Chipcon Products
from Texas Instruments

Z-Stack Developer's Guide

Document Number: F8W-2006-0022

Texas Instruments, Inc.
San Diego, California USA
(619) 542-1200

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

This document explains some of the components of the ZigBee stack and their functioning. It explains the configurable parameters in the ZigBee stack and how they may be changed by the application developer to suite the application requirements.

1.2 Scope

This document describes concepts and settings for the z-Stack Release v1.4.1. This is a ZigBee-2006 compliant stack.

1.3 Acronyms

AF	Application Framework
AIB	APS Information Base
API	Application Programming Interface
APS	Application Support Sub-Layer
APSDE	APS Date Entity
APSME	APS Management Entity
ASDU	APS Service Datagram Unit
MSG	Message
NHLE	Next Higher Layer Entity
NWK	Network
PAN	Personal Area Network
ZDO	ZigBee Device Object

1.4 Reference Documents

ZigBee Specification, R13, ZigBee Alliance document number 053474r13ZB.

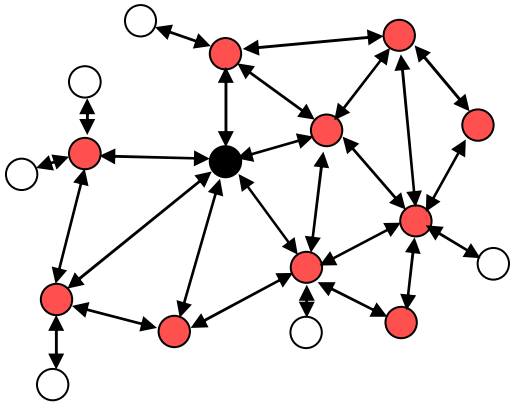
ZStack API (F8W-2006-0021)

2. ZigBee

A ZigBee network is a multi-hop network with battery-powered devices. This means that two devices that wish to exchange data in a ZigBee network may have to depend on other intermediate devices to be able to successfully do so. Because of this cooperative nature of the network, proper functioning requires that each device (i) perform specific networking functions and (ii) configure certain parameters to specific values. The set of networking functions that a device performs determines the role of the device in the network and is called a ***device type***. The set of parameters that need to be configured to specific values, along with those values, is called a ***stack profile***.

2.1 Device Types

There are three logical device types in a ZigBee network – (i) Coordinator (ii) Router and (iii) End-device. A ZigBee network consists of a Coordinator node and multiple Router and End-device nodes. Note that the device type does not in any way restrict the type of application that may run on the particular device.



An example network is shown in the diagram above, with the ZigBee coordinator (in black), the routers (in red) and the end devices (white).

2.1.1 Coordinator

This is the device that “starts” a ZigBee network. It is the first device on the network. The coordinator node chooses a channel and a network identifier (also called PAN ID) and then starts the network.

The coordinator node can also be used, optionally, to assist in setting up security and application-level bindings in the network.

Note that the role of the Coordinator is mainly related to starting up and configuring the network. Once that is accomplished, the Coordinator behaves like a Router node (or may even go away). The continued operation of the network does not depend on the presence of the Coordinator due to the distributed nature of the ZigBee network.

2.1.2 Router

A Router performs functions for (i) allowing other devices to join the network (ii) multi-hop routing (iii) assisting in communication for its child battery-powered end devices.

In general, Routers are expected to be active all the time and thus have to be mains-powered. A special mode of network operation, called “Cluster Tree”, allows Routers to operate on a periodic duty cycle and thus enables them to be battery-powered.

2.1.3 End-device

An end-device has no specific responsibility for maintaining the network infrastructure, so it can sleep and wake up as it chooses. Thus it can be a battery-powered node.

Generally, the memory requirements (especially RAM requirements) are lower for an end-device.

Notes:

In z-stack v1.4.1, the device type is usually determined at compile-time via compile options (ZDO_ COORDINATOR and RTR_NWK). All sample applications are provided with separate project files to build each device type.

It is possible to create an image with both Coordinator and Router functionality and choose the device type at runtime. See the SOFT_START compile option for more details.

2.2 Stack Profile

The set of stack parameters that need to be configured to specific values, along with those values, is called a ***stack profile***. The parameters that comprise of the stack profile are defined by the ZigBee Alliance.

All devices in a network must conform to the same stack profile (i.e., all devices must have the stack profile parameters configured to the same values).

The ZigBee Alliance has defined a stack profile for the ZigBee-2006 specification with the goal of promoting interoperability. All devices that conform to this stack profile will able to work in a network with devices from other vendors that also conform to it.

