# Application Note: Application-Level Tuning of Z-Stack

Document Number: SWRA202

Texas Instruments, Inc.
San Diego, California USA

| Version | Description | Date |
|---|---|---|
| 1.0 | Initial release. | 12/08/2006 |
| 1.1 | Changed document name. Updated title page. | 05/21/2007 |
| 1.2 | Updated for 2.1.0 release | 06/16/2008 |
| 1.3 | Updated for 2.2.0 release | 04/02/2009 |

# Table of Contents

## 1. Purpose

This document discusses the various global variables, compile options, and compiler directives that affect Z-Stack usage and configuration. The most common compiler command-line options used to define a Z-Stack device can be set by editing a configuration file called f8wConfig.cfg. This file can be found in ..\Projects\zstack\Tools\xxxxx, where xxxxx depends on the Z-Stack platform used.

## 2. Definitions

The following terms are used in this document:

**APS** – Application support sub-layer

**MT** – Monitor Test

**NV** – Non-volatile

**AES** – Advanced Encryption Engine

## 3. General Considerations

In general, the default settings should be used. Unless one wants to tune the stack for their application requirements, the default settings are adequate for most applications.

## 4. Security Global Variables

The following section is dedicated to security global variables.

**Pre-configured NWK Key**

```
/*********************************************************************
 * SECURITY GLOBAL VARIABLES
 */
// This is the default pre-configured key,
// change this to make a unique key
// SEC_KEY_LEN is defined in ssp.h.
CONST byte defaultKey[SEC_KEY_LEN] =
{
  // Key for In-House Testing
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
  0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
}; ( defaultKey is defined in nwk_globals.c)

// If true, preConfigKey should be configured on all devices on the network
// If false, it is configured only on the coordinator and sent to other devices upon joining
uint8 zgPreConfigKeys = FALSE;      //  TRUE;
```

If zgPreConfigKeys is set to TRUE, all devices should use the same pre-configured security key. If zgPreConfigKeys is set to FALSE, the pre-configured key is set only on the coordinator device, and is handed to joining devices. The key is sent in the clear over the last hop. Upon reset, the device will retrieve the pre-configured key from NV memory if the NV_INIT compile option is defined (the NV item is called ZCD_NV_PRECFGKEY).

**Trust Center Link Key**

This is the trust center link key used as part of the Smart Energy profile secure joining process (see Z-Stack Smart Energy Developer's Guide for more details).

```
// This is the default pre-configured Trust Center Link key,
// change this to make a unique key, SEC_KEY_LEN is defined in ssp.h.
CONST byte defaultTCLinkKey[SEC_KEY_LEN] =
{
  0x56, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77,
  0x77, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77
};
```
***( defaultTCLinkKey is defined in nwk_globals.c)***

If the NV_INIT compile option is defined, the stack uses the value loaded into zgPreConfigTCLinkKey from NV memory (the NV item is called ZCD_NV_SECURE_PRECFG_TCLINKKEY).

## 5. Compile Options

The following compile options are used for debug trace and monitor test purposes.

***MT_TASK***

This compile option enables the device to be able to talk to the Z-Tool PC application, and to be able to output minimal debug information using the debug_str() function. Removing this saves some code and RAM usage.

***MT_ZDO_FUNC and other MT compile options***

MT_ZDO_FUNC is an example of a compile option that enables the ZDO commands to be used with Z-Tool. Most devices would not need all of the monitor test commands to be enabled. Each ZDO command can be enabled individually in ZDConfig.h. Selectively enabling requests that need to be in the application minimizes code usage. See the Z-Stack Compile options document for more information on the different MT compile options.

***LCD_SUPPORTED***

This turns on support for displaying debug trace information on development boards that support an LCD. On development kit hardware platforms that do  not support LCDs, setting LCD_SUPPORTED enables sending information to the Serial Port (since it does not have an LCD display). Enabling this compile option uses a significant portion of RAM.

***LCD_SUPPORTED=DEBUG***

Setting LCD_SUPPORTED=DEBUG enables sending information to both the LCD and Serial Port, so code is actually larger. On platforms that do not support LCDs, debug information is only sent to the Serial Port, therefore using less code.

### *BLINK_LEDS*

The BLINK_LEDS compile option enables extended LED functionality at the expense of extra code and RAM space.

See the Z-Stack Compile Options document for more information.

## 6. Compiler Directives

The following compiler directives allow one to minimize RAM usage where applicable, or change different user settings.

### *ZDAPP_CONFIG_PAN_ID*

If this is set to something other than 0xFFFF (legal values are 0-0xFFFE), the coordinator will start with that PAN ID, and routers and end devices will join the PAN set to that ID. If the NV_INIT compile option is defined, the user can recall this value from NV memory (the NV item is called ZCD_NV_PANID). The user can configure this value in NV memory using Z-Tool.

