UG103.2: Application Development Fundamentals: ZigBee

This document describes the key features and characteristics of a ZigBee solution. It also includes a section on ZigBee 3.0.

Silicon Labs' *Application Development Fundamentals* series covers topics that project managers, application designers, and developers should understand before beginning to work on an embedded networking solution using Silicon Labs chips, networking stacks such as EmberZNet PRO or Silicon Labs Bluetooth Smart, and associated development tools. The documents can be used a starting place for anyone needing an introduction to developing wireless networking applications, or who is new to the Silicon Labs development environment.

KEY FEATURES

- Introduces ZigBee and key features—mesh networking, network node types, routing concepts, ZigBee stack, ZigBee Cluster Library, and ZigBee compliances
- Includes a section on ZigBee 3.0.

1   Introduction

ZigBee refers both to:

- An open standard for reliable, cost-effective, low power, wireless device-to-device communication
- An alliance of over 400 companies who together are defining and using the standard to communicate in a variety of applications such as smart energy and commercial building automation.

The ZigBee Alliance is operated by a set of promoter companies that make up the Board of Directors. These currently include Silicon Labs, Philips, Schneider Electric, Landis and Gyr, Itron, Legrand, Kroger, Comcast, Texas Instruments, and NXP. The Alliance activities are accomplished through workgroups dedicated to specific areas of the technology. These include a network group, a security group, an application profile group, and several others. Standards are available to members and then non-members for download. To use the ZigBee technology within a product, companies are required to become members of the Alliance.

Silicon Labs is active in a variety of areas within the Alliance and with helping customers adopt ZigBee technology through its Ember® ZigBee solutions. The Ember ZigBee platform has been a Golden Unit for testing and certification for all new revisions of the standard to date. Ember Corporation, which was acquired by Silicon Labs in July 2012, was a founding member of the ZigBee Alliance.

The ZigBee Alliance specifies three major items of interest.

1. **ZigBee networking** – The basic levels of network interaction, including acceptable RF behavior, methods of creating and joining the network, discovering routes, and using routes to transmit traffic over the network.

2. **ZigBee application layer** – Describe a set of messages and network settings for a ZigBee application. All devices adhering to these settings may interoperate. Before ZigBee 3.0, there were individual application profiles, either public or private, that had their own certification program. With ZigBee 3.0, there is one common application layer certification program, which applies to every ZigBee product. Figure 1 displays some of the original separate ZigBee application profiles.
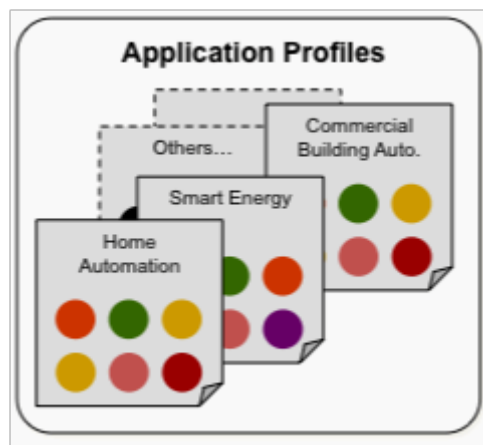

Figure 1. ZigBee Application Profiles

3. **Certification** – ZigBee applications are certified by the Alliance to indicate to end users that the device(s) are compliant. Applications using private profiles receive certification that they are using a properly operating ZigBee stack and will not adversely affect other ZigBee networks during operation. For more information, see Section 7, ZigBee Compliance.

Because ZigBee is committed to open and interoperable devices, standards have been either adopted where they existed, or developed, from the physical layer through the application layer, as shown in Figure 2. At the physical and MAC (medium access control) layer, ZigBee adopted the IEEE 802.15.4 standard. The networking, security and application layers have all been developed by the ZigBee Alliance. An ecosystem of supporting systems such as gateways and commissioning tools has also been developed to simplify the development and deployment of ZigBee networks. Extensions and additions to the standards continue to be developed, and Silicon Labs is committed to supporting these as they are available.
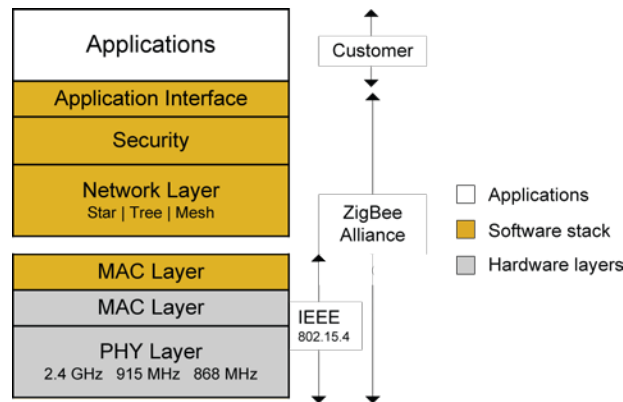
Figure 2. ZigBee Architecture

Silicon Labs provides a standard networking API based on the ZigBee specification across theEM3xx family of products and the Wireless Gecko (EFR32™) portfolio. Silicon Labs also provides tools which allow rapid ZigBee application development, including an application framework and a GUI-based tool called AppBuilder. Silicon Labs uses Simplicity Studio as its Integrated Development Environment (IDE) and leverages the IAR Embedded Workbench for ARM platforms.

The following three sections describe the general characteristics of a ZigBee network, discuss the use of IEEE 802.15.4 standard, and summarize the hardware and software elements of a ZigBee network.

1.1 General Characteristics

ZigBee is intended as a cost-effective and low power solution. It is targeted to a number of markets including home automation, building automation, sensor networks, smart energy, and personal health care monitoring. The general characteristics for a ZigBee network are as follows:

- Low power – Devices can typically operate for several years on AA type batteries using suitable duty cycles. With extremely careful design and special battery technologies, some ZigBee devices such as gas meters can achieve 20 years of battery life.
- Low data rate – The 2.4 GHz band supports a radio data rate of 250 kbps. Actual sustainable traffic through the network is lower than this theoretical radio capacity. As such, ZigBee is better used for sampling and monitoring applications or basic control applications.
- Small and large networks – ZigBee networks vary from several devices to thousands of devices communicating seamlessly. The networking layer is designed with several different data transfer mechanisms (types of routing) to optimize the network operation based on the expected use.
- Range – Typical devices provide sufficient range to cover a normal home. Readily available designs with power amplifiers extend the range substantially. A distributed spread spectrum is used at the physical layer to be more immune to interference.
- Simple network installation, start up and operation – The ZigBee standard supports several network topologies. The simple protocols for forming and joining networks allow systems to self-configure and fix routing problems as they occur.

1.2 IEEE 802.15.4

ZigBee networks are based on the IEEE 802.15.4 MAC and physical layer. The 802.15.4 standard operates at 250 kbps in the 2.4 GHz band and 40 kbps/20 kbps in the 900/868 MHz bands. A number of chip companies provide solutions in the 2.4 GHz band with a smaller number supporting the 900/868 MHz band. ZigBee PRO uses 802.15.4-2003.

The 802.15.4 MAC layer is used for basic message handling and congestion control. This MAC layer includes mechanisms for forming and joining a network, a CSMA mechanism for devices to listen for a clear channel, as well as a link layer to handle retries and acknowledgement of messages for reliable communications between adjacent devices. The ZigBee network layer builds on these underlying mechanisms to provide reliable end to end communications in the network. The 802.15.4 standard is available from www.ieee.org.

The 802.15.4 standard provides some options within the MAC layer that are not used by ZigBee in any current stack profiles. An example of such an unused feature is guaranteed time slots (GTS), which would be employed by a network to synchronize radio activity across devices. As such, these items are not normally included in the ZigBee software stack to save code space. ZigBee also has made specific changes to the 802.15.4 MAC that are documented in Annex D of the ZigBee specification.

1.3 Hardware and Software Elements

A ZigBee solution requires implementation of a ZigBee radio and associated microprocessor (together in a single chip or separately), and implementation of an application on top of a ZigBee stack. EmberZNet PRO is the Silicon Labs implementation of the ZigBee PRO stack.

Typically a developer can purchase a ZigBee radio and software as a bundled package, although some third party software stacks have been developed. In general, the hardware and software provider includes reference designs for the hardware and sample applications for the software. Based on these, hardware developers can customize the hardware to their specific needs. Alternatively, a number of module providers can deliver compact and low cost custom modules.

Because of the embedded nature of typical ZigBee applications, software application development is typically interrelated with the hardware design to provide an optimal solution. Silicon Labs offers an application framework, which gives customers a way to rapidly develop their applications based on ZigBee application profiles and the ZigBee cluster library (ZCL). Alternatively, a number of third party software development firms specialize in developing ZigBee applications and can assist in new product development.

AppBuilder works with the ZigBee application framework, and includes support for the ZigBee 3.0 application layer, as well as a number of legacy ZigBee profiles (such as Home Automation (HA), Smart Energy (SE), and ZigBee Light Link (ZLL)). Silicon Labs also offers a number of utilities and tools that are available to developers during the application process including:

- Over-the-air (OTA) bootloaders to allow upgrades to system software after deployment
- Gateway interfaces to interface the ZigBee network to other systems
- Programming tools for the microprocessor
- Network sniffer and debug tools to allow viewing and analysis of network operations.
- Processor debug tools through partnerships with leading firms.

2   ZigBee Mesh Networking

Embedded mesh networks make radio systems more reliable by allowing radios to relay messages for other radios. For example, if a node cannot send a message directly to another node, the embedded mesh network relays the message through one or more intermediary nodes.