If application developers choose to change the settings for any of these parameters, they can do so with the caveat that those devices will no longer be able to interoperate with devices from other vendors that choose to follow the ZigBee specified stack profile. Thus, developers of "closed networks" may choose to change the settings of the stack profile variables. These stack profiles are called "network-specific" stack profile.

The stack profile identifier that a device conforms to is present in the beacon transmitted by that device. This enables a device to determine the stack profile of a network before joining to it. The "network-specific" stack profile has an ID of 0 while the ZigBee-2006 stack profile has ID of 1. The stack profile is configured by the `STACK_PROFILE_ID` parameter in nwk_globals.h file.

3. Addressing

3.1 Address types

ZigBee devices have two types of addresses. A 64-bit *IEEE address* (also called *MAC address* or *Extended address*) and a 16-bit *network address* (also called *logical address* or *short address*).

The 64-bit address is a globally unique address and is assigned to the device for its lifetime. It is usually set by the manufacturer or during installation. These addresses are maintained and allocated by the IEEE. More information on how to acquire a block of these addresses is available at http://standards.ieee.org/regauth/oui/index.shtml

The 16-bit address is assigned to a device when it joins a network and is intended for use while it is on the network. It is only unique within that network. It is used for identifying devices and sending data within the network.

3.2 Network address assignment

ZigBee uses a distributed addressing scheme for assigning the network addresses. This scheme ensures that all assigned network addresses are unique throughout the whole network. This is necessary so that there is no ambiguity about which device a particular packet should be routed to. Also, the distributed nature of the addressing algorithm ensures that a device only has to communicate with its parent device to receive a unique network-wide address. There is no need for network-wide communication for address assignment and this helps in scalability of the network.

The addressing scheme requires that some parameters are known ahead of time and are configured in each router that joins the network. These are the MAX_DEPTH, MAX_ROUTERS and MAX_CHILDREN parameters. These are part of the stack profile and the ZigBee-2006 stack profile has defined values for these parameters (MAX_DEPTH = 5, MAX_CHILDREN = 20, MAX_ROUTERS = 6).

The MAX_DEPTH determines the maximum depth of the network. The coordinator is at depth 0 and its child nodes are at depth 1 and their child nodes are at depth 2 and so on. Thus the MAX_DEPTH parameter limits how “long” the network can be physically.

The MAX_CHILDREN parameter determines the maximum number of child nodes that a router (or coordinator) node can possess.

The MAX_ROUTERS parameter determines the maximum number of router-capable child nodes that a router (or coordinator) node can possess. This parameter is a subset of the MAX_CHILDREN parameter and the remaining (MAX_CHILDREN – MAX_ROUTERS) address space are for end devices.

If developer wishes to change these values, they need to follow following steps:

First it must be ensured that the new values for these parameters are legal. Since the total address space is limited to about 2^{16}, there are limits on how large these parameters can be set to. The Cskip.xls file that is distributed in the release (in the Projects\zstack\Tools folder) can be used to verify this. After entering the values for the parameters into the spreadsheet, an error message will be given if the values are not legal.

After choosing legal values, the developer needs to ensure not to use the standard stack profile and instead set it to network-specific (i.e. change the STACK_PROFILE_ID in “nwk_globals.h” to NETWORK_SPECIFIC). Then the MAX_DEPTH parameter in “nwk_globals.h” may be set to the appropriate value.

In addition, the array’s CskipChldrn and CskipRtrs must be set in the nwk_globals.c file. These arrays are populated with the values for MAX_ CHILDREN and MAX_ ROUTERS value for the first MAX_DEPTH indices followed by a zero value.

3.3 Addressing in z-stack

In order to send data to a device on the ZigBee network, the application generally uses the AF_DataRequest() function. The destination device to which the packet is to be sent of type afAddrType_t (defined in “ZComDef.h”).


```
typedef struct
{
  union
  {
    uint16  shortAddr;
  } addr;
  afAddrMode_t addrMode;
  byte endPoint;
} afAddrType_t;
```

Note that in addition to the network address, the address mode parameter also needs to be specified. The destination address mode can take one of the following values (AF address modes are defined in “AF.h”)

```
typedef enum
{
  afAddrNotPresent = AddrNotPresent,
  afAddr16Bit      = Addr16Bit,
  afAddrGroup      = AddrGroup,
  afAddrBroadcast  = AddrBroadcast
} afAddrMode_t;
```

The address mode parameter is necessary because, in ZigBee, packets can be unicast, multicast or broadcast. A unicast packet is sent to a single device, a multicast packet is destined to a group of devices and a broadcast packet is generally sent to all devices in the network. This is explained in more detail below.

