Application Note

How to Modify TX and RX Commands Imported/Exported from SysConfig or SmartRF™ Studio

Siri Johnsrud

ABSTRACT

This document goes through the necessary changes needed to have the rfPacketRx and rfPacketTx examples in simplelink_cc13xx_cc26xx_sdk_x_xx_xx_xx [1] work with both the standard and the advanced TX and RX commands (one and two length bytes), and how to modify the code to be able to receive and transmit packets using the 802.15.4g format.

When using one length byte, the payload is limited to 255 bytes. When using two length bytes, the payload is limited to 4093 bytes, and when using the 802.15.4g format, the length field is 11 bits long, and the payload is limited to 2045 bytes.

To transmit or receive packets longer than 4093 bytes, unlimited packet length must be used. This is not discussed in the app note.

Table of Contents

Trademarks

1 Introduction

Both SmartRF™ Studio (2) and SysConfig (3) can export/import register settings for all the characterized PHYs available for a specific device. For the Proprietary PHY group, settings are either exported/imported using the standard TX and RX commands (CMD_PROP_TX and CMD_PROP_RX) or the advanced commands (CMD_PROP_TX_ADV and CMD_PROP_RX_ADV). The Basic RX and TX SLA (Simplelink Academy) showcases how to import settings from SysConfig and/or export settings from SmartRF Studio. Even if both command sets are very flexible in terms of supported packet formats, only two different packet formats are implemented in the default code export/import. Figure 1-1 is showing the packet format configured for the two different cases.

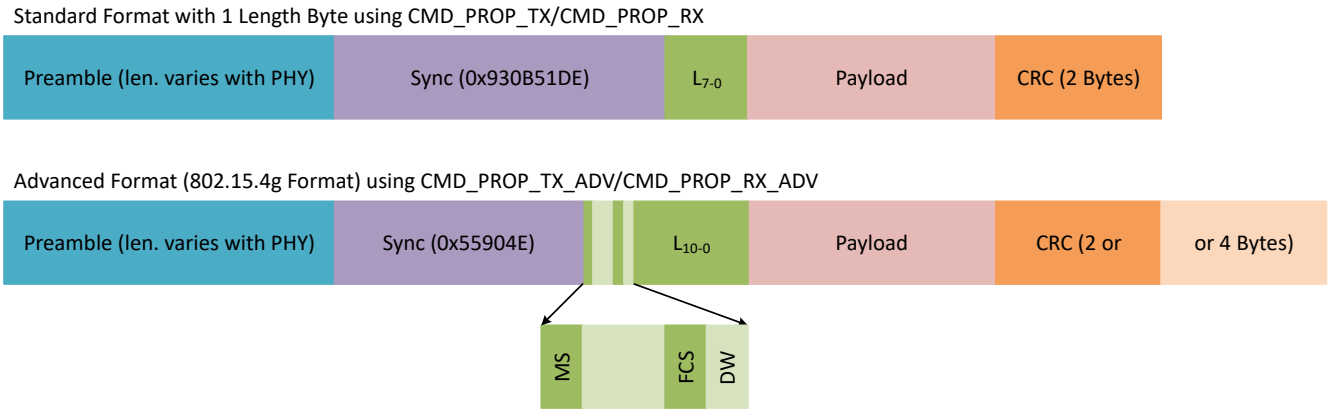

Figure 1-1. Standard vs. Advanced Packet Format

The standard packet format can also be implemented using the advanced commands and can easily be changed to use two length bytes instead of one, and the PHYs that by default use the advanced packet format, can also be modified to transmit and receive packets of the standard packet format (with both 1 and 2 length bytes). Table 1-1 summarizes the different combinations this app note discusses.

Table 1-1. Combinations of Command Types and Packet Format

Command Type Imported/ Exported	Packet Format	Command Type Used	Mode	Max Payload Length	Section
Standard CMD_PROP_TX CMD_PROP_RX	Standard (1 Length Byte)	Standard	TX	255	2.1.1
		Standard	RX	255	2.1.2
		Advanced	TX	255	2.1.3
		Advanced	RX	255	2.1.4
	Standard (2 Length Bytes)	Advanced	TX	4093	2.2.1
		Advanced	RX	4093	2.2.2
Advanced CMD_PROP_TX_ADV CMD_PROP_RX_ADV	Advanced (802.15.4g)	Advanced	TX	2043 or 2045[1]	3.1.1
		Advanced	RX	2047[2]	3.1.2
	Standard (1 Length Byte)	Advanced	TX	255	3.2.1
		Advanced	RX	255	3.2.2
	Standard (2 Length Bytes)	Advanced	TX	4093	3.3.1
		Advanced	RX	4093	3.3.2

[1] Max payload length depends on FSC type (2 or 4 bytes CRC)
[2] Max length on the RX sides includes number of CRC bytes (2 or 4)

2 PHY Settings Exported with the Standard Commands

When exporting/importing PHY settings that use the standard TX and RX commands, the commands natively support the standard packet format (with 1 length byte) and no modifications are needed to the rfPacketTx and rfPacketRx examples to run with these PHYs. The following sections (2.1 - 2.2) show how to implement the standard packet format (1 length byte) with both command types (standard and advanced) and also how the standard packet format can be modified to have two length bytes instead of one.