EmberZNet PRO supports three types of mesh network topologies, as shown in Figure 3:

- Star Network
- Full Mesh Network
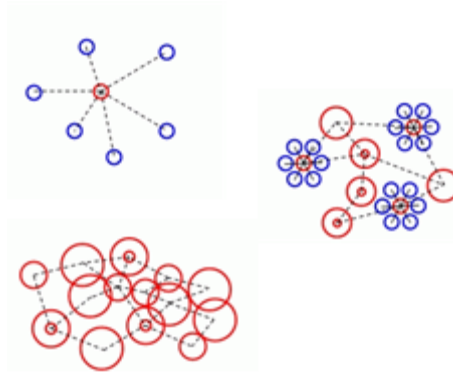- Hybrid Mesh Network

Figure 3. Star, Full Mesh, and Hybrid Mesh Networks

Note: Blue devices in Figure 4 are end devices and can be sleepy or mobile. See Section 3, Network Node Types, for more information about end devices.

2.1.1   Star Network

In a star network, one hub is the central point of all communications. The hub can become bottlenecked with network/processing bandwidth. This topology is not very mesh-like, and transmission is limited by the hub's communication radius. Outlying nodes can be battery powered. In the EmberZNet PRO stack, this topology is formed by a group of end devices with a coordinator node as their parent. The coordinator node serves as network hub.

2.1.2   Full Mesh Network

In a full mesh network, all nodes are router nodes, including the coordinator after it forms the network. Because all nodes can relay information for all other nodes, this topology is least vulnerable to link failure; it is highly unlikely that one device might act as a single point of failure for the entire network.

2.1.3   Hybrid Mesh Network

A hybrid mesh network topology combines star and mesh strategies. Several star networks exist, but their hubs can communicate as a mesh network. A hybrid network allows for longer distance communication than a star topology and more capability for hierarchical design than a mesh topology. This topology is formed by the EmberZNet PRO stack, using router devices as hubs for the star subnetworks, where each hub can have end devices attached to it.

The choice of topology must take into consideration which nodes are line-powered or battery-powered, expected battery lifetime, amount of network traffic required, latency requirements, the cost of the solution, as well as other factors. See *Fundamentals: Design Choices*, for more information.

3   Network Node Types

The ZigBee specification supports at most one coordinator, multiple routers, and multiple end devices within a single network. These node types are described in the following sections.

3.1   Coordinator

A ZigBee coordinator (ZC) is responsible for forming the network. A coordinator is a router with some additional functionality. ZigBee coordinator functions include selecting an appropriate channel after scanning available channels, and selecting an extended PAN ID (see section 5.4, Extended PAN IDs, for more information about extended PAN IDs). After forming the network, the coordinator acts as a router.

For some applications, the coordinator may have added responsibilities, such as being the trust center or network manager. The trust center manages the security settings and authorizations for the network. The network manager monitors and corrects network issues such as PAN ID conflicts or channel changing due to interference. These choices are up to the application developer, and in some cases are made by the appropriate ZigBee stack profile. For example under the Home Automation application profile the coordinator is always the trust center for the network.

Note: Only a network coordinator can be designated as a security trust center when starting a network.

3.2   Routers

ZigBee router devices (ZR) provide routing services to network devices. Routers can also serve as end devices. Unlike end devices, routers are not designed to sleep and should generally remain on as long as a network is established.

3.3   End Devices

End devices (ZED) are leaf nodes. They communicate only through their parent nodes and, unlike router devices, cannot relay messages intended for other nodes.

Depending on the network stack, end devices can be of several types:

- Sleepy end devices (EmberNodeType EMBER_SLEEPY_END_DEVICE) power down their radio when idle, and thus conserve resources. However, they must poll their parent node to receive incoming messages and acknowledgements; no data is sent to the sleepy end device until the end device requests it. Sleepy end devices are also sometimes known as rx-off-when-idle devices. This is a standard ZigBee device type.
- Non-sleepy end devices (EmberNodeType EMBER_END_DEVICE) do not route messages for other devices but they remain powered during operation. These devices are known as Rx-on-when-idle devices. This is a standard ZigBee device type.
- Mobile end devices (EmberNodeType EMBER_MOBILE_END_DEVICE) is a sleepy end device with enhanced capabilities that enable it to change its physical location and quickly switch to a new parent router. This device type is a Silicon Labs modification to the basic ZigBee sleepy end device to provide added capabilities and management of mobile devices.

The EmberZNet PRO stack supports sleepy and non-sleepy end devices. The choice of node type must also be considered carefully. For example, in a very dense network, it is not always advantageous for all line powered nodes to be routers due to possible interference issues which may occur when a child node tries to find a parent node to communicate with. It's important to try to create a balanced network where all nodes have redundant paths, but where there aren't too many routers in close proximity to create interference.

4   ZigBee Routing Concepts

4.1   Overview

ZigBee has several routing mechanisms that can be used based on the network and expected traffic patterns.[1] The application designer should choose which mechanism to use as part of the system architecture and design. In actual practice one application may use several of these routing mechanisms, because some devices may be performing one-to-one communications while other devices may be communicating to a central monitoring device. The types of routing discussed below are:

- Table Routing
- Broadcast Routing
- Multicast Routing
- Many-to-One/Source Routing

The ZigBee and ZigBee PRO stacks have different routing mechanisms.

4.1.1   Table Routing

The ZigBee PRO stack never uses tree routing for message delivery because an alternate addressing schemes is used and the tree does not exist in the network. Routes are formed when one node sends a route request to discover the path to another node. After a route is discovered between the two nodes, the source node sends its message to the first node in the route, as specified in the source node's routing table. Each intermediate node uses its own routing table to forward the message to the next node (that is hop) along the route until the message reaches its destination. If a route fails, a route error is sent back to the originator of the message who can then rediscover the route.

4.1.2   Broadcast Routing

Broadcast routing is a mechanism to send a message to all devices in a network. Network-level broadcast options exist to send to routers only or also to send to sleeping end devices. A broadcast message is repeated by all router-capable devices in the network three times to ensure delivery to all devices. While a broadcast is a reliable means of sending a message, it should be used sparingly because of the impact on network performance. Repeated broadcasts can limit any other traffic that may be occurring in the network. Broadcasts are also not a reliable means of delivery to a sleeping device because the parent device is responsible for buffering the message for the sleeping child but may drop the message before the end device wakes to receive it.

4.1.3   Multicast Routing

Multicast routing provides a one-to-many routing option. A multicast is used when one device wants to send a message to a group of devices, such as a light switch sending an on command to a bank of 10 lights. Under this mechanism, all the devices are joined into a multicast group. Only those devices that are members of the group will receive messages, although other devices will route these multicast messages. A multicast is a filtered limited broadcast. It should be used only as necessary in applications, because over-use of broadcast mechanisms can degrade network performance. A multicast message is never acknowledged.

4.1.4   Many-to-One/Source Routing

Many-to-one routing is a simple mechanism to allow an entire network to have a path to a central control or monitoring device. Under normal table routing, the central device and the devices immediately surrounding it would need routing table space to store a next hop for each device in the network, as well as an entry to the central device itself. Given the memory limited devices often used in ZigBee networks, these large tables are undesirable.

Under many-to-one routing, the central device, known as a "concentrator," sends a single route discovery that established a single route table entry in all routers to provide the next hop to the central device. This yields a result similar to that of table routing, but with a single many-to-one route request rather than many individual, one-to-one route requests from each router towards the concentrator.

All devices in the network then have a next hop path to the concentrator and only a single table entry is used. However, often the central device also needs to send messages back out into the network. This would result in a more significant increase in route table size, particularly for those nodes closest to the concentrator, since they are relay points in the concentrator's many outbound routes to the rest of the network. Instead, incoming messages to the concentrator first use a route record message to store the sequence of hops used along the route. The concentrator then stores these next hop routes in reverse order as "source routes" in a locally held table known as

---

[1] See ZigBee Specification, document 05-3474. Section 3.4 describes the frame format for command frames, and section 3.6.3 describes routing behavior.

a "source route table". Outgoing messages include this source route in the network header of the message. The message is then routed using next hops from the network header instead of from the route table. This provides for large scalable networks without increasing the memory requirements of all devices. It should be noted that the concentrator does require some additional memory if it is storing these source routes.

For detailed information on message delivery, refer to the ZigBee specification available from www.zigbee.org.

4.2   Using Link Quality to Aid in Routing

The information in this section is provided for those wishing to understand the details of the network layer's operation, which can prove useful during troubleshooting. Otherwise, link status messages are handled automatically by the stack and application writers need not be concerned with it.

Links in wireless networks often have asymmetrical link quality due to variations in the local noise floor, receiver sensitivity, and transmit power. The routing layer must use knowledge of the quality of links in both directions in order to establish working routes and to optimize the reliability and efficiency of those routes. It can also use the knowledge to establish reliable two-way routes with a single discovery.

ZigBee routers (ZR) keep track of inbound link quality in the neighbor table, typically by averaging LQI (Link Quality Indicator) measurements made by the physical layer. To handle link asymmetry, the ZigBee PRO stack profile specifies that routers obtain and store costs of outgoing links as measured by their neighbors. This is accomplished by exchanging link status information through periodic one hop broadcasts, referred to as "link status" messages. The link status algorithm is explained below, as implemented in EmberZNet PRO.

4.2.1   Description of Relevant Neighbor Table Fields

ZigBee routers store information about neighboring ZigBee devices in a neighbor table. For each router neighbor, the entry includes the following fields:

- average incoming LQI
- outgoing cost
- age

The incoming LQI field is an exponentially weighted moving average of the LQI for all incoming packets from the neighbor. The incoming cost for the neighbor is computed from this value using a lookup table.

The outgoing cost is the incoming cost reported by the neighbor in its neighbor exchange messages. An outgoing cost of 0 means the cost is unknown. An entry is called "two-way" if it has a nonzero outgoing cost, and "one-way" otherwise.