3.3.1 Unicast

This is the normal addressing mode and is used to send a packet to a single device whose network address is known. The `addrMode` is set to `Addr16Bit` and the destination network address is carried in the packet

3.3.2 Indirect

This is when the application is not aware of the final destination of the packet. The mode is set to `AddrNotPresent` and the destination address is not specified. Instead, the destination is looked up from a “binding table” that resides in the stack of the sending device. This feature is called Source binding (see later section for details on binding).

When the packet is sent down to the stack, the destination address is looked up from the binding table and used. The packet is then treated as a regular unicast packet. If more than one destination device is found, a copy of the packet is sent to each of them.

In previous versions of ZigBee (ZigBee04), there was an option to store the binding table on the coordinator. In that case, the sending device would send the packet to the coordinator which would then redirect the packet to the eventual destination that is found in its binding table. This optional feature is called Coordinator Binding.

3.3.3 Broadcast

This address mode is used when the application wants to send a packet to all devices in the network. The address mode is set to `AddrBroadcast` and the destination address can be set to one of the following broadcast addresses:

`NWK_BROADCAST_SHORTADDR_DEVALL` (0xFFFF) – the message will be sent to all devices in the network (includes sleeping devices). For sleeping devices, the message is held at its parent until the sleeping device polls for it or the message is timed out (NWK_INDIRECT_MSG_TIMEOUT in f8wConfig.cfg).

`NWK_BROADCAST_SHORTADDR_DEVRXON` (0xFFFD) – the message will be sent to all devices that have the receiver on when idle (RXONWHENIDLE). That is, all devices except sleeping devices.

NWK_BROADCAST_SHORTADDR_DEVZCZR (0xFFFC) – the message is sent to all routers (including the coordinator).

3.3.4 Group Addressing

This address mode is used when the application wants to send a packet to a group of devices. The address mode is set to afAddrGroup and the addr.shortAddr is set to the group identifier.

Before using this feature, groups have to be defined in the network [see the aps_AddGroup() in the ZStack API doc].

Note that groups can also be used in conjunction with indirect addressing. The destination address found in the binding table can be either a unicast or a group address. Also note that broadcast addressing is simply a special case of group addressing where the groups are setup ahead of time.

Sample code for a device to add itself to a group with identifier 1:

```
aps_Group_t group;

// Assign yourself to group 1
group.ID = 0x0001;
group.name[0] = 0;  // This could be a human readable string
aps_AddGroup( SAMPLEAPP_ENDPOINT, &group );
```

3.4 Important Device Addresses

An application may want to know the address of its device and that of its parent. Use the following functions to get this device's address (defined in ZStack API Doc):

- NLME_GetShortAddr() – returns this device's 16 bit network address.
- NLME_GetExtAddr() – returns this device's 64 bit extended address.

Use the following functions to get this device's parent's addresses (defined in ZStack API Doc). Note that the term "Coord" in these functions does not refer to the Zigbee Coordinator, but instead to the device's parent (MAC Coordinator):

- NLME_GetCoordShortAddr() – returns this device's parent's 16 bit short address.
- NLME_GetCoordExtAddr() – returns this device's parent's 64 bit extended address.

4. Binding

Binding is a mechanism to control the flow of messages from one application to another application (or multiple applications). In the Zigbee 2006 release, the binding mechanism is implemented in all devices and is called source binding.

Binding allows an application to send a packet without knowing the destination address, the APS layer determines the destination address from its binding table, and then forwards the message on to the destination application (or multiple applications) or group.

Notice: This is a change from the Zigbee 1.0 release, which stored all of the binding entries on coordinator. Now, all the binding entries are stored on the device that is sending the data.

4.1 Building a Binding Table

There are 3 ways to build a binding table:

- Zigbee Device Object Bind Request – a commissioning tool can tell the device to make a binding record.
- Zigbee Device Object End Device Bind Request – 2 devices can tell the coordinator that they would like to setup a binding table record. The coordinator will make the match up and create the binding table entries in the 2 devices.
- Device Application – An application on the device can build or manage a binding table.

4.1.1 Zigbee Device Object Bind Request

Any device or application can send a ZDO message to another device (over the air) to build a binding record for that other device in the network. This is called Assisted Binding and it will create a binding entry for the sending device.

