDelivered: Represents the number of intervals the device is returning. Please note the number of periods returned in the Get Profile Response command can be calculated when the packets are received and can replace the usage of this field. The intent is to provide this information as a convenience.

Intervals: Series of interval data captured using the period specified by the ProfileIntervalPeriod field. The content of the interval data depend of the type of information requested using the Channel field in the Get Profile Command. 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.

D.3.2.3.1.1.2 When Generated

This command is generated when the Client command GetProfile is received. Please refer to sub-clause D.3.2.4.2.

D.3.2.3.1.2 Request Mirror Command

This command is used to request the ESP to mirror Metering Device data.

D.3.2.3.1.2.1 Payload Details

There are no fields for this command.

D.3.2.3.1.2.2 Effect on Receipt

On receipt of this command, the Server shall send a RequstMirrorReponse command (see sub-clause D.3.2.4.3).

D.3.2.3.1.3 Remove Mirror Command

This command is used to request the ESP to remove its mirror of Metering Device data.

D.3.2.3.1.3.1 Payload Details

There are no fields for this command.

D.3.2.3.1.3.2 Effect on Receipt

On receipt of this command, the Server shall send a MirrorRemoved command (see sub-clause D.3.2.4.4).

D.3.2.4 Client Commands

D.3.2.4.1 Commands Generated

The command IDs generated by the Simple Metering client cluster are listed in Table D.25.

Table D.25 Generated Command IDs for the Simple Metering Client

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Get Profile	O
0x01	Request Mirror Response	O
0x02	Remove Mirror	O
0x03 – 0xff	Reserved	

D.3.2.4.2 Get Profile Command

The Get Profile command payload shall be formatted as illustrated in Table D.26.

Table D.26 GetProfile Payload

Octets	1	4	1
Data Type	Unsigned 8-bit Integer	UTCTime	Unsigned 8-bit Integer
Field Name	Interval Channel	End Time	NumberOfPeriods

D.3.2.4.2.1 Payload Details

Interval Channel: Enumerated value used to select the quantity of interest returned by the Get Profile Response Command. The Interval Channel values are listed in Table D.27.

Table D.27 Interval Channel Values

Bit	Description
0	Consumption Delivered
1	Consumption Received
2 to 255	Not used

EndTime: 32 bit value (in UTCTime) used to select an Intervals block from all the Intervals blocks available. The Intervals block returned is the most recent block with its EndTime equal or older to the one provided. The most recent Intervals block is requested using an End Time set to 0x00000000, subsequent Intervals block are requested using an End time set to the EndTime of the previous block - (number of intervals of the previous block * ProfileIntervalPeriod).

NumberofPeriods: Represents the number of intervals being requested. This value can't exceed the size stipulated in the *MaxNumberOfPeriodsDelivered* attribute. If more intervals are requested than can be delivered, the GetProfileResponse will return the number of intervals equal to *MaxNumberOfPeriodsDelivered*. If fewer intervals available for the time period, only those available are returned.

D.3.2.4.2.2 When Generated

The GetProfile command is generated when a client device wishes to retrieve a list of captured Energy, Gas or water consumption for profiling purposes. Due to the potentially large amount of profile data available, the client device should

store previously gathered data and only request the most current data. When initially gathering significant amounts of historical interval data, the GetProfile command should not be issued any more frequently than 7.5 seconds to prevent overwhelming the ZigBee network.

D.3.2.4.2.3 Command Processing Response

If success or failure occurs in recognizing or processing the payload of the GetProfile command, the appropriate enumerated ZCL status (as referenced in the ZCL Cluster Library specification) will be returned.

D.3.2.4.2.4 Effect on Receipt

On receipt of this command, the device shall send a GetProfileReponse command (see sub-clause D.3.2.3.1.1).

D.3.2.4.3 Request Mirror Response Command

The Request Mirror Response Command allows the ESP to inform a sleepy Metering Device it has the ability to store and mirror its data.

D.3.2.4.3.1 Payload Format

The Request Mirror Response command payload shall be formatted as illustrated in Figure D.13

Octets	**2**
Data Type	Unsigned 16-bit Integer
Field Name	EndPoint ID

Figure D.13 Request Mirror Response Command Payload

D.3.2.4.3.1.1 Payload Details

EndPoint ID: 16 Bit Unsigned Integer indicating the End Point ID to contain the Metering Devices meter data. Valid End Point ID values are 0x0001 to 0x00F0. If the ESP is unable to mirror the Metering Device data, EndPoint ID shall be returned as 0xFFFF. All other EndPoint ID values are reserved. If valid, the Metering device shall use the EndPoint ID to forward its metered data.

D.3.2.4.4 Mirror Removed Command

The Mirror Removed Command allows the ESP to inform a sleepy Metering Device mirroring support has been removed or halted.

D.3.2.4.4.1 Payload Format

The Mirror Removed command payload shall be formatted as illustrated in Figure D.14

Octets	2
Data Type	Unsigned 16-bit Integer
Field Name	Removed EndPoint ID

Figure D.14 Mirror Removed Command Payload

D.3.2.4.4.1.1 Payload Details

Removed EndPoint ID: 16 Bit Unsigned Integer indicating the End Point ID previously containing the Metering Devices meter data.

D.3.3 Simple Meter Application Guidelines

D.3.3.1 Attribute Reporting

Attribute reporting may be used for sending information in the Reading Information, TOU Information, Meter Status, ESP Historical Consumption attribute sets.

D.3.3.2 Fast Polling or Reporting for Monitoring Energy Savings

Client devices, such as an energy gateway, smart thermostat, or in-home displays can monitor changes to energy saving settings within the premise and give users near real time feedback and results. The Simple Meter cluster can support this by using Attribute Reporting and sending updates at a much faster rate for a short period of time. Client devices can also perform a series of Attribute reads to accomplish the same task. In either case, requests or updates shall be limited to a maximum rate of once every two seconds for a maximum period of 15 minutes. These limitations are required to ensure Smart Energy profile based devices do not waste available bandwidth or prevent other operations within the premise.

D.3.3.3 Metering Data Updates

The frequency and timeliness of updating metering data contained in the Simple Metering Cluster Attributes and Profile Intervals is up to the individual Metering device manufacturer’s capabilities. As a best practice recommendation, updates of

the metering data should not cause delivery of the information to end devices more often than once every 30 seconds. End devices should also not request information more often than once every 30 seconds.

D.4 Price Cluster

D.4.1 Overview

The Price Cluster provides the mechanism for communicating Gas, Energy, or Water pricing information within the premise. This pricing information is distributed to the ESP from either the utilities or from regional energy providers. The ESP conveys the information (via the Price Cluster mechanisms) to both Smart Energy devices in secure method and/or optionally conveys it anonymously in an unsecure to very simple devices that may not be part of the Smart Energy network. The mechanism for sending anonymous information is called the Anonymous Inter-PAN transmission mechanism and is outlined in Annex B.

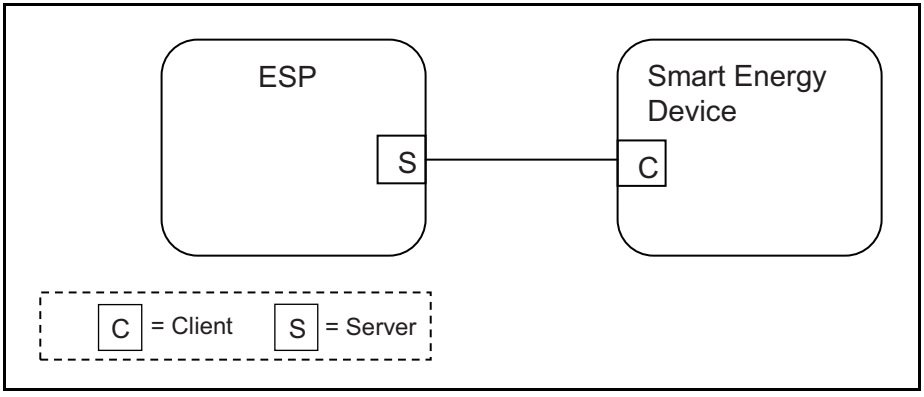

Figure D.15 Price Cluster Client Server Example

Please note the ESP is defined as the Server due to its role in acting as the proxy for upstream price management systems and subsequent data stores.