The age field measures the amount of time since the last neighbor exchange message was received. A new entry starts at age 0. The age is incremented every EM_NEIGHBOR_AGING_PERIOD, currently 16 seconds. Receiving a neighbor exchange packet resets the age to EM_MIN_NEIGHBOR_AGE, as long as the age is already at least EM_MIN_NEIGHBOR_AGE (currently defined to be 3). This makes it possible to recognize nodes that have been recently added to the table and avoid evicting them, which reduces thrashing in a dense network. If the age is greater than EM_STALE_NEIGHBOR (currently 6), the entry is considered stale and the outgoing cost is reset to 0.

4.2.2   Link Status Messages

Routers send link status messages every 16 seconds plus or minus 2 seconds of jitter. If the router has no two-way links it sends them eight times faster. The packet is sent as a one-hop broadcast with no retries. In the EmberZNet PRO stack, they are sent as ZigBee network command frames.

The payload contains a list of short IDs of all non-stale neighbors, along with their incoming and outgoing costs. The incoming cost is always a value between 1 and 7. The outgoing cost is a value between 0 and 7, with the value 0 indicating an unknown outgoing cost. For frame format details, refer to the ZigBee specification. Link status messages are also automatically decoded by the Simplicity Studio Network Analyzer for easy reference.

Upon receipt of a link status message, either a neighbor entry already exists for that neighbor, or one is added if there is space or if the neighbor selection policy decides to replace an old entry with it. If the entry does not get into the table, the packet is simply dropped. If it does get in, then the outgoing cost field is updated with the incoming cost to the receiving node as listed in the sender's neighbor exchange message. If the receiver is not listed in the message, the outgoing cost field is set to 0. The age field is set to EM_MIN_NEIGHBOR_AGE.

4.2.3   How Two-way Costs are used by the Network Layer

As mentioned above, the routing algorithm makes use of the bidirectional cost information to avoid creating broken routes, and to optimize the efficiency and robustness of established routes. For the reader familiar with the ZigBee route discovery process, this subsection gives details of how the outgoing cost is used. The mechanism is surprisingly simple, but provides all the benefits mentioned above.

Upon receipt of a route request command frame, the neighbor table is searched for an entry corresponding to the transmitting device. If no such entry is found, or if the outgoing cost field of the entry has a value of 0, the frame is discarded and route request processing is terminated.

If an entry is found with non-zero outgoing cost, the maximum of the incoming and outgoing costs is used for the purposes of the path cost calculation, instead of only the incoming cost. This value is also used to increment the path cost field of the route request frame prior to retransmission.

4.2.4   Key Concept: Rapid Response

Rapid response allows a node that has been powered on or reset to rapidly acquire two-way links with its neighbors, minimizing the amount of time the application must wait for the stack to be ready to participate in routing. This feature is 100% ZigBee compatible.

If a link status message is received that contains no two-way links, and the receiver has added the sender to its neighbor table, then the receiver sends its own link status message immediately in order to get the sender started quickly. The message is still jittered by 2 seconds to avoid collisions with other rapid responders. To avoid a chain reaction, rapid responders must themselves have at least one two-way link.

4.2.5   Key Concept: Connectivity Management

By nature ZigBee devices are RAM-constrained, but often ZigBee networks are dense. This means that each router is within radio range of a large number of other routers. In such cases, the number of neighbors can exceed the maximum number of entries in a device's neighbor table. In such cases, the wrong choice of which neighbors to keep can lead to routing inefficiencies or worse — a disconnected network. The EmberZNet PRO stack employs 100% ZigBee-compatible algorithms to manage the selection of neighbors in dense networks to optimize network connectivity.

4.3   Route Discovery & Repair

Routing in ZigBee is automatically handled by the networking layer, and the application developer usually does not need to be concerned with its behavior. However, it is useful to have a feel for how the network behaves when a route needs to be discovered or repaired.

4.3.1   Route Discovery

Route discovery is initiated when a unicast message is sent from one device to another and there is no pre-existing route.

We assume that there is no existing route so the networking software will begin the process of route discovery. For simplicity, assume that the routing tables of all devices are blank.

For example, assume that device A needs to send a message to device C, as shown in Figure 4. Device A will broadcast a message to the entire network asking the device C to reply. This broadcast message also serves to establish a temporary route back to A, as each intermediate device records the device from which it received the message. Routes are updated on intermediate nodes — note that these are temporary entries that have a shorter lifetime than regular entries and are not intended for re-use. Because A is a one-hop neighbor, B and D do not need to store routing information about it.
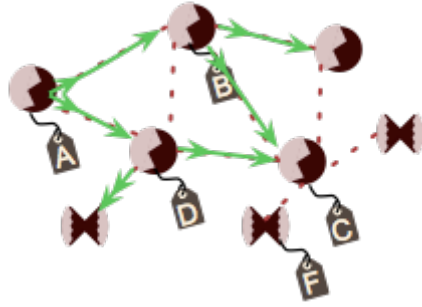
Figure 4. Example Network

C could use either B or D as its next hop back to A. ZigBee leaves this choice to the implementation; Silicon Labs uses a weighting algorithm to choose the most apparently reliable next hop.

When the message reaches device C, C sends a special unicast message (called a Route Response message) back to A using the temporary route constructed in step 1, as shown in Figure 5. This message is used by intermediate devices to establish a (permanent) route back to C.

Because C is a one-hop neighbor, B does not need to store routing information about it. D is not involved in this part of the discovery process because it was not selected by A in the above step. When the message reaches device A, the route discovery is complete and the new route can be used to send data messages from A to C, as shown in

ZigBee PRO networks will detect asymmetric RF links and avoid them during route discovery. This improves the reliability of the discovery process and the resulting routes.

Routes that have not been used within a certain timeout period (1 minute in EmberZNet 3.0 and later) are marked for re-use and new routes may then overwrite that memory location. In some cases a new route may be needed and one or more intermediate devices will not have an available routing table entry; in this case the message will be reported as undeliverable to the sending node.

The application specifies if an end-to-end acknowledgement should be sent by the receiver (this is called an APS acknowledgement). If yes, the sender will be notified upon successful delivery of in the case of a timeout waiting for acknowledgement. In the case of a timeout, the route may need to be repaired.
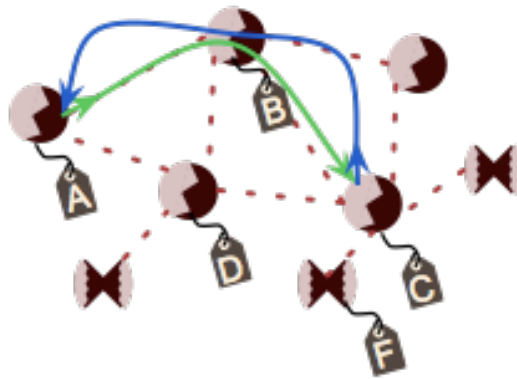

Figure 5. Unicast Message (Green) with Acknowledgement (Blue)

4.3.2   Route Repair

When a unicast message is sent with an acknowledgement requested, the sending device will be informed when the message is successfully delivered. If does not receive this acknowledgement, it can then take steps to repair the route. Route repair follows exactly the same steps as route discovery, above, but the damaged node (B, in Figure 6) does not participate, resulting in a different route choice.
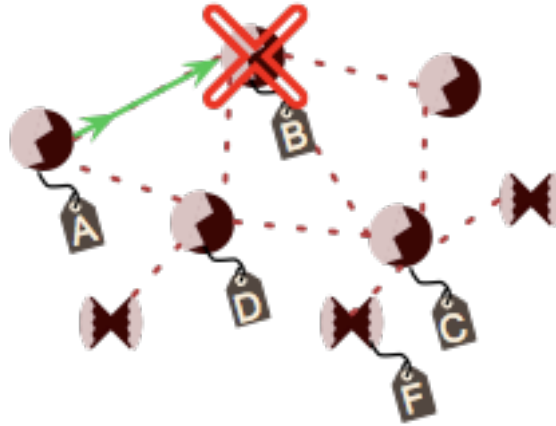
Figure 6. Network with a Damaged Node

The routing table for A is updated to reflect that the next hop is D and the message is successfully delivered along the new path, as shown in Figure 7.
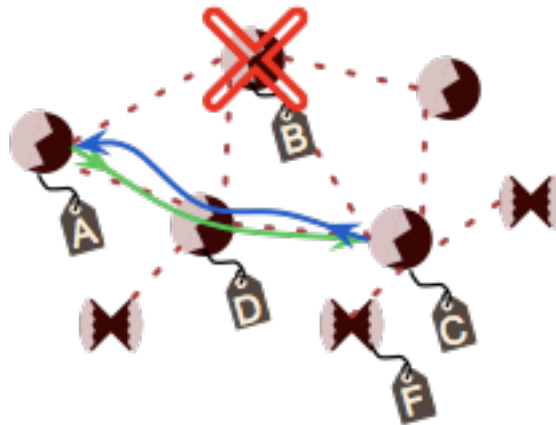

Figure 7. Alternative Route

In the case that no alternative path is available, the sender will be informed that the message could not be delivered. In EmberZNet PRO this is a 0×66 EMBER_DELIVERY_FAILED response.

EmberZNet PRO will attempt to deliver a message again before performing the route repair. Route repair is performed automatically when EMBER_APS_OPTION_RETRY and EMBER_APS_OPTION_ENABLE_ROUTE_DISCOVERY are both set in the message options.

4.4   Retries and Acknowledgements

ZigBee and its underlying network layers provide a system of retries and acknowledgments that are designed to efficiently manage the uncertainty of RF communication. It is not necessary to understand this concept in order to start using ZigBee but it may be of interest to some application developers in specific situation.