4.1.1.1 The Commissioning Application

An application can do this by calling `ZDP_BindReq()` [defined in ZDProfile.h] with 2 applications (addresses and endpoints) and the cluster ID wanted in the binding record. The first parameter (target dstAddr) is the short address of the binding's source address (where the binding record will be stored).

Make sure you have the feature enabled in ZDConfig.h [`ZDO_BIND_UNBIND_REQUEST`].

You can use `ZDP_UnbindReq()` with the same parameters to remove the binding record.

The target device will send back a Zigbee Device Object Bind or Unbind Response message which the ZDO code will parse and notify ZDApp.c by calling `ZDApp_BindRsp()` or `ZDApp_UnbindRsp()` with the status of the action.

For the Bind Response, the status returned from the coordinator will be `ZDP_SUCCESS`, `ZDP_TABLE_FULL` or `ZDP_NOT_SUPPORTED`.

For the Unbind Response, the status returned from the coordinator will be `ZDP_SUCCESS`, `ZDP_NO_ENTRY` or `ZDP_NOT_SUPPORTED`.

4.1.1.2 Zigbee Device Object End Device Bind Request

This mechanism uses a button press or other similar action at the selected devices to bind within a specific timeout period. The End Device Bind Request messages are collected at the coordinator within the timeout period and a resulting Binding Table entry is created based on the agreement of profile ID and cluster ID. The default end device

binding timeout (APS_DEFAULT_MAXBINDING_TIME) is 16 seconds (defined in nwk_globals.h), but can be changed if added to f8wConfig.cfg.

The sample applications used in the "User's Guide" are examples of an End Device Bind implementation (pressing SW2 on each device).

You'll notice that all sample applications have a function that handles key events [for example, `TransmitApp_HandleKeys()` in TransmitApp.c]. This function calls ZDApp_`SendEndDeviceBindReq()` [in ZDApp.c], which gathers all the information for the application's endpoint and calls `ZDP_EndDeviceBindReq()` [ZDProfile.c] to send the message to the coordinator. Or, as in SampleLight and SampleSwitch, `ZDP_EndDeviceBindReq()` is called directly with only the cluster IDs relevant to the lamp On/Off functions.

The coordinator will receive [`ZDP_IncomingData()` in ZDProfile.c] and parse [`ZDO_ProcessEndDeviceBindReq()` in ZDObject.c] the message and call `ZDApp_EndDeviceBindReqCB()` [in ZDApp.c], which calls `ZDO_MatchEndDeviceBind()` [ZDObject.c] to process the request.

When the coordinator receives 2 matching End Device Bind Requests, it will start the process of creating source binding entries in the requesting devices. The coordinator follows the following process, assuming matches were found in the ZDO End Device Bind Requests:

1. Send a ZDO Unbind Request to the first device. The End Device Bind is toggle process, so the unbind is sent first to remove an existing bind entry.
2. Wait for the ZDO Unbind Response, if the response status is `ZDP_NO_ENTRY`, send a ZDO Bind Request to make the binding entry in the source device. If the response status is ZDP_SUCCESS, move on to the cluster ID for the first device (the unbind removed the entry – toggle).
3. Wait for the ZDO Bind Response. When received, move on to the next cluster ID for the first device.
4. When the first device is done, do the same process with the second device.
5. When the second device is done, send the ZDO End Device Bind Response messages to both the first and second device.

4.1.1.3 Device Application Binding Manager

Another way to enter binding entries on the device is for the application to manage the binding table for itself. Meaning that the application will enter and remove binding table entries locally by calling the following binding table management functions (ref. ZStack API Document – Binding Table Management section):

- `bindAddEntry()` – Add entry to binding table
- `bindRemoveEntry()` – Remove entry from binding table
- `bindRemoveClusterIdFromList()` – Remove a cluster ID from an existing binding table entry
- `bindAddClusterIdToList()` – Add a cluster ID to an existing binding table entry
- `bindRemoveDev()` – Remove all entries with an address reference
- `bindRemoveSrcDev()` – Remove all entries with a referenced source address
- `bindUpdateAddr ()` – Update entries to another address
- `bindFindExisting ()` – Find a binding table entry
- `bindIsClusterIDinList()` – Check for an existing cluster ID in a table entry
- `bindNumBoundTo()` – Number of entries with the same address (source or destination)
- `bindNumOfEntries()` – Number of table entries
- `bindCapacity()` – Maximum entries allowed
- `BindWriteNV()` – Update table in NV.