D.4.2 Server

D.4.2.1 Dependencies

- Events carried using this cluster include a timestamp with the assumption that target devices maintain a real time clock. Devices can acquire and synchronize their internal clocks via the ZCL Time server.

- If a device does not support a real time clock it is assumed that the device will interpret and utilize the “Start Now” value within the Time field.
- Anonymous Inter-PAN transmission mechanism outlined in Annex B.

D.4.2.2 Server Cluster Attributes

Table D.28 Price Server Cluster Attributes

Identifier	Name	Type	Length	Access	Default	Mandatory / Optional
0x0000	Tier1PriceLabel	Octet String	0 to 12 Octets	Read/ Write	"Tier 1"	O
0x0001	Tier2PriceLabel	Octet String	0 to 12 Octets	Read/ Write	"Tier 2"	O
0x0002	Tier3PriceLabel	Octet String	0 to 12 Octets	Read/ Write	"Tier 3"	O
0x0003	Tier4PriceLabel	Octet String	0 to 12 Octets	Read/ Write	"Tier 4"	O
0x0004	Tier5PriceLabel	Octet String	0 to 12 Octets	Read/ Write	"Tier 5"	O
0x0005	Tier6PriceLabel	Octet String	0 to 12 Octets	Read/ Write	"Tier 6"	O
0x0006 to 0xFFFF	Reserved					

D.4.2.2.1 TierNPriceLabel Attributes

The *TierNPriceLabel* attributes provide a method for utilities to assign a label to the Price Tier declared within the Publish Price command. The *TierNPriceLabel* attributes are a ZCL Octet String field capable of storing a 12 character string (the first Octet indicates length) encoded in the UTF-8 format. Example Tier Price Labels are "Normal", "Shoulder", "Peak", "Real Time Pricing" and "Critical Peak".

D.4.2.3 Commands Received

The server side of the Price cluster is capable of receiving the commands listed in Table D.29.

Table D.29 Received Command IDs for the Price Cluster

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Get Current Price	M
0x01	Get Scheduled Prices	O
0x02 – 0xff	Reserved	

D.4.2.3.1 Get Current Price Command

This command initiates a Publish Price command (see sub-clause D.4.2.4.1) for the current time.

D.4.2.3.1.1 Payload Format

The payload of the Get Current Price command is formatted as shown in Figure D.16:

Octets	1
Data Type	Unsigned 8 bit integer
Field Name	Command Options

Figure D.16 The Format of the Get Current Price Command Payload

D.4.2.3.1.1.1 Payload Details

The Command Options Field: The command options field is 8 Bits in length and is formatted as a bit field as shown in Figure D.17:

Bits	0	1 to 7
Field Name	Requestor Rx On When Idle	Reserved

Figure D.17 Get Current Price Command Options Field

The Requestor Rx On When Idle sub-field: The Requestor Rx On When Idle sub-field has a value of 1 if the requestor's receiver may be, for all practical purposes, enabled when the device is not actively transmitting, thereby making it

very likely that regular broadcasts of pricing information will be received by this device, and 0 otherwise.

A device that publishes price information may use the value of this bit, as received from requestors in its neighborhood, to determine publishing policy. For example, if a device makes a request for current pricing information and the requestor Rx on when idle sub-field of the GetCurrentPrice command payload has a value of 1 (indicating that the device will be likely to receive regular price messages), then the receiving device may store information about the requestor and use it in future publishing operations.

D.4.2.3.1.2 Effect on Receipt

On receipt of this command, the device shall send a Publish Price command (sub-clause D.4.2.4.1) for the currently scheduled time. Please note: The PublishPrice command is sent out on the network from which the GetCurrentPrice command was received (either the Inter-Pan or SE network). Example: If the GetCurrentPrice command is received on the Inter-Pan network, the ESP shall respond on the Inter-Pan. If the GetCurrentPrice command is received on the SE Network, the ESP shall respond to the device requesting the pricing information.

D.4.2.3.2 Get Scheduled Prices Command

This command initiates a Publish Price command (see sub-clause D.4.2.4.1) for all currently scheduled times. A server device shall be capable of storing five scheduled price events at a minimum.

D.4.2.3.2.1 Payload Details

The Get Scheduled Prices command payload shall be formatted as illustrated in Figure D.18:

Octets	**4**	**1**
Data Type	UTCTime	Unsigned8-bit integer
Field Name	Start Time (M)	Number of Events (M)

Figure D.18 Format of the Get Scheduled Prices Command Payload

Start Time (mandatory): UTC Timestamp representing the starting time for any scheduled pricing events to be re-sent.

Number of Events (mandatory): Represents the maximum number of events to be sent. A value of 0 would indicate all available events are to be returned.

D.4.2.3.2.2 When Generated

This command is generated when the client device wishes to verify the available Price Events or after a loss of power/reset occurs and the client device needs to recover currently active or scheduled Price Events.

D.4.2.3.2.3 Effect on Receipt

On receipt of this command, the device shall send a Publish Price command (see sub-clause D.4.2.4.1) for all currently scheduled price events. Please note: The PublishPrice command is sent out on the network from which the GetScheduledPrices command was received (either the Inter-Pan or SE network). Example: If the GetScheduledPrices command is received on the Inter-Pan network, the ESP shall respond on the Inter-Pan. If the GetCurrentPrice command is received on the SE Network, the ESP shall respond to the device requesting the pricing information.

D.4.2.4 Commands Generated

The server side of the Price cluster is capable of generating the commands listed in Table D.30.

Table D.30 Generated Command IDs for the Price Cluster

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Publish Price	M
0x01 – 0xff	Reserved	

D.4.2.4.1 Publish Price Command

The Publish Price command is generated in response to receiving a Get Current Price command (see sub-clause D.4.2.3.1), a Get Scheduled Prices command (see sub-clause D.4.2.3.2), or when an update to the pricing information is available from the commodity provider. When a Get Current Price or Get Scheduled Prices command is received over a ZigBee Smart Energy network, the Publish Price command should be sent unicast to the requester. In the case of an update to the pricing information from the commodity provider, the Publish Price command should be unicast to all individually registered devices implementing the Price Cluster on the ZigBee Smart Energy network. When responding to a request via the Inter-PAN SAP, the Publish Price command should be broadcast to the PAN of the requester after a random delay between 0 and 0.5 seconds, to avoid a potential broadcast storm of packets.

Devices capable of receiving this command must be capable of storing and supporting at least two pricing information instances, the current active price and the next price. By supporting at least two pricing information instances, receiving devices will allow the Publish Price command generator to publish the next pricing information during the current pricing period.

Nested and overlapping Publish Price commands are not allowed. The current active price will be replaced if new price information is received by the ESP.

D.4.2.4.1.1 Payload Format

The PublishPrice command payload shall be formatted as illustrated in Figure D.19.

Octets	4	0-12	4	4	1	2	1
Data Type	Unsigned 32 bit Integer	Octet String	Unsigned 32 bit Integer	UTCTime	8 bits enumeration	Unsigned 16 bit Integer	8 bit BitMap
Field Name	Provider ID (M)	Rate Label (M)	Issuer Event ID (M)	Current Time (M)	Unit of Measure (M)	Currency (M)	Price Trailing Digit & Price Tier (M)

Octets	1	4	2	4	1	4	1
Data Type	8 bit BitMap	UTCTime	Unsigned 16 bit Integer	Unsigned 32 bit Integer	Unsigned 8 bit Integer	Unsigned 32 bit Integer	Unsigned 8 bit Integer
Field Name	Number of Price Tiers & Register Tier (M)	Start Time (M)	Duration In Minutes (M)	Price (M)	Price Ratio (O)	Generation Price (O)	Generation Price Ratio (O)

Octets	4	1	1
Data Type	Unsigned 32 bit Integer	8 bits enumeration	8 bit BitMap
Field Name	Alternate Cost Delivered (O)[a]	Alternate Cost Unit (O)[b]	Alternate Cost Trailing Digit(O)[c]

a. CCB 973

b. CCB 973

c. CCB 973

Figure D.19 Format of the Publish Price Command Payload

Provider ID (mandatory): An unsigned 32 bit field containing a unique identifier for the commodity provider. This field is thought to be useful in deregulated markets where multiple commodity providers may be available.