### *DEFAULT_CHANLIST*

This is where the user can configure the channel list supported by their device.

### *NUM_DISC_ATTEMPTS*

Controls the number of times a network discovery is performed before trying to join a network. The value of zdoDiscCounter is initialized to 1 in ZDApp.c. This is tested against NUM_DISC_ATTEMPTS. If it is greater, then it will call NLME_JoinRequest and associate to the chosen router. Example: Setting NUM_DISC_ATTEMPTS to 0 will only cause the device to do a beacon request once (i.e. only one scan). By default 3 scans are performed, since zdoDiscCounter needs to reach 4 to break out of the scanning state.

### *MAX_RESUME_RETRY*

Number of times to try orphan join before going to rejoin mode. With a default value of 1, the end device only sends out one orphan notification over the air. If a coordinator alignment isn't received, the end device will assume no previous parent exists on the network and will go to rejoin mode

### *NWK_START_DELAY*

This #define gives the user the option of waiting NWK_START_DELAY number of milliseconds before starting up the device on the network.

### *NWK_RETRY_DELAY*

If starting the device failed (due to channel busy or no network found), this governs the time to wait to try again

### *BEACON_REQUEST_DELAY*

Delay between each beacon request in a joining cycle

### *STARTING_SCAN_DURATION*

The amount of time spent scanning on each channel

### *MAX_SCAN_DURATION*

The maximum amount of time spent scanning on each channel

### *NWK_MAX_DEVICE_LIST*

This is used to pre-allocate RAM for the child devices. Changing NWK_MAX_DEVICE_LIST by 1 changes the size of the association table by 18 bytes. This setting should match the maximum number of children that a parent node can support.

### *STACK_PROFILE_ID*

By default the **STACK_PROFILE_ID** is set to the HOME_CONTROLS stack profile. The stack profile ID determines the settings for the Cskip parameters, NWK_MODE, and the type of security being used (if used). Therefore, changing the stack profile ID will typically allow the user to change the Cskip parameters using one #define. If one needs to define their own topology, one can use the NETWORK_SPECIFIC stack profile ID.

### MAX_NODE_DEPTH

Refers to how many successive parent-child links are possible and thus imposes a maximum range of the network. This parameter is not applicable when using the Zigbee Pro stack profile ID since Zigbee Pro uses stochastic addressing as its address assignment scheme.

These should be set based on how dense your network is expected to be and what ratio of router to end-device devices you expect.

### *MAX_BCAST*

This is the number of broadcast entries (for data packets) that can be held for the period of BCAST_DELIVERY_TIME. Together, these two settings enforce the settling time for broadcast data packets traversing through the network. For example, if MAX_BCAST_ENTRIES is set to 9, and BCAST_DELIVERY_TIME is set to 10 (seconds), a device could send out 9 sequential broadcast data packets, but would not be able to send out a 10th one until after 10 seconds.

### *PASSIVE_ACK_TIMEOUT*

The amount of time the network layer waits before re-transmitting a broadcast message if it didn't hear the re-transmission from its neighbors.

### *MAX_BCAST_RETRIES*

Number of retries for a broadcast message based on not hearing a neighbor reply in the amount of time defined by PASSIVE_ACK_TIMEOUT.

### *BCAST_DELIVERY_TIME*

The length of time a broadcast message is kept in the broadcast table. Set to 500ms more than retry time. Retry time is defined as PASSIVE_ACK_TIMEOUT * (MaxBroadCastRetries + 1). If a broadcast message is being sent to end devices, then this should be set in accordance with the polling rate of the end devices. For example 200 might be a good number if the poll rate was set to 10 seconds for the end device.

### *MAX_RTG_ENTRIES*

Defines the actual number of routing table entries available for general use and for use only in the case of route repair. Routing entries refer to the number of routes that go through a particular node. If all devices are sending data to a single device, there will be only route (because data has to reach only a single destination). Otherwise there can be as many routes as there are destinations. But it is unlikely all routes will not go through any given device.

A gateway (concentrator) device ideally should have as many routing entries as devices it wishes to send messages to, or have less, and let route entries expire. This will however let the system incur the overhead of doing route discovery if a route doesn't exist for a particular destination. The number of routing table entries required can be mitigated by using the Many-to-one/Source Routing feature of Zigbee Pro.

### *ROUTE_EXPIRY_TIME*

This setting determines the number of seconds before a route expires if no data is transmitted on it (by default set to 30 which means that established routes will expire in 30 seconds from the time they are recorded). If routes change often (due to router devices moving locations or end devices moving around within the network), one will want the old routes to expire. The expiry time is reset to this value, each time a packet goes over a particular route. Therefore, routes only expire if there is no traffic along it. After a route expires, the routing table entry will be deleted. It is recommended to set this value to 0 to never allow routes to expire if the value of MAX_RTG_ENTRIES is proportional to the number of devices to be reached on the network. The route expiry time is not so much a factor when the Many-to-one/Source routing feature of Zigbee Pro is used.