4.1.2 Configuring Source Binding

To enable source binding in your device include the REFLECTOR compile flag in f8wConfig.cfg. Also in f8wConfig.cfg, look at the 2 binding configuration items (NWK_MAX_BINDING_ENTRIES & MAX_BINDING_CLUSTER_IDS). NWK_MAX_BINDING_ENTRIES is the maximum number of entries in the binding table and MAX_BINDING_CLUSTER_IDS is the maximum number of cluster IDs in each binding entry.

The binding table is maintained in static RAM (not allocated), so the number of entries and the number of cluster IDs for each entry really affect the amount of RAM used. Each binding table entry is 8 bytes plus (MAX_BINDING_CLUSTER_IDS * 2 bytes). Besides the amount of static RAM used by the binding table, the binding configuration items also affect the number of entries in the address manager.

5. Routing

5.1 Overview

A mesh network is described as a network in which the routing of messages is performed as a decentralized, cooperative process involving many peer devices routing on each others' behalf.

The routing is completely transparent to the application layer. The application simply sends data destined to any device down to the stack which is then responsible for finding a route. This way, the application is unaware of the fact that it is operating in a multihop network.

Routing also enables the "self healing" nature of ZigBee networks. If a particular wireless link is down, the routing function will automatically find a new route that avoids that particular broken link. This greatly enhances the reliability of the wireless network and is one of the key features of ZigBee.

5.2 Routing protocol

The ZigBee implementation uses a routing protocol that is based on the AODV (Ad hoc On demand Distance Vector) routing protocol for ad hoc networks. Simplified for use in sensor networks, the ZigBee routing protocol facilitates an environment capable of supporting mobile nodes, link failures and packet losses.

When a router receives a unicast packet, from its application or from another device, the NWK layer forwards it according to the following procedure. If the destination is one of the neighbors of the router (including its child devices), the packet will be transmitted directly to the destination device. Otherwise, the router will check its routing table for an entry corresponding to the routing destination of the packet. If there is an active routing table entry for the destination address, the packet will be relayed to the next hop address stored in the routing entry. If an active entry can not be found, a route discovery is initiated and the packet is buffered until that process is completed.

ZigBee end-devices do not perform any routing functions. An end-device wishing to send a packet to any device simply forwards it to its parent device which will perform the routing on its behalf. Similarly, when any device wishes to send a packet to an end-device and initiate route discovery, the parent of the end-device responds on its behalf.

Note that the ZigBee address assignment scheme makes it possible to derive a route to any destination based on its address. In z-stack, this mechanism is used as an automatic fallback in case the regular routing procedure cannot be initiated (usually, due to lack of routing table space).

Also in z-stack, the routing implementation has optimized the routing table storage. In general, a routing table entry is needed for each destination device. But by combining all the entries for end-devices of a particular parent with the entry for that parent device, storage is optimized without loss of any functionality.

ZigBee routers, including the coordinator, perform the following routing functions (i) route discovery and selection (ii) route maintenance (iii) route expiry.

5.2.1 Route Discovery and Selection

Route discovery is the procedure whereby network devices cooperate to find and establish routes through the network. A route discovery can be initiated by any router device and is always performed in regard to a particular destination device. The route discovery mechanism searches all possible routes between the source and destination devices and tries to select the best possible route.

Route selection is performed by choosing the route with the least possible cost. Each node constantly keeps track of "link costs" to all of its neighbors. The link cost is typically a function of the strength of the received signal. By adding up the link costs for all the links along a route, a "route cost" is derived for the whole route. The routing algorithm tries to choose the route with the least "route cost".

Routes are discovered by using request/response packets. A source device requests a route for a destination address by broadcasting a Route Request (RREQ) packet to its neighbors. When a node receives an RREQ packet it in turn

rebroadcasts the RREQ packet. But before doing that, it updates the cost field in the RREQ packet by adding the link cost for the latest link. This way, the RREQ packet carries the sum of the link costs along all the links that it traverses. This process repeats until the RREQ reaches the destination device. Many copies of the RREQ will reach the destination device traveling via different possible routes. Each of these RREQ packets will contain the total route cost along the route that it traveled. The destination device selects the best RREQ packet and sends back a Route Reply (RREP) back to the source.

The RREP is unicast along the reverse routes of the intermediate nodes until it reaches the original requesting node. As the RREP packet travels back to the source, the intermediate nodes update their routing tables to indicate the route to the destination.