Rate Label (mandatory): A ZCL Octet String field capable of storing a 12 character string (the first Octet indicates length) containing commodity provider-

specific information regarding the current billing rate. The String shall be encoded in the UTF-8 format. This field is thought to be useful when a commodity provider may have multiple pricing plans.

Issuer Event ID (mandatory): Unique identifier generated by the commodity provider. When new pricing information is provided that replaces older pricing information for the same time period, this 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 or a UTC based time stamp (UTCTime data type) identifying when the Publish Price command was issued. Thus, newer pricing information will have a value in the Issuer Event ID field that is larger than older pricing information.

Current Time (mandatory): A UTCTime field containing the current time as determined by the device. This field is thought to be useful to provide an extra value-added feature for the broadcast price signals.

Unit of Measure (mandatory): An 8 bit enumeration field identifying the commodity as well as its base unit of measure. The enumeration used for this field shall match one of the UnitOfMeasure values using a pure Binary format as defined in the Simple Metering cluster (see sub-clause D.3.2.2.5.1).

Currency (mandatory): An unsigned 16 bit field containing identifying information concerning the local unit of currency used in the price field. This field is thought to be useful for displaying the appropriate symbol for a currency (i.e.: $).

The value of the currency field should match the values defined by ISO 4217.

Price Trailing Digit and Price Tier (mandatory): An 8 bit field used to determine where the decimal point is located in the price field and to indicate the current pricing tier as chosen by the commodity provider. The most significant nibble is the Trailing Digit sub-field which indicates the number of digits to the right of the decimal point. The least significant nibble is an enumerated field containing the current Price Tier. Valid values for the Price Tier sub-field are from 1 to 6 reflecting the least expensive tier (1) to the most expensive tier (6). All other values for the Price Tier are reserved. This sub-field also references the associated *TiernPriceLabel* attribute assigned to the Price Tier. Table D.31 depicts the assignments:

Table D.31 Price Tier Sub-field Enumerations

Enumerated Value	Price Tier
0x0	No Tier Related
0x1	Reference *Tier1PriceLabel*
0x2	Reference *Tier2PriceLabel*
0x3	Reference *Tier3PriceLabel*

Table D.31 Price Tier Sub-field Enumerations

0x4	Reference *Tier4PriceLabel*
0x5	Reference *Tier5PriceLabel*
0x6	Reference *Tier6PriceLabel*
0x7 to 0xF	Reserved

Number of Price Tiers & Register Tier (mandatory): An 8 bit BitMap where the most significant nibble is an enumerated sub-field representing the maximum number of price tiers available, and the least significant nibble is an enumerated sub-field indicating the register tier used with the current Price Tier. Valid values for the Number of Price Tiers sub-field are from 0 to 6 reflecting no tiers in use (0) to six tiers available (6). All other values for the Price Tier are reserved.

The Register Tier values correlates which *CurrentTierNSummationDelivered* attribute, found in sub-clause D.3.2.2.2, is accumulating usage information. Both attributes can be used to calculate and display usage and subsequent costs. Register Tier enumerated values are listed in Table D.32.

Table D.32 Register Tier Sub-field Enumerations

Enumerated Value	**Register Tier**
0x0	No Tier Related
0x1	Usage accumulating in *CurrentTier1SummationDelivered* attribute
0x2	Usage accumulating in *CurrentTier2SummationDelivered* attribute
0x3	Usage accumulating in *CurrentTier3SummationDelivered* attribute
0x4	Usage accumulating in *CurrentTier4SummationDelivered* attribute
0x5	Usage accumulating in *CurrentTier5SummationDelivered* attribute
0x6	Usage accumulating in *CurrentTier6SummationDelivered* attribute
0x7 - 0x0F	Reserved

Start Time (mandatory): A UTCTime field to denote the time at which the price signal becomes valid. A start Start time Time of 0xffffffff 0x00000000 is a special time denoting “now.”

Duration In Minutes (mandatory): An unsigned 16 bit field used to denote the amount of time in minutes after the Start Time during which the price signal is valid. Maximum value means “until changed”.

Price (mandatory): An unsigned 32 bit field containing the price of the commodity measured in base unit of Currency per Unit of Measure with the decimal point located as indicated by the Price Trailing Digit field when the commodity is delivered to the premise.

Price Ratio (optional): An unsigned 8 bit field that gives the ratio of the price denoted in the Price field to the “normal” price chosen by the commodity provider. This field is thought to be useful in situations where client devices may simply be interested in pricing levels or ratios. The value in this field should be scaled by a factor of 0.1, giving a range of ratios from 0.1 to 25.5. A value of 0xFF indicates the field is not used.

Generation Price (optional): An unsigned 32 bit field containing the price of the commodity measured in base unit of Currency per Unit of Measure with the decimal point located as indicated by the Price Trailing Digit field when the commodity is received from the premise. An example use of this field is in energy markets where the price of electricity from the grid is different than the price of electricity placed on the grid. A value of 0xFFFFFFFF indicates the field is not used.

Generation Price Ratio (optional): An unsigned 8 bit field that gives the ratio of the price denoted in the Generation Price field to the “normal” price chosen by the commodity provider. This field is thought to be useful in situations where client devices may simply be interested in pricing levels or ratios. The value in this field should be scaled by a factor of 0.1, giving a range of ratios from 0.1 to 25.5 A value of 0xFF indicates the field is not used.

Alternate Cost Delivered (optional): An unsigned 32 Integer field that provides a mechanism to describe an alternative measure of the cost of the energy consumed. An example of an Alternate Cost might be the emissions of CO_2 for each kWh of electricity consumed providing a measure of the environmental cost. Another example is the emissions of CO_2 for each cubic meter of gas consumed (for gas metering). A different value for each price tier may be provided which can be used to reflect the different mix of generation that is associated with different TOU rates.

Alternate Cost Unit (optional): An 8 bit enumeration identifying the unit (as specified in table D.3x) for the Alternate Cost Delivered field.

Table D.33 Alternate Cost Unit Enumerations

Values	Description
0x00	Reserved for future use
0x01	Kg of CO_2 per unit of measure
0x02 to 0xFF	Reserved for future use

Alternate Cost Trailing Digit (optional): An 8 bit BitMap field used to determine where the decimal point is located in the alternate cost field. The most significant nibble indicates the number of digits to the right of the decimal point. The least significant nibble is reserved.[22]

D.4.2.4.1.2 Effect on Receipt

On receipt of this command, the device is informed of a price event for the specific provider, commodity, and currency indicated.

Should the device choose to change behavior based on the price event, the change of behavior should occur after a random delay between 0 and 5 minutes, to avoid potential spikes that could occur as a result of coordinated behavior changes. Likewise, should a device choose to change behavior based on the expiration of the price event, the change in behavior should occur after a random delay between 0 and 5 minutes. (CCB CC-900 [SE])

D.4.3 Client

D.4.3.1 Dependencies

Events carried using this cluster include a timestamp with the assumption that target devices maintain a real time clock. Devices can acquire and synchronize their internal clocks via the ZCL Time server.

If a device does not support a real time clock it is assumed that the device will interpret and utilize the “Start Now” 0x00000000 value within the Time field.

Anonymous Inter-PAN transmission mechanism outlined in Annex B.

D.4.3.2 Attributes

None.

D.4.3.3 Commands Received

The client receives the cluster specific response commands detailed in sub-clause D.4.2.3.1.

D.4.3.4 Commands Generated

The client generates the cluster specific commands detailed in sub-clause D.4.2.4.1, as required by the application.