### *NWK_INDIRECT_MSG_TIMEOUT*

Time to hold a message for a sleeping End Device. This setting depends on the end device poll rate. One should make this greater than the end device poll rate.

### *NWK_INDIRECT_CNT_RTG_TMR*

Constant for seconds interval when calculating the indirect message timeout. Multiply this value by NWK_INDIRECT_MSG_TIMEOUT to get the number of seconds that an indirect message gets buffered.

### *POLL_RATE*

Milliseconds to wait between data request polls to the parent.

### *QUEUED_POLL_RATE*

After receiving a data indication to poll immediately for queued messages. Units are in milliseconds.

### *RESPONSE_POLL_RATE*

After receiving a data confirmation to poll immediately for response messages. Units are in milliseconds

### *MAX_POLL_FAILURE_RETRIES*

Number of times an end device will poll its parent before indicating a loss of synchronization. Once this threshold has been exceeded, the end device will attempt a Network Rejoin Request (per the Zigbee Specification). Note that a larger value will cause longer delays for the child to rejoin the network.

By default, if any two poll requests are not mac ack'd in a row, then this causes a link failure to be flagged in the stack. On every message sent out, neighbor table maintenance is performed. If two poll requests in a row are not mac ack'd, this causes the forceLinkDown variable to be set to TRUE. This ultimately causes ZDO_SyncIndicationCB to be called, which notifies the application that a sync loss with the parent occurred. Because there are 3 mac level retries on data packets, and MAX_DATA_RETRIES is set to 2 by default, this results in a total of 8 physical data request packets over the air missed in a row before sync indication is called. The equation would be something like:
No. of OTA retries before sync loss indication =
(MAX_POLL_FAILURE_RETRIES=MAX_DATA_RETRIES)*MAC_MAX_FRAME_RETRIES.
MAC_MAX_FRAME_RETRIES is set to 3 by default in mac_pib.c

### *REJOIN_POLL_RATE*

This is used as an alternate response poll rate only for the network rejoin request. This rate is determined by the response time of the parent that the device is trying to join.

### *NWK_MAX_DATA_RETRIES*

This controls the number of retries that is perform by the network layer for the next hop message once the MAC layer retries are exhausted for that message (note that MAC layer will retry 3 times before returning a failure status to the network layer).

### *Cskip Array (Zigbee non-Pro ONLY)*

The Cskip array needs to be edited by the user if they are attempting to change the network toplogy configuration.

For example if one uses:

```
 #if ( STACK_PROFILE_ID == HOME_CONTROLS )
   byte CskipRtrs[MAX_NODE_DEPTH+1] = {6,6,6,6,6,0};
   byte CskipChldrn[MAX_NODE_DEPTH+1] = {20,20,20,20,20,0};
```

then any router or coordinator can have a maximum of 20 child devices, and out of those 20, 14 can be end devices, 6 can be router devices.

### *MAX_NEIGHBOR_ENTRIES*

A neighbor is defined as anyone that the device hears from. i.e. a device within in next hop range. As seen from the description of MAX_POLL_FAILURE_RETRIES, the neighbor table is checked for each poll request. It is then important to note that if a neighbor table entry doesn't exist, and if there is no room in the neighbor table to create one, then the ZDO_SyncIndicationCB will never be called and the device will never try to rejoin the network. So it is advised to make this table as large as possible on the end device. It is also why clearing the neighbor table is required as part of the ZDO_SyncIndicationCB code. The size of the neighbor table becomes even more important when the SECURE=1 compile option is turned on since the received frame counter is stored here. Incoming messages will not be processed if the link info doesn’t exist for the received frame.

***NWK_MAX_BINDING_ENTRIES***

This controls the maximum number of entries in the binding table. If binding is not used, set this to 1 to be minimize RAM usage. Adjusting the number of entries by 1 changes the size of the binding table by 12 bytes.

***MAX_BINDING_CLUSTER_IDS***

This controls the maximum number of cluster IDs for each binding table entry. Set this to 1 if binding is not used in order to minimize RAM usage.

***APSC_MAX_FRAME_RETRIES***

This controls the number of retries that will be attempted by the APS layer if APS acknowledgement service is used for the transmitted message. In order to use APS acknowledgement service, pass into the options parameter a value logically OR'd with the bitmask of AF_ACK_REQUEST (0x10) when calling the AF_DataRequest() function.

***APSC_ACK_WAIT_DURATION_POLLED***

This controls the time in milliseconds between retries when an APS ACK is not received. This is in addition to nwk layer retries governed by MAX_DATA_RETRIES