Once a route is created, data packets can be sent. When a node loses connectivity to its next hop (it doesn't receive a MAC ACK when sending data packets), the node invalidates its route by sending an RERR to all nodes that potentially received its RREP. Upon receiving a RREQ, RREP or RERR, the nodes update their routing tables.

5.2.2 Route maintenance

Mesh networks provide route maintenance and self healing. Intermediate nodes keep track of transmission failures along a link. If a link is determined as bad, the upstream node will initiate route repair for all routes that use that link. This is done by initiating a rediscovery of the route the next time a data packet arrives for that route. If the route rediscovery cannot be initiated, or it fails for some reason, a route error (RERR) packet is sent back to source of the data packet, which is then responsible for initiating the new route discovery. Either way the route gets reestablished automatically.

5.2.3 Route expiry

The routing table maintains entries for established routes. If no data packets are sent along a route for a period of time, the route will be marked as expired. Expired routes are not deleted until space is needed. Thus routes are not deleted until it is absolutely necessary. The automatic route expiry time can be configured in "f8wconfig.cfg". Set ROUTE_EXPIRY_TIME to expiry time in seconds. Set to 0 in order to turn off route expiry feature.

5.3 Table storage

The routing functions require the routers to maintain some tables.

5.3.1 Routing table

Each ZigBee router, including the ZigBee coordinator, contains a routing table in which the device stores information required to participate in the routing of packets. Each routing table entry contains the destination address, the next hop node, and the link status. All packets sent to the destination address are routed through the next hop node. Also entries in the routing table can expire in order to reclaim table space from entries that are no longer in use.

Routing table capacity indicates that a device routing table has a free routing table entry or it already has a routing table entry corresponding to the destination address. The routing table size is configured in "f8wconfig.cfg". Set MAX_RTG_ENTRIES to the number of entries in the (set to at least 4). See the section on Route Maintenance for route expiration details.

5.3.2 Route discovery table

Router devices involved in route discovery, maintain a route discovery table. This table is used to store temporary information while a route discovery is in progress. These entries only last for the duration of the route discovery operation. Once an entry expires it can be used for another route discovery operation. Thus this value determines the maximum number of route discoveries that can be simultaneously performed in the network. This value is configured by setting the MAX_RREQ_ENTRIES in "f8wconfig.cfg".

5.4 Routing Settings Quick Reference

Setting Routing Table Size	Set MAX_RTG_ENTRIES Note: the value must be greater than 4. (See f8wConfig.cfg)
Setting Route Expiry Time	Set ROUTE_EXPIRY_TIME to expiry time in seconds. Set to 0 in order to turn off route expiry. (See f8wConfig.cfg)
Setting Route Discovery Table Size	Set MAX_RREQ_ENTRIES to the maximum number of simultaneous route discoveries enabled in the network. (See f8wConfig.cfg)

6. Portable Devices

End devices, in Zigbee 2006, are automatically portable. Meaning that when an end device detects that its parent isn't responding (out of range or incapacitated) it will try to rejoin the network (joining a new parent). There are no setup or compile flags to setup this option.

The end device detects that a parent isn't responding either through polling (MAC data requests) failures and/or through data message failures. The sensitivity to the failures (amount of consecutive errors) is controlled by `MAX_POLL_FAILURE_RETRIES`, which is changeable in f8wConfig.cfg (the higher the number – the less sensitive and the longer it will take to rejoin).

When the network layer detects that its parent isn't responding, it will call `ZDO_SyncIndicationCB()`, which will initiate a "rejoin". The rejoin process will first orphan-scan for an existing parent, then scan for a potential parent and rejoin (network rejoin command) the network with the potential parent.

In a secure network, it is assumed that the device already has a key and a new key isn't issued to the device.

7. End-to-end acknowledgements

For non-broadcast messages, there are basically 2 types of message retry: end-to-end acknowledgement (APS ACK) and single-hop acknowledgement (MAC ACK). MAC ACKs are always on by default and are usually sufficient to guarantee a high degree of reliability in the network. To provide additional reliability, as well as to enable the sending device get confirmation that a packet has been delivered to its destination, APS acknowledgements may be used.

APS acknowledgement is done at the APS layer and is an acknowledgement system from the destination device to the source device. The sending device will hold the message until the destination device sends an APS ACK message indicating that it received the message. This feature can be enabled/disabled for each message sent with the `options` field of the call to `AF_DataRequest()`. The options field is a bit map of options, so OR in `AF_ACK_REQUEST` to enable APS ACK for the message that you are sending. The number of times that the message is retried (if APS ACK message isn't received) and the timeout between retries are configuration items in `f8wConfig.cfg`. `APSC_MAX_FRAME_RETRIES` is the number of retries the APS layer will send the message if it doesn't receive an APS ACK before giving up. `APSC_ACK_WAIT_DURATION_POLLED` is the time between retries.