22. CCB 973

D.5 Messaging Cluster

D.5.1 Overview

This cluster provides an interface for passing text messages between ZigBee devices. Messages are expected to be delivered via the ESP and then unicast to all individually registered devices implementing the Messaging Cluster on the ZigBee network, or just made available to all devices for later pickup. Nested and overlapping messages are not allowed. The current active message will be replaced if a new message is received by the ESP.

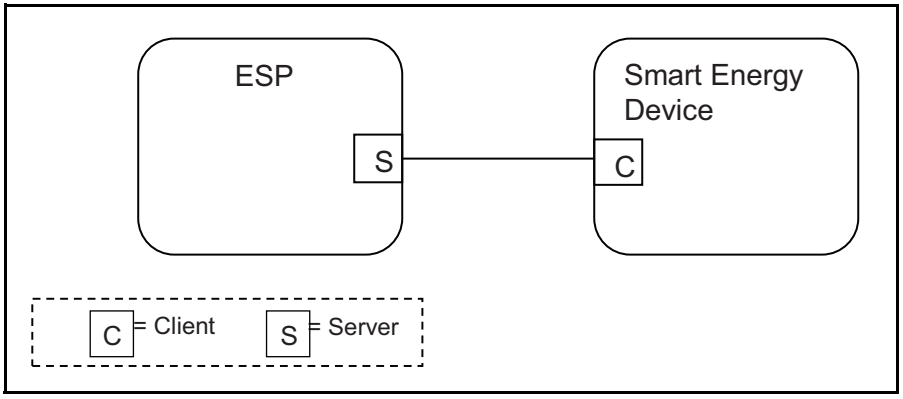

Figure D.20 Messaging Cluster Client/Server Example

Please note the ESP is defined as the Server due to its role in acting as the proxy for upstream message management systems and subsequent data stores.

D.5.2 Server

D.5.2.1 Dependencies

Support for ZCL Data Types.

Anonymous Inter-PAN transmission mechanism outlined in Annex B.

No dependencies exist for other Smart Energy Clusters.

D.5.2.2 Attributes

None.

D.5.2.3 Commands Generated

The command IDs generated by the Messaging server cluster are listed in Table D.34.

Table D.34 Generated Command IDs for the Messaging Server

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Display Message	M
0x01	Cancel Message	M
0x02 – 0xff	Reserved	

D.5.2.3.1 Display Message

D.5.2.3.1.1 Payload Format

The Display Message command payload shall be formatted as illustrated in Table D.35.

Table D.35 Display Message Command Payload

Octets	4	1	4	2	Variable
Data Type	Unsigned 32-bit integer	8-bit BitMap	UTCTime	Unsigned 16 bit Integer	Character string
Field Name	Message ID	Message Control	Start Time	Duration In Minutes	Message

D.5.2.3.1.1.1 Payload Details

Message ID: A unique unsigned 32 bit number identifier for this message. It's expected the value contained in this field is a unique number managed by upstream systems or a UTC based time stamp (UTCTime data type) identifying when the message was issued.

MessageControl: An 8-bit BitMap field indicating the need to optionally pass the message onto the Anonymous Inter-PAN transmission mechanism outlined in Annex B or that a user confirmation is required for a message. Bit encoding of this field is outlined in Table D.36:

Table D.36 Message Control Field Bit Map

Bits	Enumeration	Value	Description
Bits 0 to 1	Normal transmission only	0	Send message through normal command function to client.
	Normal and Anonymous Inter-PAN transmission	1	Send message through normal command function to client and pass message onto the Anonymous Inter-PAN transmission mechanism.
	Anonymous Inter-PAN transmission only	2	Send message through the Anonymous Inter-PAN transmission mechanism.
	Reserved	3	Reserved value for future use.
Bits 2 to 3	Low	0	Message to be transferred with a low level of importance.
	Medium	1	Message to be transferred with a medium level of importance.
	High	2	Message to be transferred with a high level of importance.
	Critical	3	Message to be transferred with a critical level of importance.
Bits 4 to 6	Reserved	N/A	These bits are reserved for future use.
Bit 7	Message Confirmation	0	Message Confirmation not required.
		1	Message Confirmation required.

If the Anonymous Inter-PAN transmission mechanism outlined in Annex B is not supported on a particular device, Bits 0 to 6 can be ignored.

The Message Confirmation bit indicates the message originator requests a confirmation of receipt from a Utility Customer. If confirmation is required, the device should display the message or alert the user until it is either confirmed via a button or by selecting a confirmation option on the device. Confirmation is typically used when the Utility is sending down information such as a disconnection notice, or prepaid billing information. Message duration is ignored when confirmation is requested and the message is displayed until confirmed.

Note: It is desired that the device provide a visual indicator (flashing display or indicate with its LEDs as examples) that a message requiring confirmation is being displayed, and requires confirmation.

Start Time: A UTCTime field to denote the time at which the message becomes valid. A Start Time of 0x00000000 is a special time denoting “now.”

Duration In Minutes: An unsigned 16 bit field is used to denote the amount of time in minutes after the Start Time during which the message is displayed. A Maximum value of 0xFFFF means “until changed”.

Message: A ZCL String containing the message to be delivered. The String shall be encoded in the UTF-8 format. Please note: Since the Anonymous Inter-PAN transmission mechanism outlined in Annex B does not support fragmentation and is limited in its message size, any message forwarded will be truncated to match the maximum message length supported. For messages not sent through the Anonymous Inter-PAN transmission mechanism and received by devices that display messages smaller than 80 bytes, they shall have the ability to receive up to an 80 byte message. Devices will have the ability to choose the methods for managing messages that are larger than can be displayed (truncation, scrolling, etc.).

For supporting larger messages sent over the SE Profile network, both devices must agree upon a common Fragmentation ASDU size. Please refer to sub-clause 5.3.8 for further details on Fragmentation settings.

Any message that needs truncation shall truncate on a UTF-8 character boundary.

D.5.2.3.2 Cancel Message

The Cancel Message command described in Table D.37 provides the ability to cancel the sending or acceptance of previously sent messages. When this message is received the recipient device has the option of clearing any display or user interfaces it supports, or has the option of logging the message for future reference.

Table D.37 Cancel Message Command Payload

Octets	**4**	**1**
Data Type	Unsigned 32-bit integer	8-bit BitMap
Field Name	Message ID	Message Control

D.5.2.3.2.1 Payload Details

Message ID: A unique unsigned 32 bit number identifier for the message being cancelled. It’s expected the value contained in this field is a unique number

managed by upstream systems or a UTC based time stamp (UTCTime data type) identifying when the message was originally issued.

MessageControl: An enumerated field indicating the optional ability to pass the cancel message request onto the Anonymous Inter-PAN transmission mechanism outlined in Annex B. If the Anonymous Inter-PAN transmission mechanism is not supported on a particular device, this parameter is ignored. Bitmap values for this field are listed in Table D.36.

D.5.3 Client

D.5.3.1 Dependencies

Support for ZCL Data Types.

No dependencies exist for other Smart Energy Clusters.

D.5.3.2 Attributes

None.

D.5.3.3 Commands Generated

The command IDs generated by the Messaging cluster are listed in Table D.38.

Table D.38 Messaging Client Commands

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Get Last Message	M
0x01	Message Confirmation	M
0x02 – 0xff	Reserved	

D.5.3.3.1 Get Last Message

This command has no payload.

D.5.3.3.1.1 Effect on Receipt

On receipt of this command, the device shall send a Display Message command (refer to sub-clause D.5.2.3.1).

D.5.3.3.2 Message Confirmation

The Message Confirmation command described in Table D.39 provides the ability to acknowledge a previously sent message.

Table D.39 Message Confirmation Command Payload

Octets	**4**	**4**
Data Type	Unsigned 32-bit integer	UTCTime
Field Name	Message ID	Confirmation Time

D.5.3.3.2.1 Payload Details

Message ID: A unique unsigned 32 bit number identifier for the message being cancelled.