This section discusses retries and acknowledgments layer-by-layer:

- MAC retries and ACKs (802.15.4)
- NWK retries (ZigBee NWK layer)
- APS retries and ACKs (ZigBee APS layer)

4.4.1   MAC Retries and ACKs (802.15.4)

Figure 8 illustrates the MAC layer transmission retry process.
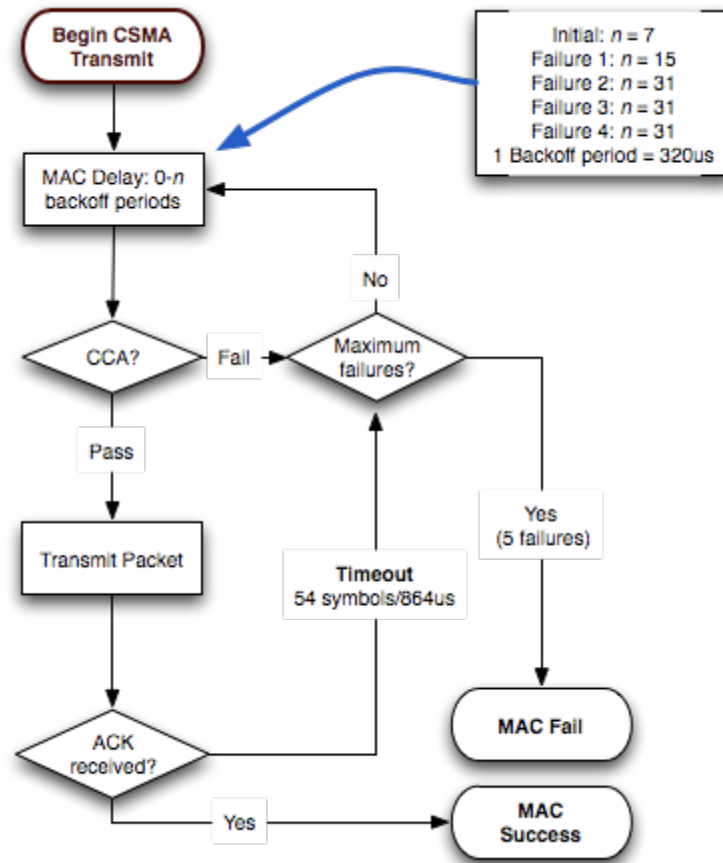
Figure 8. MAC Retry and ACK Process

The MAC layer attempts transmission five times.

- Unicast retries will occur if the channel was not clear (CCA fail) or if the MAC level ACK was not received from the next hop destination.
- Broadcast retries will occur in the case of CCA failure but broadcast does not use the MAC ACK capability.

These retries occur very quickly — maximum retry time for complete failure is approximately 37ms. Note that the MAC ACK is sent back immediately from the sender without additional CCA — see 802.15.4 documentation for more information.

4.4.2 NWK Retries

NWK retries in ZigBee are vendor specific. Figure 9 illustrates the EmberZNet PRO stack's NWK layer transmission retry process.
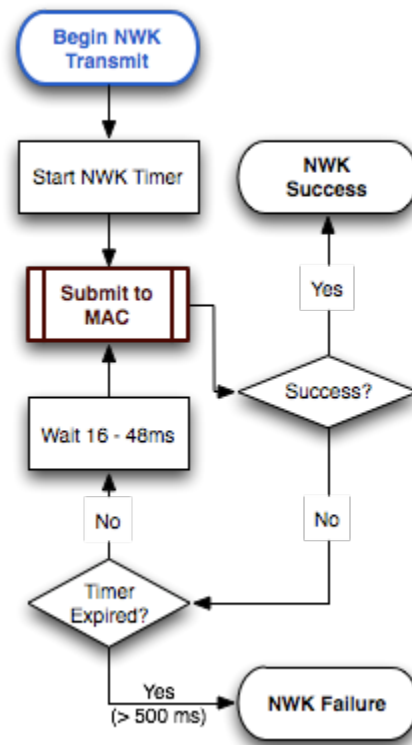

Figure 9. NWK Retry Process

The NWK retries occur only if the MAC layer indicates a transmission failure. They operate on a longer time scale than MAC retries and therefore give the network additional robustness in the presence of medium term (1-500ms) interference.

Silicon Labs' interference research shows that the NWK layer retries are important for overcoming temporary interference from WiFi in certain situations.

- Unicast: unicast behavior is as described in the flow chart, with retries for up to 500 ms.
- Broadcast: broadcast messages are re-sent every 500 ms up to a total of 3 times (including the initial broadcast) or until all neighbors are heard to rebroadcast the message themselves (thereby ensuring complete delivery).

4.4.3 APS Retries and ACKs

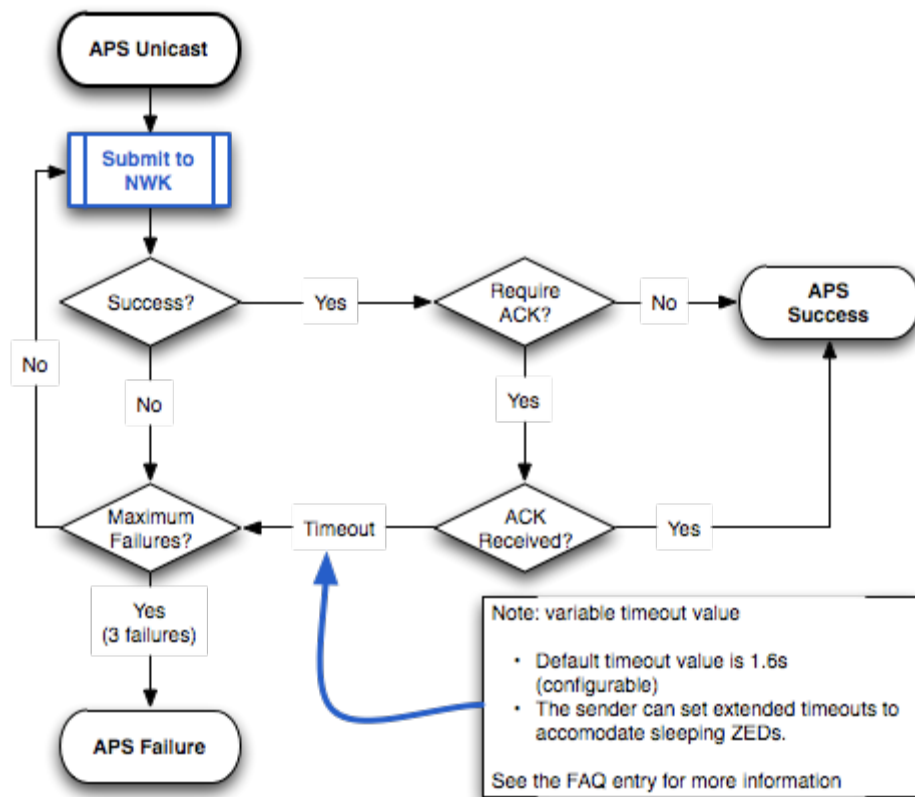Figure 10 illustrates the APS layer retry process.

Figure 10. APS Layer Retry Process

The APS layer has an ACK flag that controls whether it uses the additional logic to wait for an acknowledgment and retry if the acknowledgment is not heard. This represents a full end-to-end acknowledgment from the recipient device.

The APS layer can be further optionally configured to repair the route to the destination in the case that the APS sending fails.

There is no equivalent of the APS end-to-end acknowledgment for broadcast messages.

4.4.4 Conclusions

If it is possible to send a message to the destination, the automatic ZigBee behavior provides "maximum effort", by attempting retries over several different time scales, from 1ms to several seconds with optional end-to-end ACK and route repair.

If the delivery fails the application is recommended to wait a more significant amount of time before retrying — this gives the interference or failure time to clear up. In cases of extreme bandwidth congestion the application retries may actually contribute to the problem.

5   The ZigBee Stack

ZigBee provides several separate stacks. This has been a point of confusion so the summary is as follows:

- ZigBee 2004 – was released in 2004 and supported a home control lighting profile. This stack was never extensively deployed with customers and is no longer supported.
- ZigBee 2006 – was released in 2006 and supports a single stack known as the ZigBee stack (explained below)
- ZigBee 2007 – was released in Q4 2007. It has two feature sets, ZigBee and ZigBee PRO.

The ZigBee and ZigBee PRO stacks are complete implementations of the MAC, networking layer, security services and the application framework. Devices implementing ZigBee and ZigBee PRO can interoperate by acting as end devices in the other type of network. For example, if a network is formed around a ZigBee PRO coordinator it can have ZigBee PRO routers only, but both ZigBee and ZigBee PRO end devices.

Note: The ZigBee 2004 stack is not interoperable with ZigBee 2007.

The ZigBee stack is formed around a central coordinator and uses tree addressing to establish the network.

The ZigBee PRO stack is formed around a central coordinator but uses a stochastic addressing scheme to avoid limitations with the tree. Table routing is always used and additional features are available such as:

- Network level multicasts
- Many-to-one and source routing
- Frequency agility
- Asymmetric link handling
- PAN ID Conflict
- Fragmentation
- Standard or high security.

Silicon Labs does not recommend deploying systems on the ZigBee stack because the ZigBee PRO stack has a number of features that are necessary for long term network stability and reliability. The ZigBee stack is typically used for small static networks.

Note the ZigBee 2007 specification includes updates to the ZigBee stack to allow the use of frequency agility and fragmentation on this stack. This stack remains interoperable with the ZigBee 2006 stack and therefore use of these features must be limited to those networks which can handle the complexity of some devices having these capabilities and some devices not.

5.1   ZigBee Device Object (ZDO)

The ZDO (occupying Endpoint 0 of each node) is a stack-level entity defined by the ZigBee Networking Specification for use in network management and information gathering. This section explains how to make use of ZDO functionality in the EmberZNet PRO stack and/or EZSP. An example is described as well.