8. Miscellaneous

8.1 Configuring channel

Every device must have a DEFAULT_CHANLIST (in f8wConfig.cfg) that controls the channel selection. For a Zigbee coordinator, this list will be used to scan for a channel with the least amount of noise. For Zigbee Routers and End Devices, this list will be used to scan for existing networks to join.

8.2 Configuring the PAN ID and network to join

This is an optional configuration item to control which network a Zigbee Router or End Device will join. The ZDO_CONFIG_PAN_ID parameter in f8wConfig.cfg can be set to a value (between 0 and 0x3FFF). A coordinator will use this value as the PANId of the network that it starts. A router or end-device will only join a network that has a PANId configured in this parameter. To turn this feature off, set the parameter to a value of 0xFFFF.

For further control of the joining procedure, the ZDO_NetworkDiscoveryConfirmCB function in the ZDApp.c should be modified.

8.3 Maximum payload size

The maximum payload size for an application is based on several factors. The MAC layer provides a constant payload length of 102. The NWK layer requires a fixed header size, one size with security and one without security. The APS layer has a required, but variable, header size based on a variety of settings, including the ZigBee Protocol Version, use of KVP (not supported in Zigbee 2006), APS frame control settings, etc. Ultimately, the user does not have to calculate the maximum payload size using the aforementioned factors. The AF module provides an API that allows the user to query the stack for the maximum payload size, or the maximum transport unit (MTU). The user can call the function, “afDataReqMTU” (see “af.h”) which will return the MTU, or maximum payload size.

```
typedef struct
{
  uint8              kvp;
  APSDE_DataReqMTU_t aps;
} afDataReqMTU_t;

uint8 afDataReqMTU( afDataReqMTU_t* fields )
```

Currently the only field that should be set in the “afDataReqMTU_t” structure is “kvp”, which indicates whether KVP is being used. The “aps” field is reserved for future use.

8.4 Leave Network

The ZDO Management implements the function, “ZDO_ProcessMgmtLeaveReq”, which offers access to the “NLME-LEAVE.request” primitive. The “NLME-LEAVE.request” allows a device to remove itself or remove a child device. The “ZDO_ProcessMgmtLeaveReq” removes the device based on the provided IEEE address. If a device removes itself, it will wait for approximately 5 seconds and then reset. Once the device resets, it will come back up in an idle state. It will not attempt to associate or rejoin. If a device removes a child device it will remove the device from the local “association table”. The NWK address will only be reused in the case where a child device is a ZigBee End Device. In the case of a child ZigBee Router, the NWK address will not be reused.

If the parent of a child device leaves the network, the child will stay on the network.

Although the "NLME-LEAVE.request" offers several optional parameters, ZigBee 2006 (as well as Texas Instrument's current implementation) limits the use of these parameters. Currently, the optional parameters ("RemoveChildren", "Rejoin", and "Silent") should be set to the default values used in "ZDO_ProcessMgmtLeaveReq". If these values are changed unexpected results may occur.

8.5 Descriptors

All devices in a ZigBee network have descriptors that describe that type of device and its applications. This information is available to be discovered by other devices in the network.

Configuration items are setup and defined in `ZDConfig.c` and `ZDConfig.h`. These 2 files also contain the Node, Power Descriptors and default User Descriptor. Make sure to change these descriptors to define your device.

8.6 Non-Volatile Memory Items

8.6.1 Network Layer Non-Volatile Memory

A ZigBee device has lot of state information that needs to be stored in non-volatile memory so that it can be recovered in case of an accidental reset or power loss. Otherwise, it will not be able to rejoin the network or function effectively.

To enable this feature include the `NV_RESTORE` compile option. Note that this feature must usually be always enabled in a real ZigBee network. The ability to turn it off is only intended to be used in the development stage.

The ZDO layer is responsible for the saving and restoring of the Network Layer's vital information. This includes the Network Information Base (NIB - Attributes required to manage the network layer of the device); the list of child and parent devices, and the table containing the application bindings. Also, if security is used, some information like the frame counters will be stored.