Confirmation Time: UTCTime of user confirmation of message.

D.5.4 Application Guidelines

For Server and Client transactions, please refer to Figure D.21.

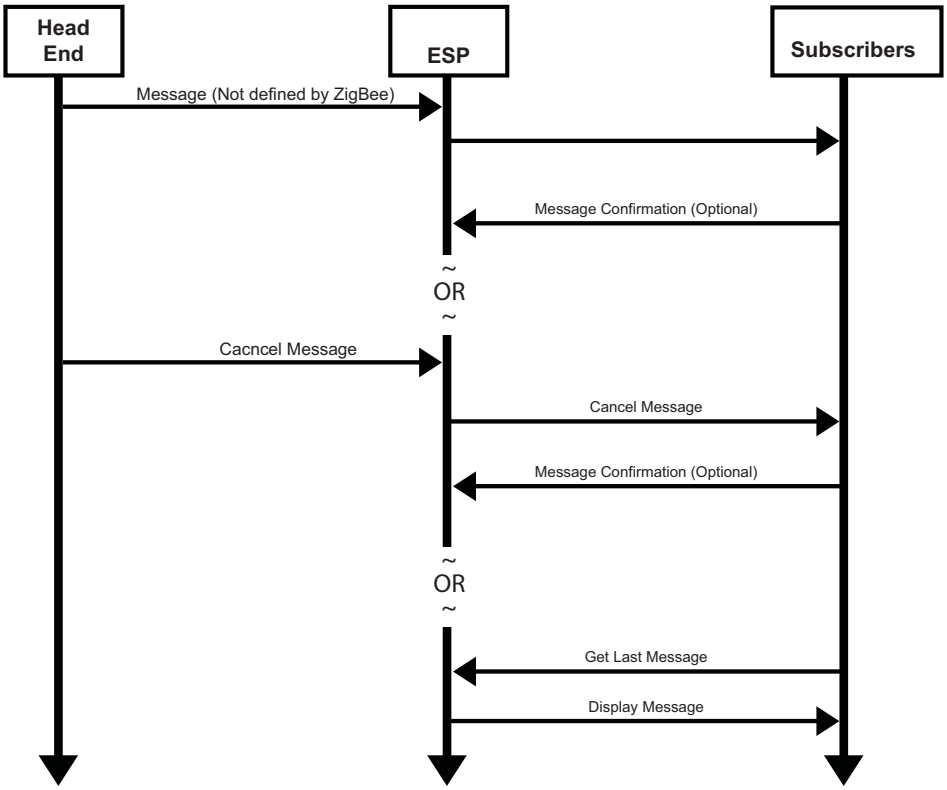

Figure D.21 Client/Server Message Command Exchanges

D.6 Smart Energy Tunneling (Complex Metering) Cluster

TBD in a future revision of the Smart Energy Profile specification.

D.7 Pre-Payment Cluster

TBD in a future revision of the Smart Energy Profile specification.

ANNEX

E

RULES AND GUIDELINES FOR OVERLAPPING EVENTS

This section describes multiple scenarios that Demand Response and Load Control devices may encounter over the Smart Energy network. The examples describe situations of overlapping events that are acceptable and where overlapping events that will be superseded due to conflicts.

E.1 Definitions

Start Time – "Start Time" field contained within the Load Control Event packet indicating when the event should start. Please note, a "Start Time" value of 0x00000000 denotes "now" and the device should use its current time as the "Start Time".

Duration – "Duration" field contained within the Load Control Event packet indicating how long the event should occur.

End Time – Time when Event completes as calculated by adding *Duration* to *Start Time*.

Scheduled Period - Represents the time between the *Start Time* and the *End Time* of the event.

Effective Start Time - Represents time at which a specific device starts a load control event based on the *Start Time* plus or minus any randomization offsets.

Effective End Time - Represents time at which a specific device ends a load control event based on the *Start Time* plus *Duration*, plus or minus any randomization offsets.

Effective Scheduled Period - Represents the time between the *Effective Start Time* and the *Effective End Time*.

Overlapping Event - Defined as an event where the *Scheduled Period* covers part or all of an existing, previously scheduled event.

Successive Events - Defined as two events where the scheduled *End Time* of the first event is equal the *Start Time* of a subsequent scheduled event.

Nested Events - Defined as two events where the scheduled *Start Time* and *End Time* of the second event falls during the *Scheduled Period* of the first scheduled event and the second event is of shorter duration than the first event.

E.2 Rules and Guideline

The depicted behaviors and required application management decisions are driven from the following guidance and rule set:

1. Upstream Demand Response/Load Control systems and/or the ESP shall prevent mismanaged scheduling of *Overlapping Events* or *Nested Events*. It is recognized Upstream Demand Response/Load Control systems and/or the ESP will need to react to changing conditions on the grid by sending *Overlapping Events* or *Nested Events* to supersede previous directives. But those systems must have the proper auditing and management rules to prevent a cascading set of error conditions propagated by improperly scheduled events.
2. When needed, Upstream Demand Response/Load Control systems and/or the ESP may resolve any event scheduling conflicts by performing one of the following processes:
   a. Canceling individual events starting with the earliest scheduled event and re-issuing a new set of events.
   b. Canceling all scheduled events and re-issuing a new set of events.
   c. Sending *Overlapping Events* or *Nested Events* to supersede previous directives.

   It is recommended that process 2.c is used for most situations since it can allow a smoother change between two sets of directives, but no way does it negate the responsibilities identified in rule #1.
3. When an End Device receives an event with the *End Time* in the past (*End Time* < Current Time), this event is ignored and a Report Event Status command is returned with the Event Status set to 0xFB (Rejected - Event was received after it had expired).
4. When an End Device receives an event with a *Start Time* in the past and an *End Time* in the future ((*Start Time* < Current Time) AND (*End Time* > Current Time)), the event is processed immediately. The Effective *Start Time* is calculated using the Current Time as the *Start Time*. Original *End Time* is preserved.

5 Regardless of the state of an event (scheduled or executing), when an *End Device* detects an *Overlapping Event* condition the latest *Overlapping Event* will take precedence over the previous event. Depending on the state of the event (scheduled or executing), one of the following steps shall take place:

   a If the previous event is scheduled and not executing, the End Device returns a Report Event Status command (referencing the previous event) with the Event Status set to 0x07 (The event has been superseded). After the Report Event Status command is successfully sent, the End Device can remove the previous event schedule.

   b If the previous event is executing, the End Device shall change directly from its current state to the requested state at the *Effective Start Time* of the *Overlapping Event* (Note: Rule #4 effects *Effective Start Time*). The End Device returns a Report Event Status command (referencing the previous event) with the Event Status set to 0x07 (the event has been superseded).

6 Randomization ***shall not*** cause event conflicts or unmanaged gaps. To clarify:

   a When event starting randomization is requested, time periods between the *Start Time* of an event and the *Effective Start Time* a device should either maintain its current state or apply changes which contribute to energy saving. Preference would be to maintain current state.

   b When event ending randomization is used and the *Effective End Time* overlaps the *Effective Start Time* of a *Successive Event*, the *Effective Start Time* takes precedence. Events are not reported as superseded, End devices should report event status as it would a normal set of *Successive Events*.

   c It is recommended devices apply the same Start and Stop Randomization values for consecutive events to help prevent unexpected gaps between events.

   d Devices ***shall not*** artificially create a gap between *Successive Events*.

7 It is permissible to have gaps when events are not *Successive Events* or *Overlapping Events*.

8 If multiple device classes are identified for an event, future events for individual device classes (or a subset of the original event) that cause an *Overlapping Event* will supersede the original event strictly for that device class (or a subset of the original event). Note: Rule #5 applies to all *Overlapping Events*.

E.3 Event Examples

Smart Energy devices which act upon Demand Response and Load Control events shall use the following examples for understanding and managing overlapping and superseded events. Within those examples, references to multiple device

classes will be used. Figure E.1 depicts a representation of those devices in a Smart Energy network.