The ZDO, an entity in the stack, provides network management capabilities that nodes can use to learn about each other, about the network in general, or for managing certain stack-level functionality. Many features of the ZDO must be implemented as part of a ZigBee Certified Platform (ZCP) and are therefore available on all ZigBee/ZigBee PRO devices, making them useful as an interoperable way to gather and manage system properties on a ZigBee network. The ZDO is implemented through an OTA messaging format known as the ZigBee Device Profile.

The following basic guidelines apply to using the ZDO for network management:

- Broadcasts requests must go to broadcast mask 0xFFFD.
- Requests should always be sent to destination endpoint 0, with application profile ID 0x0000, as this indicates that the ZDO (intrinsic in the stack) should handle the message. The source endpoint generally corresponds to the portion of your application that is interested in the requested information.
- Responses to ZDO requests come from source endpoint 0, to the destination endpoint corresponding to the source used for the Request.
- If a particular ZDO request handler is not implemented in the stack, a status of EMBER_ZDP_NOT_SUPPORTED (0x84) will be returned in the response.
- Cluster ID is used to indicate the Request or Response type being sent.
- Most requests can be sent as unicast, broadcast or multicast, just like any other APS frame (with optional APS acknowledgement), and responses arrive in the IncomingMessageHandler like any other incoming APS message. However, certain kinds of requests, if sent as broadcasts, will not generate a response (in the interest of network bandwidth conservation). There are

also certain requests that cannot be sent broadcast. For more information on these specific commands, please refer to the ZigBee core specification (document 14-0392).

- `stack/include/zigbee-device-stack.h` details the frame formats for commands and responses, including which request types are supported by the EmberZNet PRO stack. Further details about the frame formats are described in the ZigBee Device Profile (ZDP) section of the ZigBee Networking Specification (ZigBee document 14-0392).

- Utility functions (provided as source code) for generating ZDO requests and parsing certain kinds of responses can be found in `app/util/zigbee-framework/zigbee-device-library.h`. (There is also an alternative version of this file for EZSP host applications.)

- All ZDO requests and responses begin with the transaction sequence number as the first byte of the payload. This allows a request to be matched up with a particular response.

As an example, take the case of a Permit Joining ZDO transaction, which facilitates remote control of the PermitAssociation flag for one or more devices in the network.

For this particular transaction, zigbee-device-stack.h explains the following:

```
1.    // Request: [duration:1] [permit:1]
2.    // Response: [status:1]
3.    #define PERMIT_JOINING_REQUEST 0x0036
4.    #define PERMIT_JOINING_RESPONSE 0x8036
```

So a Permit Joining request frame would use cluster ID 0x0036, and the response would arrive with cluster ID 0x8036. For the request parameters, the "duration" byte corresponds to the argument you would ordinarily provide to the PermitJoining call in EmberZNet PRO/EZSP (a number of seconds to allow joining, or 0x00 for always-off joining, or 0xFF for always-on joining). The "permit authentication" byte used to only be applicable when the recipient of the request was the Trust Center (TC) and controlled whether the TC should perform authentication on "certain devices" during future association requests.

However, the current ZigBee Networking Specification mandates that the permit authentication byte be 0x01, so this argument in the Permit Joining Request is no longer significant.

Some examples (current as of EmberZNet 3.0.1) of sending this request frame through the EZSP Training-Host application follow. (0x42 is used as an arbitrary value here for a sequence number; the application would substitute its own values in these places.)

Unicast to node 0x401C asking it to permit joining for the next 60 (0x3C) seconds:

```
sendunicast 0 0x401C 0x0000 0x0036 1 0 0x1140 0xFFFF 0x42 0x42 4 {42 3C 00}
```

This generates an APS ACK frame (because the RETRY bit is set in the APS options above) as well as an incoming unicast (Permit Joining Response) from the request's destination signifying the result of the request:

```
incomingMessage
type:UNICAST profileId:0x0000 clusterId:0x8036 sourceEndpoint:0
destinationEndpoint:0 options:RETRY | ENABLE_ROUTE_DISCOVERY
groupId:0x0000 sequence:10 lastHopLqi:255 lastHopRssi:0 sender:0x401C
bindingIndex:255 addressIndex:255 messageLength:2 messageContents: 42
00
```

Broadcast to entire network asking it to permit joining forever (0xFF):

```
sendbroadcast 0xfffd 0x0000 0x0036 1 0 0x0000 0xffff 0x42 10 0x42 3 {42 FF 00}
```

No response is generated as a result (since there could be potentially many responders), but devices can now join at any router in the network after this broadcast request has been processed.

5.1.1 Advanced ZDO Usage

Additionally, as of EmberZNet release 3.3.2, Silicon Labs has added the ability for the application (host MCU) to be informed about ZDO requests that the stack is handling, and/or informed about those that the stack does not handle (so that it can handle them itself), and also to give the host a chance to handle endpoint requests (instead of handling these automatically) if desired. Please see comments in the `hal/ember-configuration.c` file (in SoC platforms) or `app/util/ezsp/ezsp-enum.h` (for NCP platforms) in EmberZNet releases 3.3.2 and later for descriptions of these available options, which are described as EzspConfigId values 0×26-0×28 (in EZSP) or the following configuration defines (in SoC API):

```
EMBER_APPLICATION_RECEIVES_SUPPORTED_ZDO_REQUESTS
EMBER_APPLICATION_HANDLES_UNSUPPORTED_ZDO_REQUESTS
EMBER_APPLICATION_HANDLES_ENDPOINT_ZDO_REQUESTS
EMBER_APPLICATION_HANDLES_BINDING_ZDO_REQUESTS
```

5.2   ZigBee Profiles

Before ZigBee 3.0, application profiles, or simply profiles, sat on top of the basic ZigBee stack. These were developed to specify the OTA messages required for device interoperability. A given application profile could be certified on either the ZigBee or ZigBee PRO stack. Now, ZigBee 3.0 has introduced an all-encompassing application layer specification for defining OTA behavior for all ZigBee applications.

The following are the application profile groups that existed before ZigBee 3.0:

- Home Automation (HA) – Devices for typical residential and small commercial installations.
- Smart Energy (SE) – For utility meter reading and interaction with household devices.
- Commercial Building Automation (CBA) – Devices for large commercial buildings and networks.
- Telecom Application (TA) – Wireless applications within the telecom area.
- Health Care (HC) – Monitoring of personal health in the home or hospital environment.
- Retail – Monitoring and information delivery in a retail environment.
- ZigBee Light Link – Wireless control of LED lighting systems.

The ZigBee Cluster Library (ZCL) forms a generic basis for the ZigBee common application layer. This library defines common elements that are shared such as data types and allows reuse of simple devices such as on/off switch protocols between different profiles.

Application profiles defined the roles and functions of devices in a ZigBee network. Two types of application profiles were administered by the Alliance:

- Public Application Profiles, developed by members of the ZigBee Alliance so that devices from different manufacturers can interoperate.
- Manufacturer Specific Application Profiles, developed by product developers creating private networks for their own applications where interoperability is not required.

The ZigBee Alliance now recognizes one application layer, which defines roles and functions of a device in any ZigBee network. These application behaviors interoperate with any other application.

If you develop a product based upon your own private application layer specification, the ZigBee Alliance requires you to make use of a unique private profile identifier to ensure the product can successfully co-exist with other products. The ZigBee Alliance issues these unique private profile IDs to member companies upon request and at no charge.

An application can also be developed using the common ZigBee 3.0 application layer with private extensions for specific features from a manufacturer.

5.3   ZigBee Addressing Scheme

The ZigBee PRO stack uses a stochastic address assignment mechanism. Under this mechanism the coordinator is still used to start the network. Each device (routers and end devices) that joins the network is given a randomly assigned address from the device it is joining. The device conducts conflict detection on this address using network level messages to ensure the address is unique. This address is then retained by a device, even if the parent address changes.

Under ZigBee PRO, provisions are intended for a commissioning tool for setup and configuration of networks. This commissioning tool can be used to assign addresses manually.

5.4   Extended PAN IDs

Developers who have used stack software earlier than ZigBee 2007 should note a new and important change to PAN IDs. ZigBee has added an 8 byte extended PAN ID (EPID or XPID) to facilitate provisioning and PAN ID conflict detection. The extended PAN ID is now included in the beacon payload, following the existing 3 bytes.

The Extended PAN ID was a network parameter in ZigBee 2007 (EmberZNet 3.x and later) was used in the ZigBee and ZigBee PRO feature sets. This extended PAN ID [EPID] can be thought of as an extension of the basic, 16-bit PAN ID [PID]. The EPID is a 64-bit value set for the entire network by the ZigBee Coordinator [ZC] at the time the personal area network [PAN] is formed and must not change while the PAN is operating (unlike the PID). Like the PID, all nodes within the same PAN share an EPID.

Other than the scanning and joining processes, the EPID rarely appears in transmitted ZigBee packets due to its large overhead (8 bytes) in the header. However, the EPID is used during a Network Update transaction, where a Network Manager node informs the other devices

in the PAN about a PID conflict or channel change, so that nodes moving to a new channel and/or PID can rely on the consistency of the EPID as a criteria for finding the moved network.

Note that the Extended PAN ID (even if it's zero) is saved to the non-volatile memory (as part of the tokens) once the node associates into the network, so it won't change over the lifetime of the device until the device leaves the network.