When a device starts up after a reset, this information is restored into the device. This information is used to restore the device in the network if the device is reset. In `ZDApp_Init,` a call to `NLME_RestoreFromNV()` instructs the network layer to restore its network state from values stored in NV. This function call will also initialize the NV space needed for the network layer if the space isn't already established.

8.6.2 Application Non-Volatile Memory

NV can also be used to save information specific to the application and the User Descriptor is a good example. The NV item ID for the User Descriptor is `ZDO_NV_USERDESC` (defined in ZComDef.h).

In `ZDApp_Init()`, `osal_nv_item_init()` is called to initialize the NV space needed for the User Descriptor. If this is the first time that this function is called for this NV item, the init function will reserve the space for the User Descriptor and set the default value to `ZDO_DefaultUserDescriptor`.

Then when the NV stored User Descriptor is needed, as in `ZDO_ProcessUserDescReq()` (in ZDObject.c), it calls `osal_nv_read()` to get the User Descriptor from NV.

To update the User Descriptor in NV, as in `ZDO_ProcessUserDescSet()` (in ZDObject.c), it calls `osal_nv_write()` to set the updated User Descriptor in NV.

Remember: the NV items are each unique and if your application creates its own NV item is must select an ID from the application value range (0x0201 – 0x0FFF).

9. Security

9.1 Overview

ZigBee security is built with the AES block cipher and the CCM mode of operation as the underlying security primitive. AES/CCM security algorithms were developed by external researchers outside of ZigBee Alliance and are also used widely in other communication protocols.

ZigBee offers the following security features:

- Infrastructure security
- Network access control
- Application data security

9.2 Configuration

In order to have a secure network, first all device images must be built with the preprocessor flag `SECURE` set equal to 1. This can be found in the "f8wConfig.cfg" file.

Next, a default key must be chosen. This can be set via the `DEFAULT_KEY` in the "f8wConfig.cfg" file. Ideally, this must be set to a random 128-bit value.

The default key can be preconfigured on each device in the network or it can be configured only on the coordinator and distributed to each device over-the-air as it joins the network. This is chosen via the `gPreConfigKeys` option in “nwk_globals.c” file. If it is set to TRUE, then the value of default key must be preconfigured on each device (to the exact same value). If it is set to FALSE, then the default key parameter needs to be set only on the coordinator device. Note that in the latter case, the key will be distributed to each joining device over-air. So there is a *“moment of vulnerability”* during the joining process during which an adversary can determine the key by listening to the on-air traffic and compromise the network security.

9.3 Network access control

In a secure network, the trust center (coordinator) is informed when a device joins the network. The coordinator has the option of allowing that device to remain on the network or denying network access to that device.

The trust center may use any logic to determine if the device should be allowed into the network or not. One option is for the trust center to only allow devices to join during a brief time window. This may be possible, for example, if the trust center has a “push” button. When the button is pressed, it could allow any device to join the network for a brief time window. Otherwise all join requests would be rejected. A second possible scenario would be to configure the trust center to accept (or reject) devices based on their IEEE addresses.

This type of policy can be realized by modifying the `ZDSecMgrDeviceValidate()` function found in the "ZDSecMgr.c" module.

9.4 Key Updates

The Trust Center can update the common Network key at its discretion. Application developers have to modify the Network key update policy. The default Trust Center implementation can be used to suit the developer's specific policy. An example policy would be to update the Network key at regular periodic intervals. Another would be to update the NWK key upon user input (like a button-press). The ZDO Security Manager(ZDSecMgr.c) API provides the necessary functionality via "ZDSecMgrUpdateNwkKey" and "ZDSecMgrSwitchNwkKey". "ZDSecMgrUpdateNwkKey" allows the Trust Center to broadcast a new Network key to all devices on the network. At this point the new Network key is stored as an alternate key in all devices. Once the Trust Center calls "ZDSecMgrSwitchNwkKey", a network wide broadcast will trigger all devices to use their alternate key.

9.5 Quick Reference

Enabling security	set `SECURE` = 1 (in f8wConfig.cfg)
Enabling preconfigured Network key	set `gPreConfigKeys` = TRUE (in nwk_globals.c)
Setting preconfigured Network key	set `defaultKey` = {KEY} (in nwk_globals.c)
Enabling/disabling joining permissions on the Trust Center	call ZDSecMgrPermitJoining() (in ZDSecMgr.c)
Specific device validation during joining	modify `ZDSecMgrDeviceValidate` (in ZDSecMgr.c)
Network key updates	call ZDSecMgrUpdateNwkKey() and ZDSecMgrSwitchNwkKey() (in ZDSecMgr.c)