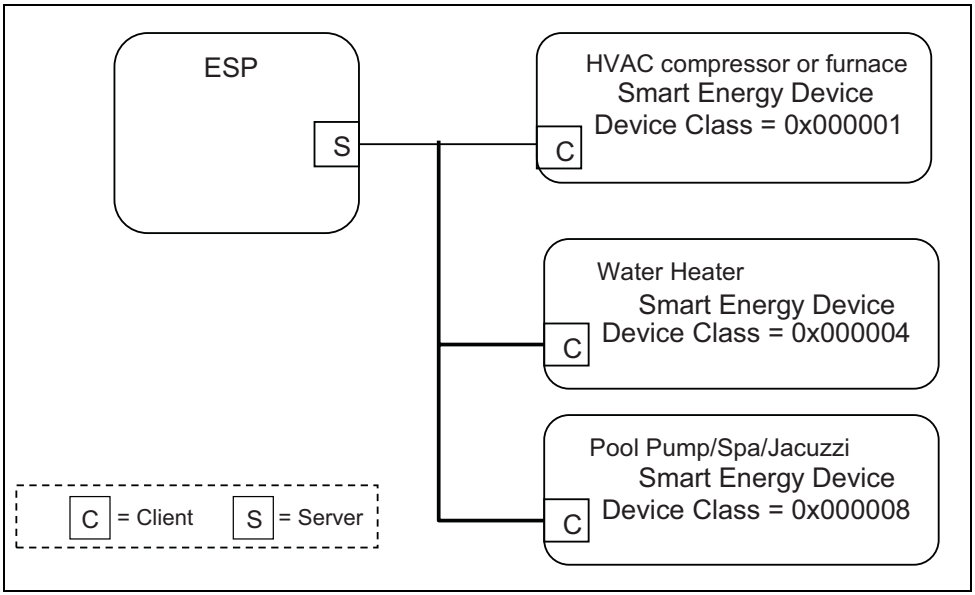

Figure E.1 Smart Energy Device Class Reference Example

E.3.1 Correct Overlapping Events for Different Device Classes

Figure E.2 depicts a correct series of DR/LC event for device class of 0x000001 (reference for the BitMap definition) with an event scheduled for another device class during the same period.

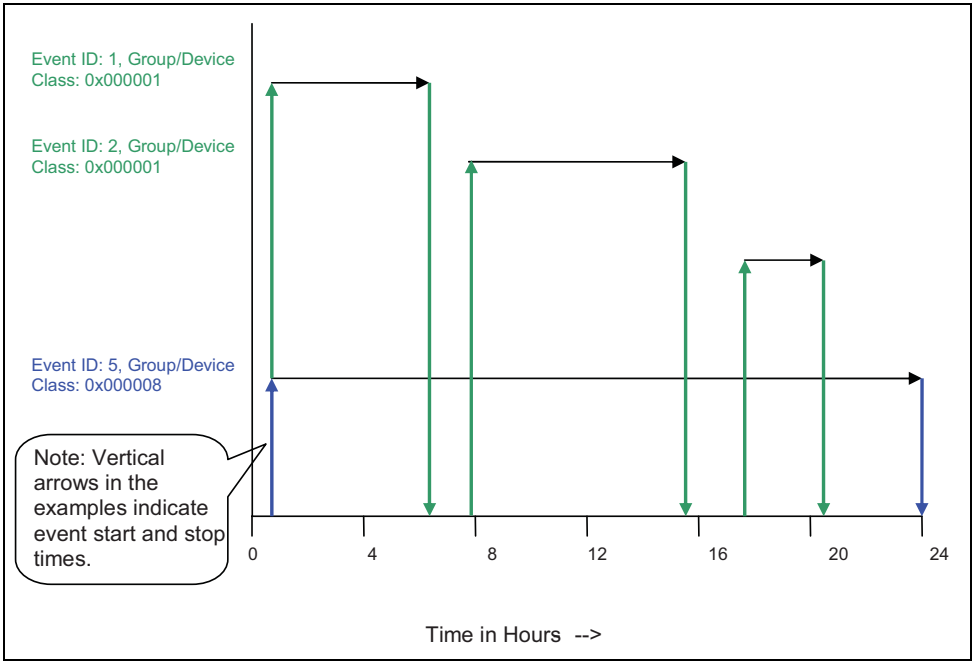

Figure E.2 Correctly Overlapping Events

In Figure E.2, Device Class 0x000001 receives a sequence of 3 unique DR/LC events to be scheduled and acted upon. During this same 24 hour period, Device Class 0x000008 receives one scheduled DR/LC event that spans across the same time period as the events scheduled for Device Class 0x0000001. Because both Device Classes are unique, there are no conflicts due to Overlapping Events.

E.3.2 Correct Superseded Event for a Device Class

Figure E.3 below depicts a correct series of DR/LC events for device class of 0x000001 (reference for the BitMap definition) where an event is scheduled then later superseded.

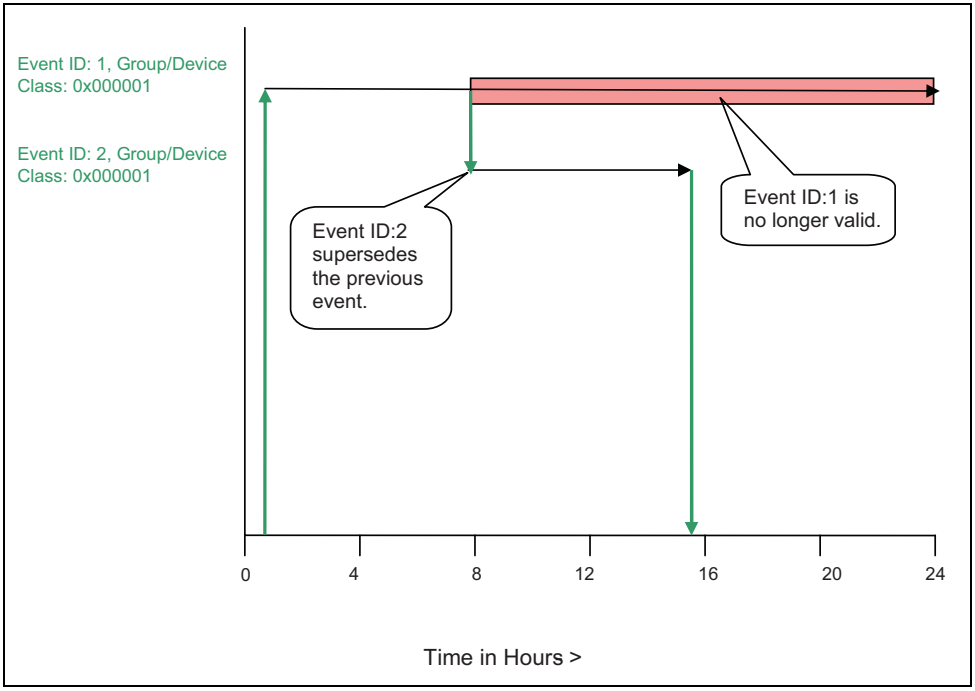

Figure E.3 Correct Superseding of Events

In Figure E.3, Device Class 0x000001 receives DR/LC Event ID#1 setup for a 24 hour *Scheduled Period*, which later is superseded by DR/LC Event ID#2, invalidating the remainder of Event ID#1, which is cancelled.

E.3.3 Superseding Events for Subsets of Device Classes

Figure E.4 below depicts a correct series of DR/LC events for device class of 0x000001 (reference for the BitMap definition) with an event scheduled for another device class during the same time period.