5.4.1 Use in Scanning / Forming / Joining Process

The EPID will appear (along with other information such as the PID) in the beacon results seen by an Active Scan, and when a join is initiated, if the EPID is non-zero in the join request, the EPID will take precedence as the primary criteria for matching up the joiner to the joinee (16-bit PAN ID is effectively ignored). However, it's possible that some networks (particularly older ones) may not want (or be able) to deal with EPIDs, so the EPID would be zero in the network parameters from the beacon. If the requested EPID is all zeroes during the JoinNetwork operation, only the 16-bit PAN ID will be used to match to a network for the purposes of association.

While Silicon Labs provides some sample utilities for scanning networks and parsing the results to make a determination about which one to join, this process will generally vary depending on the application design and the level of strictness desired in selecting a network. (For example, the joining device may simply want any available ZigBee PRO network; it may seek a closed network designed specifically for the application in use but may not care which closed network it chooses; or it may seek one specific closed network with specific properties, in the way that I want my laptop at home to join not just any home WiFi network, but my WiFi network in my home.)

The following guidelines apply to EmberZNet in determining the expected behavior of a Form/Join action given a particular EPID:

- If an all-zero value is specified for extended PAN ID during FormNetwork, the stack will generate a random 64-bit value for this field.
- If an all-zero value is specified for extended PAN ID during JoinNetwork, the stack will use the 16-bit PAN ID specified in the JoinNetwork parameters as the primary criteria for selecting a network during the join process.
- If a non-zero value is specified for extended PAN ID during JoinNetwork, the stack will use the 64-bit extended PAN ID specified in the JoinNetwork parameters (even if 16-bit PAN ID is different) as the primary criteria for selecting a network during the join process.

5.4.2 Choosing an EPID

While the PAN ID is meant to be a randomly chosen, 16-bit value, unique to each network, the EPID is often used more like the SSID field of WiFi networks to give them a user-friendly designation (which is often not so random and is either set by the manufacturer or the installer). However, Silicon Labs discourages using a fixed EPID for all deployments because (unlike the conflicts of the PAN ID) EPID conflicts cannot be resolved if they occur at runtime (because no other unique information exists to distinguish the PANs). Customers wishing to use non-random EPIDs should, at minimum, scan the network (either at the coordinator or through some external commissioning tool) and check that the new PAN's preferred EPID is not already in use by some other PAN. One approach is to use a small set of preferred EPIDs when forming PANs so that the coordinator can have alternatives if the first choice is in conflict.

OEMs creating consumer-grade products requiring customer installation (rather than by trained installers) should take particular care to ensure variety of EPIDs, as two customers living next door to each other may purchase the same product for their homes and would prefer to isolate their networks from each other. If those two neighboring homes were to each use PANs with the same EPID, network difficulties would likely arise for both users because the both homes would be considered as one network, and many network address conflicts could occur.

5.4.3 EPID versus PID

Here is a quick overview of differences between the Extended PAN ID and the standard PAN ID:

- EPID is 64 bits; PID is 16 bits
- EPID is usually used as stack's criteria for matching to the requested network; PID is only used for matching criteria when EPID is all 0x00 bytes.
- EPID is only present in a few kinds of packets (beacons, Network Update messages); PID is present in almost all 802.15.4 frames (except MAC ACKs).
- EPID is used as criteria for uniquely identifying the network and for resolving conflicts of PID; PID is used for MAC destination filtering at the radio receiver.
- EPID may help provide some indication of the network's identity in the scan results; PID should always be completely random, so it is not as useful in determining which PAN is the "right one".

- EPID can be any value in range of 0x0000000000000001 to 0xfffffffffffffffe (all 0's and all F's are reserved values); PID can be any value in the range of 0x0000 - 0xFFFE (all F's constitute reserved value).

6 ZigBee Cluster Library

6.1 Overview

In the ZigBee Cluster Library (ZCL), a cluster is a set of messages used to send and receive related commands and data over a ZigBee network. For example, a temperature cluster would contain all the necessary OTA messages required to send and receive information about temperature.

To facilitate learning and management, these clusters are further grouped into functional domains, such as those useful for HVAC, Security, Lighting, and so on. Developers may also define their own clusters, if the pre-defined clusters do not meet their specific application needs.

The ZigBee common application layer then references which clusters are used within certain applications, and specifies which clusters are supported by different devices–some clusters are mandatory, others optional. In this way, the ZCL simplifies the documentation of a particular application and allows the developer to understand quickly which behaviors devices support.

A more detailed overview of the ZCL, the format of messages within clusters, and a set of messages that may be used within any cluster are described in the ZigBee Cluster Library Specification document (15-02017-002). Functional domain clusters are described in separate documents, such as the Functional Domain: Generic, Security and Safety document.

Silicon Labs provides source code to easily assemble and disassemble ZCL messages, whether they are pre-defined by the ZCL or custom messages created by the developer. In addition, Silicon Labs provides a powerful tool called AppBuilder that allows developers to create and configure most of their ZCL-based application. See document UG102, *Application Framework Developer Guide*, for more information.

6.2 Inside Clusters

6.2.1 Clients and Servers

Each cluster is divided into two ends, a client end and a server end. The client end of a cluster sends messages that may be received by the server end. The client end may also receive messages that are sent by the server end. In this sense, the client and server ends of a cluster are always complementary. In contrast to many other systems (for example, HTTP), both have the same potential for sending and receiving messages: the "client" designation does not imply a subordinate or response-only status.

Because all commands have a sender and a receiver, each cluster is described in two parts – a server part and a client part, as shown in Figure 11. A device supporting the server half of a cluster will communicate with a device supporting the client half of the same cluster.

	Server	Client
Attributes	On/Off	None
Received Commands	On Off Toggle	None
Generated Commands	None	On Off Toggle

Figure 11. A ZigBee Cluster

This equality complicates discussions; for clarity, this document always refers to "cluster end" when one of the client or the server end must be used, "cluster ends" when speaking of both client and server ends, and "cluster server" or "cluster client" when a specific end is required (usually examples).

6.2.2 Attributes

An attribute is data associated with a cluster end; the server and client ends of a cluster may each possess multiple attributes.

Each attribute declares a 16-bit identifier, a data type, a read-only or read/write designator, a default value, and an indicator of whether its support by any implementation is mandatory or optional. Table 1 lists example data types. The data types are fully described in the ZCL Specification. Table 2 is an excerpt from that document.

Table 1. Data Type Quick Reference (Example Data Types)

Data Type	Description
Binary data types	8, 16, 24, or 32 bits in length
Logical data type	Boolean
Bitmap data type	8, 16, 24, or 32 bits in length
Unsigned Integer	8, 16, 24, or 32 bits in length
Signed Integer:	8, 16, 24, or 32 bits in length
Enumeration:	8 or 16 bits in length
Floating Point:	2-byte semi-precision, 4-byte single precision, or 8 byte double precision
String:	binary octet string or character string; first byte is length
Time:	time of day, date
Identifier:	cluster ID, attribute ID
IEEE address type	8-byte unique IEEE address of an 802.15.4 radio

The attribute identifier is unique only within the specific cluster end: this means that the attribute 0x0002 within the cluster server does not need to be the same as the attribute 0x0002 within the cluster client, even within the same cluster.

Table 2. Example ZCL Data Types

Type Class	Data Type ID	Data Type	Length of Data (octets)	Invalid Number	Analog / Discrete
Null	0x00	No data	0	-	-
	0x01 – 0x7	Reserved	-	-	
General Data	0x08	8-bit data	1	-	D
	0x09	16-bit data	2	-	
	0x0a	24-bit data	3	-	
	0x0b	32-bit data	4	-	
	0x0c – 0x0f	Reserved	-	-	
Logical	0x10	Boolean	1	0xff	D
	0x11 – 0x17	Reserved	-	-	
Bitmap	0x18	8-bit bitmap	1	-	D
	0x19	16-bit bitmap	2	-	
	0x1a	24-bit bitmap	3	-	
	0x1b	32-bit bitmap	4	-	
	0x1c – 0x1f	Reserved	-	-	
Unsigned Integer	0x20	Unsigned 8-bit integer	1	0xff	A
	0x21	Unsigned 16-bit integer	2	0xffff	
	0x22	Unsigned 24-bit integer	3	0xffffff	
	0x23	Unsigned 32-bit integer	4	0xffffffff	
	0x24 – 0x27	Reserved	-	-	
Signed Integer	0x28	Signed 8-bit integer	1	0x80	A

Type Class	Data Type ID	Data Type	Length of Data (octets)	Invalid Number	Analog / Discrete
	0x29	Signed 16-bit integer	2	0x8000	
	0x2a	Signed 24-bit integer	3	0x800000	
	0x2b	Signed 32-bit integer	4	0x80000000	
	0x2c – 0x2f	Reserved	-	-	
Enumeration	0x30	8-bit enumeration	1	0xff	D
	0x31	16-bit enumeration	2	0xffff	
	0x32 – 0x37	Reserved	-	-	
Floating Point	0x38	Semi-precision	2	Not a Number	A
	0x39	Single precision	4	Not a Number	
	0x3a	Double precision	8	Not a Number	
	0x3b – 0x3f	Reserved	-	-	
String	0x40	Reserved	-	-	D
	0x41	Octet string	Defined in first octet	0xff in first octet	
	0x42	Character string	Defined in first octet	0xff in first octet	
	0x43 – 0x47	Reserved	-	-	
Array	0x48 – 0x4f	Reserved	-	-	-
List	0x50 – 0x57	Reserved	-	-	-
Reserved	0x58 – 0xdf	-	-	-	-
Time	0xe0	Time of day	4	0xffffffff	A
	0xe1	Date	4	0xffffffff	
	0xe2 – 0xe7	Reserved	-	-	
Identifier	0xe8	Cluster ID	2	0xffff	D
	0xe9	Attribute ID	2	0xffff	
	0xea	BACnet OID	4	0xffffffff	
	0xeb – 0xef	Reserved	-	-	
Miscellaneous	0xf0	IEEE address	8	0xffffffffffffffff	D
	0xf1 – 0xfe	Reserved	-	-	-
Unknown	0xff	Unknown	0	-	-

Attributes may be accessed over the air by use of the attribute commands described later in this document.