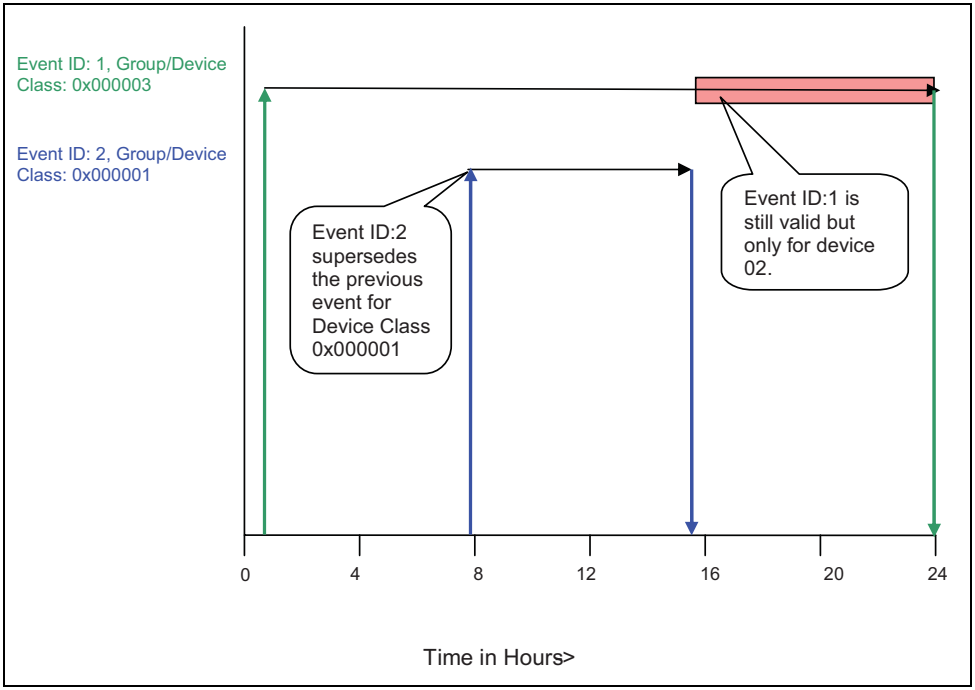

Figure E.4 Superseded Event for a Subset of Device Classes

In Figure E.4, Device Class 0x000003 receives DR/LC Event ID#1 setup for a 24 hour *Scheduled Period*, which is targeted for both Device Class 0x000002 and 0x000001 (OR'ed == 0x000003). In the example, Event ID#2 is issued only for Device Class 0x000001, invalidating the remainder of Event ID#1 for that device class. DR/LC Event ID#1 is still valid for Device Class 0x000002, which in the example should run to completion.

E.3.4 Ending Randomization Between Events

Figure E.5 below depicts an *Effective End Time* that overlaps a second scheduled DR/LC event for device class of 0x000001 (reference for the BitMap definition).

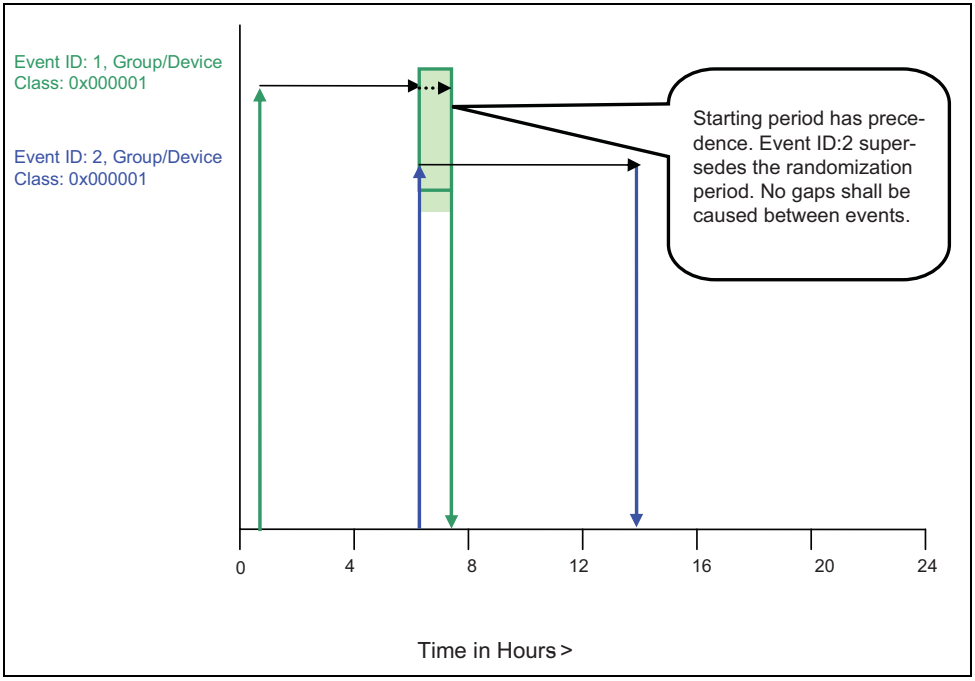

Figure E.5 Ending Randomization Between Events

In Figure E.5, Device Class 0x000001 receives a DR/LC Event ID#1 with an ending randomization setting (please refer to sub-clause D.2.2.3.1.1.1 for more detail). A second DR/LC (Event ID#2) is issued with a starting time which matches the ending time of DR/LC Event ID#1. In this situation, the *Start Time* of Event ID#2 has precedence. Event ID#1 is not reported as superseded.

E.3.5 Start Randomization Between Events

Figure E.6 below depicts an *Effective Start Time* that overlaps a previously scheduled DR/LC event for device class of 0x000001 (reference for the BitMap definition).

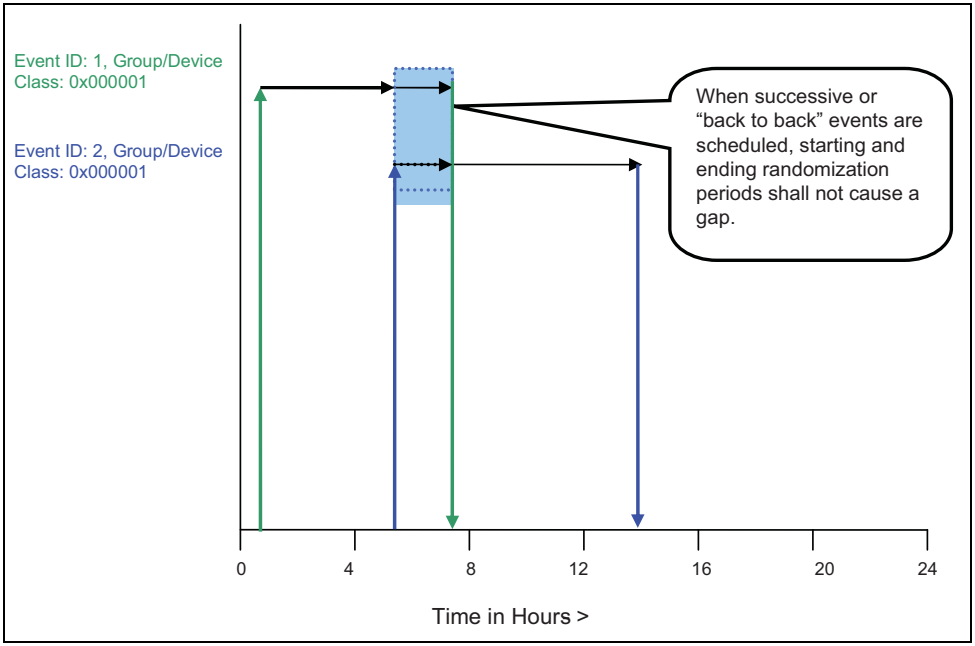

Figure E.6 Start Randomization Between Events

Figure E.6 above, Device Class 0x000001 receives a DR/LC Event ID#1 with an ending randomization setting (please refer to sub-clause D.2.2.3.1.1.1 for more detail). *Effective End Time* of Event ID#1 is not known. A second DR/LC (Event ID#2) is issued with a starting randomized setting, which has an *Effective Start Time* that could overlap or start after the *Effective End Time* of DR/LC Event ID#1. In this situation, the *Effective Start Time* of Event ID#2 has precedence but the DR/LC device must also prevent any artificial gaps caused by the *Effective Start Time* of Event ID#2 and *Effective End Time* of Event ID#1.