Global attributes were introduced in revision 6 of the ZCL. These attributes are defined for every cluster instance on a device and function as normal ZCL attributes. Cluster Revision (attribute ID 0xFFFD) is one example of a global attribute. This attribute is a way to version each of the individual cluster specifications, to improve backwards compatibility, and allow for testing of specific versioned cluster behavior. One device may read the Cluster Revision attribute on another device to determine how to communicate certain cluster behavior to them over the air. This Cluster Revision attribute is incremented for every major change to each cluster's specification.

6.2.3 Commands

A command is composed of an 8-bit command-identifier and a payload format. Like attributes, the 8-bit identifier is unique only within the specific cluster end. The payload format is arbitrary to the command type, conforming only to the general packet format guidelines as described in the ZCL Specification.

Commands are divided into two types: global and cluster-specific. Global commands are defined in the ZCL Specification and are not specific to any cluster. These global commands were originally referred to as Profile-Wide, but have changed name to fall in line with the ZigBee 3.0 common application layer. Cluster-specific commands are defined inside the cluster definitions in the ZCL functional domain documents, and are unique to the cluster in which they are defined.

6.2.3.1 Global Commands

Global commands are not unique to a specific cluster; they are defined in the ZCL General Command Frame.[2] Table 3 lists example profile-wide commands:

Table 3. Example Profile Wide Commands

Messages Sent to the Cluster End Supporting the Attribute	Messages Sent From the Cluster End Supporting the Attribute
Read Attributes	Read Attributes Response
Write Attributes	Write Attributes Response/No Response
Write Attributes Undivided	Write Attributes Response/No Response
Configure Reporting	Configure Reporting Response
Read Reporting Configuration	Read Reporting Configuration Response
Discover Attributes	Discover Attributes Response
Report Attributes	Report Attributes
Default Response	Default Response

- Read Attributes: Requests one or more attributes to be returned by the recipient; replies with Read Attributes Response.
- Write Attributes: Provides new values for one or more attributes on the recipient; the reply will contain a Write Attributes Response portion indicating which attributes were successfully updated, and/or a Write Attributes No Response portion for attributes that were not successfully updated.
- Write Attributes Undivided: Updates all attributes and replies with Write Attributes Response; if any single attribute cannot be updated, no attributes are updated and this command replies with Write Attributes No Response.
- Configure Reporting: Configures a reporting interval, trigger events, and a destination for indicated attributes. Replies with Configure Reporting Response.
- Read Reporting Configuration: Generates a Read Reporting Configuration Response containing the current reporting configuration sent in reply.
- Discover Attributes: Requests all supported attributes to be sent; replies with a Discover Attributes Response.
- Report Attributes: A report of attribute values configured by Configure Reporting command.
- Default Response: A response sent when no more specific response is available (and the default response is not disabled by the incoming message).

Since attributes are always tied to a cluster, the commands affecting attributes specify which cluster and which attributes are to be accessed or modified. Additionally, each cluster defines which attributes support which commands – for example, an attributes may be declared READ ONLY, in which case it will not support the Write Attributes command. Thus, while the command format is not cluster-specific, the attributes it describes and its result on the receiving system are both cluster-specific.

Readers interested in more detail about the format or specific behaviors of these messages should review the ZCL Specification (075123r02).

6.2.3.2 Cluster-Specific Commands

The payload format, support requirements (mandatory, optional), and behavior on receipt of a cluster-specific command are all defined in the cluster definition. Typically, these commands affect the state of the receiving device and may alter the attributes of the cluster as a side-effect.

For example, the ZCL defines three commands (OFF, ON, and TOGGLE) that are received by the On/Off cluster server. It further declares that each of these commands is mandatory and the payload format for each command (in this case, none of them have payloads). The ZCL defines that the On/Off cluster client is responsible for generating the commands received by the server.

6.3 Example: Temperature Measurement Sensor Cluster

As an example, consider a portion of the Temperature Measurement Sensor cluster that is fully described in chapter 4 of the ZigBee Cluster Library Specification document (15-02017-002)[3]. Table 4 lists some of the attributes from the ZCL specification.

---

[2] See Chapter 7, ZCL Specification 075123r02.
[3] All ZigBee document are available on the ZigBee website www.zigbee.org. ZigBee Alliance membership is required to access specification documents.

Table 4. Temperature Measurement Sensor Server Attributes (incomplete)

Identifier	Names	Types	Range	Access	Default	Mandatory/Optional
0x0000	MeasuredValue	Signed 16-bit Integer	MinMeasuredValue to MaxMeasuredValue	Read only	0	M
0x0001	MinMeasuredValue	Signed 16-bit Integer	0x954b – 0x7ffe	Read only	-	M
0x0002	MaxMeasuredValue	Signed 16-bit Integer	0x954c – 0x7fff	Read only	-	M
0x0003	Tolerance	Unsigned 16-bit Integer	0x0000 – 0x0800	Read only	-	O
0xFFFD	ClusterRevision	Unsigned 16-bit Integer	0x0001 – 0xFFFE	Read only	0x0001	M

Step 1. Overview of Attributes

The cluster server supports at least those five attributes, four of which must be supported by any implementation (MeasuredValue, MinMeasuredValue, MaxMeasuredValue, and ClusterRevision), and one of which may be optionally supported (Tolerance). All of these attributes are read-only, indicating that any write attempts to them will fail. Note the ClusterRevision global attribute functions as a regular attribute.

Step 2. Implications for Cluster Server, Cluster Client

Clearly, the cluster server should be implemented by the device that contains the temperature sensor. Meanwhile the cluster client should be implemented by any device that wishes to receive temperature sensor information, either actively (through a read attributes command) or passively (through first a configure report attributes command and then from report attributes).

Step 3. Further Information

The cluster description also provides useful information about the actual format of the data (for example, the range of the signed 16-bit integer for MaxMeasuredValue) and the mandatory supported operations – not all attributes support all basic commands. For example, MaxMeasuredValue is a read-only command and cannot be written.

Note: While the incoming write attribute commands are supported, in this case MaxMeasuredValue always generates a Write Attributes No Response reply.

Step 4. Commands

No custom commands are supported by this cluster (either the server or the client). For an example of a cluster containing custom commands, see the Thermostat Cluster in the ZCL (15-02017-002).

6.4   Functional Domains

As of this writing, the ZigBee Cluster Library defines the following functional domains:
- General
- Closures
- HVAC
- Lighting
- Measurement & Sensing
- Security & Safety
- Protocol Interfaces
- Smart Energy
- Over-the-Air Upgrading
- Telecommunications
- Commissioning
- Retail
- Appliances

Each domain defines a number of clusters that are then used by the ZigBee Application Layer to describe the OTA behavior of devices (see Figure 1213).
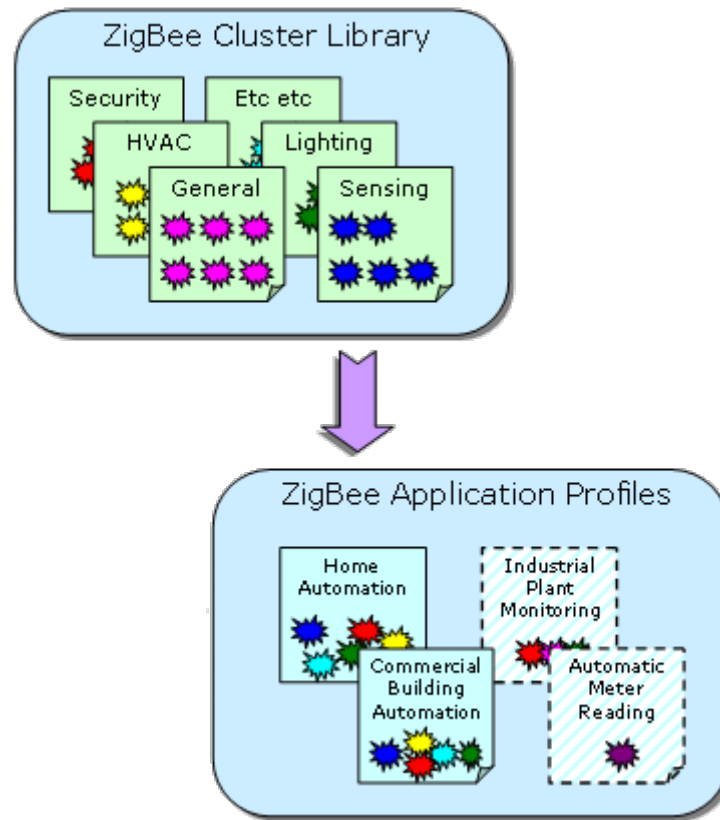

Figure 12. Cluster Library Functional Domains

6.5   Manufacturer Extensions

The ZCL allows extension of the existing library in two ways: users may add manufacturer specific commands or attributes to existing clusters, or they may define entirely new clusters that are manufacturer specific.

Manufacturer-specific commands are identified by setting a special bit in the ZCL header and including the manufacturer code (received from the ZigBee Alliance) in the ZCL header. This guarantees that manufacturer-specific extensions do not interfere with other manufacturer-specific extensions or existing ZCL clusters, commands, or attributes.

7   ZigBee Compliance

ZigBee compliance is based on a building block of compliance testing used to ensure that each layer is tested.

Products that meet all compliance requirements may be branded "ZigBee Compliant" and can display the ZigBee logo.

The ZigBee radio and MAC are required to have passed the applicable parts of IEEE 802.15.4 certification testing. Parts of the MAC not required for ZigBee operation are not required for this testing.