E.3.6 Acceptable Gaps Caused by Start and Stop Randomization of Events

Figure E.7 below depicts an acceptable gap between two scheduled DR/LC events for device class of 0x000001 (reference for the BitMap definition) using both starting and ending randomization with both events.

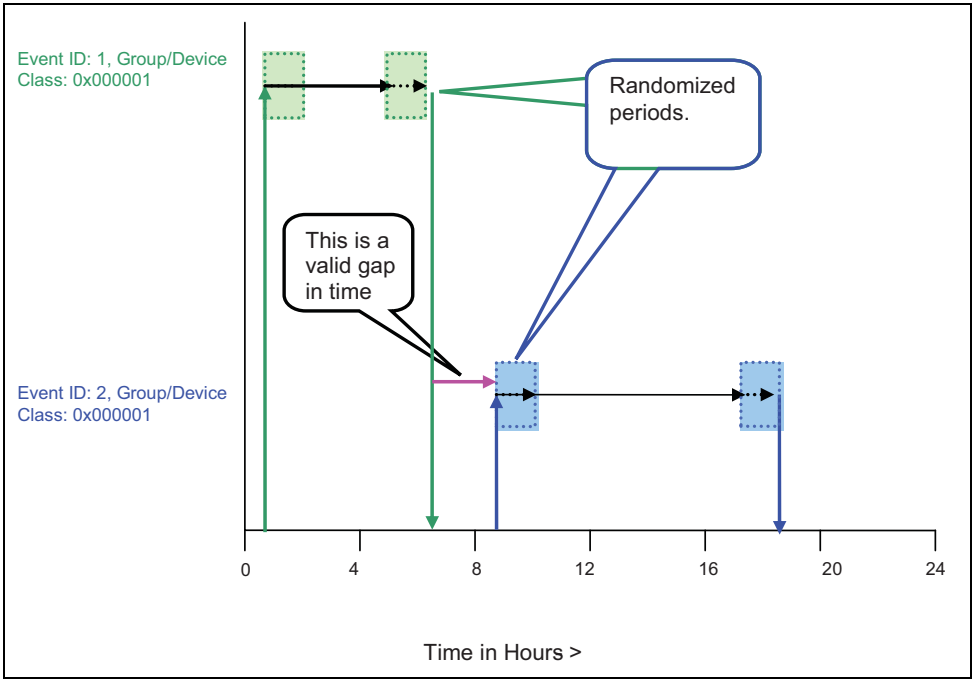

Figure E.7 Acceptable Gaps with Start and Stop Randomization

Figure E.7 above, Device Class 0x000001 receives a DR/LC Event ID#1 with both a starting and ending randomization setting (please refer to sub-clause D.2.2.3.1.1.1 for more detail). A second DR/LC Event ID#2 is also issued with both a starting and ending randomized setting. The primary configuration to note in this example is the *Effective End Time* of DR/LC Event ID#1 completes well in advance of the *Effective Start Time* of DR/LC Event ID#2. In this scenario, regardless of randomization a gap is naturally created by the scheduling of the events and is acceptable.

ANNEX

F

JOINING PROCEDURE USING PRE-CONFIGURED TRUST CENTER LINK KEYS

The secure join procedure is detailed as follows:

- The secured joining procedure is as stated in [B4] Section 4.6.3.2.3. The case used in the Smart Energy application is the “Pre-configured trust center link key and address”
- In [B4] Section 4.6.3.2.3.2, in the case of “Pre-configured Trust Center Link Key”, the joining device waits for the APSME-TRANSPORT-KEY. Indication. The frame is encrypted/authenticated with the key-transport key according to the methodologies specified in sections 4.4.1.1 and 4.5.3 of the ZigBee specification r17, which describe the key-transport keys and their association with link keys, in this case the pre-configured trust center link key. The source address will be that of the Trust Center. The key transported will be the NWK Key Key type == 0x01.
- When the trust center sends the tunneled Transport Key command, the Extended Nonce bit on the Auxiliary Frame Header must be set to 1 on the Transport Key frame from the Trust Center to the joining child as described in [B4] Section 4.5.1. The Trust Center must also insert its long address into the Source Address field of the Auxiliary Frame Header since that information will be needed at the child to decrypt the Transport Key command.
- Sub-clause 5.4 of this document calls out two cases for secured join: pre-configured link keys and temporary link keys. The joining device and trust center perform the same join operation in both cases. The only difference is how the joining device and trust center treat the initial key material (either using it directly as the pre-configured link key or hashing with some data like the long address of the joining device at application level first, see Annex E for this method). From the perspective of the security joining process what happens afterwards is the secure join procedure is the same.

- In either case called out in sub-clause 5.4 of this document, the joining device is authenticated using the [B4] Section 4.6.3.2.3.2 procedure or leaves if the security timeout expires. If authenticated, the key delivered via the APSME-TRANSPORT-KEY.indication in [B4] Section 4.6.3.2.3.2 is the same for either case called out in the AMI specification sub-clause 5.4 (no matter how the application determined the pre-configured link key).

In terms of the message exchange between the child and trust center in performing the secure join procedure, the following is employed:

- Child joining device uses NLME-JOIN.request to parent. Parent sends an APSME-UPDATE-DEVICE.request to the Trust Center on behalf of the child to the Trust Center. APSME-UPDATE-DEVICE.request is transported encrypted/authenticated with the NWK key that the parent has
- Upon receipt at the trust center, the trust center must perform the following processing:
  - Validity check of the child's address to determines if a trust center link key exists between the trust center and the address provided by the joining child.
  - If the child has the trust center as its parent, the APSME-TRANSPORT-KEY.request is sent directly to the child encrypted with the key-transport key derived from the trust center link key known to the child device and the trust center, ELSE
    - —If the child does not have the trust center as its parent, the APSME-TRANSPORT-KEY command frame is encrypted using the key-transport key derived from the trust center link key shared between the child and the trust center.
  - The resulting encrypted payload is sent to the child using the APS Tunnel command. The APS Tunnel command and its (already encrypted) payload is encrypted using the NWK key from the trust center to the child's parent. On the final hop, the child's parent will perform the following processing according to [B4] Section 4.6.3.7.2:
    - — The parent sends the contents within the APS Tunnel command to the child without network layer encryption. The message from the parent to the joining child is an APS encrypted transport key command using the key-transport key derived from the trust center link key.

Here are the details on the message that is routed from the trust center to the joining device's parent via the tunnel command:

- NWK Data Frame (Dest: Parent)
- APS Header (Command)
- APS Command Frame (Tunnel)

- Dest EUI: Child
- Tunnel Payload
- APS Header
- APS Auxiliary Header
- Encrypted Payload
- APS Command Frame (Transport Key)

Here are the details on the message that is routed from the joining device's parent to the joining child:

- NWK Data Frame (added by parent, Dest: child)
- APS Header (from Tunnel Payload)
- APS Auxiliary Header (from Tunnel Payload)
- Encrypted Payload (from Tunnel Payload)
- APS Command Frame (Transport Key)

The message to the child from the parent is identical if the device joins directly to the Trust Center.

As a note on the final hop contents of the payload:

- The last hop of the APME-TRANSPORT-KEY message from parent to joining child has NO network layer encryption, but does have application layer encryption
- Thus: There will be no NWK auxiliary header, but there will be an APS auxiliary header
- The APS auxiliary header will have the Key Identifier Sub-Field set to 0x02 == A key-transport key (see [B4] Section 4.5.1.1.2)
- The APS frame will be encrypted with the key-transport key derived from the pre-configured trust center link key. The pre-configured trust center link key must be part of the apsDeviceKeyPairSet in the AIB of the joining device and also known to the trust center.
- The resulting APS frame from the parent to the joining child is the APS-TRANSPORT-KEY message encrypted with the key-transport key derived from the trust center link key delivered with the key type of key-transport key (0x02).
- Per [B4] Section 4.4.3.2, the KeyType field will be set to (0x01) == Network Key
- The TransportKeyData will be the active network key and sequence number

- The joining device must set the network key and sequence number in its NWK Information Block.
- The device is then joined and authenticated.