ZigBee stacks are required to be built on certified IEEE 802.15.4 platforms. ZigBee stack providers are required to have the stack tested against a standard ZigBee test plan known as ZigBee Compliant Platform Testing. This testing validates the basic network, security, and ZigBee Device Object (ZDO) operations for the stack.

ZigBee products are required to be built on a ZigBee Compliant Platform. The developer can choose to build a public or manufacturer-specific application. Those applications which are manufacturer-specific are tested against a specific set of tests to validate basic operation. Products based on the common ZigBee application layer are more extensively tested using a standard ZigBee test for the application layer commands. Testing scrutiny has been heavily increased in ZigBee 3.0; there are now many negative test cases that products must pass in order to pass certification.

The ZigBee Test Harness is new in ZigBee 3.0. The ZigBee Test Harness is provided to ZigBee members for testing their products and also is used for the final compliance testing for a ZigBee 3.0 product. The ZigBee Test Harness involves both a test harness ZigBee software stack running on a compliant IEEE 802.15.4 platform and a desktop application offering a friendly user interface to the tester.

Each shipping product has to be certified and substantive changes to the product after certification can require recertification. Silicon Labs sends all chips and stack releases through compliance testing.

All of the ZigBee test plans are developed and tested using at least three members' products to ensure interoperability between different implementations.

Companies are required to become members of the ZigBee alliance to ship product containing the ZigBee stack. There are specific rules for use of the ZigBee logo on product that are detailed on the ZigBee web site (www.zigbee.org).

8 ZigBee 3.0

ZigBee 3.0 marks the beginning of a new chapter in the ZigBee Alliance. ZigBee 3.0 unites all of the different application profiles into one common application layer. Furthermore, it introduces greater test coverage for product certification, ensuring better interoperability for ZigBee devices in the field. The ZigBee 3.0 document suite contains both revised and completely new material for the ZigBee application specification.

The Base Device Specification (BDB) is perhaps the most important new document in the ZigBee 3.0 document suite. It specifies mandatory and optional application layer behavior for any ZigBee product. These behaviors include, but are not limited to:

- Network commissioning
- Network security
- ZDO command handling
- Reporting
- MAC data polling
- Persistent data

See document 13-0402 for more specific information on these items. Because there is now one specification for all ZigBee applications, two different ZigBee 3.0-compliant products in different application domains can interoperate on the same network (for example, a light switch and a garage door opener can perform separate functions on the same network).

At the heart of the BDB are the commissioning methods that a ZigBee 3.0 device can use. The methods follow an order of execution, but they are all optional for a device. For example, it may only makes sense for a ZigBee End Device switch application to performing touchlink commissioning, classical joining, and then finding and binding. On the other hand, a ZigBee Coordinator gateway application may only wish to form a network. Lastly, maybe a ZigBee Router light application could look perform classical joining, and then fall back to classical network formation if it does not find any networks to join. The commissioning methods, and order, are as follows.

- Touchlink commissioning
- Classical network joining
- Classical network formation
- Finding and binding

Touchlink commissioning is the first step in the ZigBee 3.0 network commissioning process. A device may choose to perform the touchlink process for an initiator to try and find a touchlink target. A typical example of this interaction may be a switch and a light. The switch may perform touchlink commissioning to establish itself on the same network as the light. Please see 13-0402 for a more in-depth description of touchlink commissioning.

The next step in the BDB defined process is classical joining. This means finding open networks and joining one using the original ZigBee joining process. Finding a network involves performing IEEE 802.15.4 active scans on a product-defined set of channels. Once a device has found a network that it wishes to join, it may join that network using an install code derived link key, a default link key for a centralized security network, and a default link key for a distributed security network. For more information on the security mechanisms involved in the ZigBee 3.0 application layer, see *UG103.5, Application Development Fundamentals: Security*.

A ZigBee 3.0 application may then choose to form a classical ZigBee network. A product may form a centralized security network as a trust center or form a distributed security network as a router. This is different from many legacy application profiles where only a ZigBee coordinator could form a network. A light may wish to form a network as a router so that it can act as a touchlink target, whereas it makes more sense for a gateway to form a centralized network as a trust center. Trust centers may choose to only allow only certain devices on the network by using install code derived link keys for joining. Again, see *UG103.5, Application Development Fundamentals: Security* for a thorough discussion of ZigBee 3.0 security.

Once devices are on a network, they can perform finding and binding, where devices create bindings to establish application layer links. For example, an On/Off switch may perform finding and binding to create a binding to an On/Off Light. Again, as all commissioning steps, this step is optional.

As mentioned previously, ZigBee 3.0 also introduces more stringent testing on products. This means products will go through more negative test cases and specific cluster functionality tests before they are allowed on the market. Included in the ZigBee 3.0 document suite are the new Cluster Test Specifications. These documents contain test cases for ZigBee products implementing certain clusters. For a product using a certain cluster to be ZigBee 3.0 certified, it must pass the test cases contained within this document for that specific cluster. There is an individual ZigBee document for each cluster test spec. For an example, see document 15-0310, the On/Off Cluster test specification.

ZigBee 3.0 also mandates that devices should support proxy functionality for Green Power, a networking and application-level specification designed for energy-harvesting end devices. Essentially, this means that every ZigBee 3.0 device will be able to process and forward Green Power messages to other devices with more application-layer Green Power functionality. The Green Power Basic specification and test specification are included in the ZigBee 3.0 document suite.

It is worth mentioning that the 6th revision of the ZigBee Cluster Library (document 15-02017) is also included in the ZigBee 3.0 document suite. See Section 6, ZigBee Cluster Library, for more information about this specification.
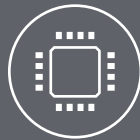
9 Applying ZigBee

A set of design decisions must be made in any ZigBee implementation. For more information, see *UG103.3, Application Development Fundamentals: Design Choices.*
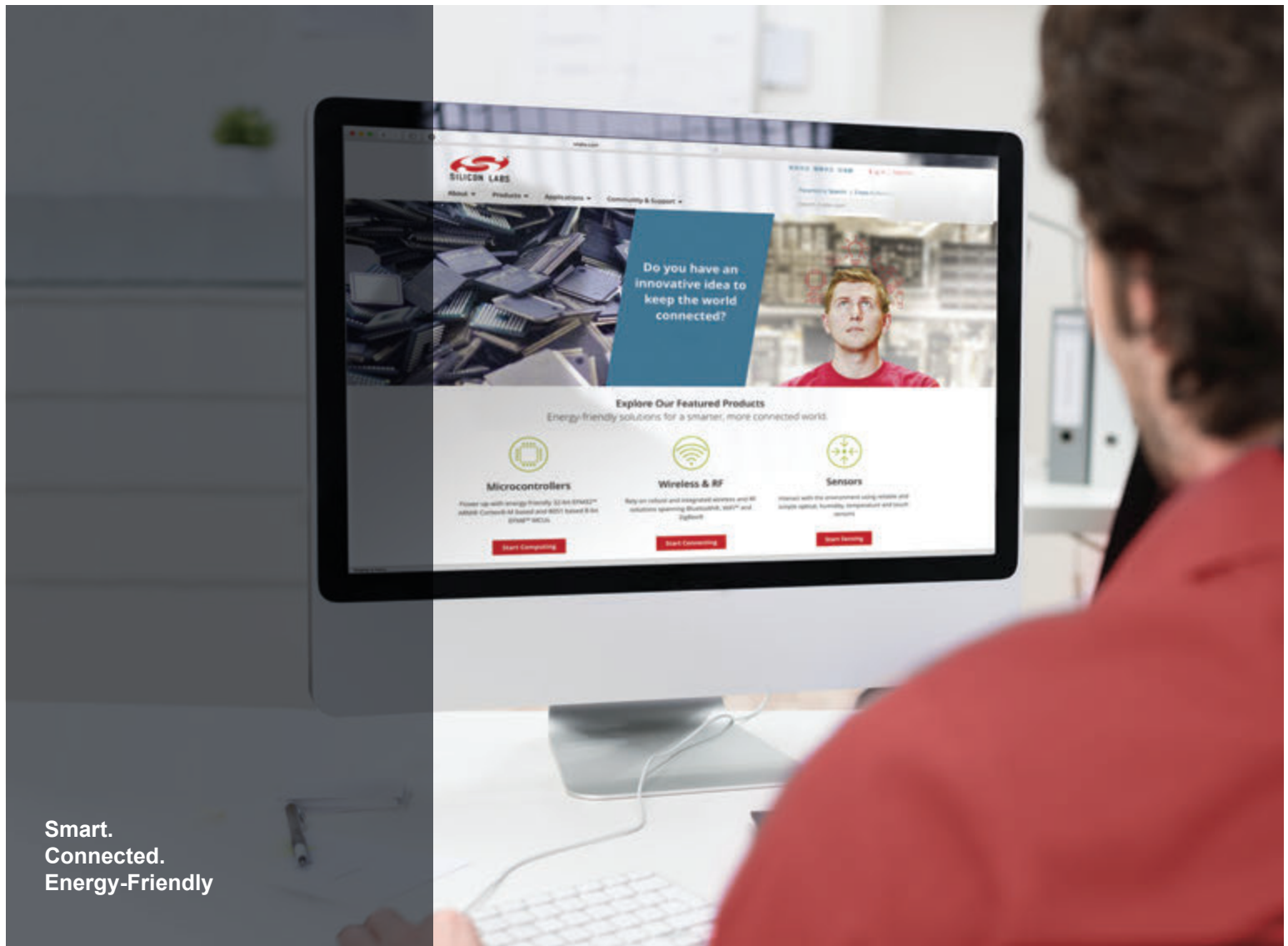
Smart.
Connected.
Energy-Friendly

Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com