2.1 Standard Packet Format (1 Length Byte)

Assume that you want to transmit a payload of 3 bytes (0x01, 0x02, 0x03).

Over the air, the packet will look like what is shown in Figure 2-1.

Figure 2-1. Standard Packet Format (1 Length Byte)

2.1.1 TX using CMD_PROP_TX and Standard Packet Format (1 Length Byte)

The rfPacketTx example is written to support this format, and no changes are needed to the code examples, other than to the payload length and content. However, in the following code snippet, some modifications have been implemented to improve readability. The following code shows how rfPacketTx.c can be modified to send the wanted payload once every 0.5 s.

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format with CMD_PROP_TX (1 Length Byte)
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH  3 // Max 255 bytes
7:
8:  uint8_t packet[PAYLOAD_LENGTH];
9:
10: static RF_Object rfObject;
11: static RF_Handle rfHandle;
12:
13: void *mainThread(void *arg0)
14: {
15:     RF_Params rfParams;
16:     RF_Params_init(&rfParams);
17:
18:     RF_cmdPropTx.pktLen = PAYLOAD_LENGTH; // Application specific settings
19:     RF_cmdPropTx.pPkt   = packet;
20:
21:     rfHandle = RF_open(&rfObject, &RF_prop,
22:                        (RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);
23:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
24:
25:     while(1)
26:     {
27:         //-----------------------------------------------------------------------------------
28:         // Could be placed outside the while(1) since the packet does not change
29:         for (uint8_t i = 0; i < PAYLOAD_LENGTH; i++)
30:         {
31:             packet[i] = i + 1;
32:         }
33:         //-----------------------------------------------------------------------------------
34:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTx, RF_PriorityNormal, NULL, 0);
35:         RF_yield(rfHandle);
36:         usleep(500000);
37:     }
38: }
```

When using the standard TX command, the maximum payload length is 255, and the length byte (PAYLOAD_LENGTH) must be written to the .pktLen field of the command.

2.1.2 RX using CMD_PROP_RX and Standard Packet Format (1 Length Byte)

To be able to receive a packet with the standard packet format shown in Figure 1-1 using CMD_PROP_RX, the following code could be used (also here the code has been slightly modified compared to the original rfPacketRx, to increase readability and enable easier debugging).

```
1:  //------------------------------------------------------------------------------------------
2:  // Receive Standard Packet Format with CMD_PROP_RX (1 Length Byte)
3:  //------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define DATA_ENTRY_HEADER_SIZE  8   // Constant header size of a Generic Data Entry
7:  #define NUM_DATA_ENTRIES        2   // NOTE: Only two data entries supported
8:  #define CRC                     2   // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
9:  #define RSSI                    1   // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
10: #define TIMESTAMP               4   // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
11: #define STATUS                  1   // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
12: #define LENGTH_FIELD            1   // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
13: #define MAX_LENGTH              255 // Max length the radio will accept
14: #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
15:
16: uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
17: uint8_t packetLength;                                          // packetLength
18:
19: static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
20:
21: static RF_Object rfObject;
22: static RF_Handle rfHandle;
23:
24: static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
25:                                   NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
26: static dataQueue_t dataQueue;
27: static rfc_dataEntryGeneral_t* currentDataEntry;
28: static uint8_t* packetDataPointer;
29: rfc_propRxOutput_t rxStatistics;
30: uint16_t crc16;
31: int8_t rssi;
32: uint32_t timestamp;
33: uint8_t status;
34:
35: void *mainThread(void *arg0)
36: {
37:     RF_Params rfParams;
38:     RF_Params_init(&rfParams);
39:
40:     if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
41:                            NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
42:     {
43:         while(1);
44:     }
45:
46:     RF_cmdPropRx.pktConf.bRepeatOk = 0x1; // Application specific settings
47:     RF_cmdPropRx.pktConf.bRepeatNok = 0x1;
48:     RF_cmdPropRx.rxConf.bAutoFlushIgnored = 0x1;
49:     RF_cmdPropRx.rxConf.bAutoFlushCrcErr = 0x1;
50:     RF_cmdPropRx.maxPktLen = MAX_LENGTH;
51:     RF_cmdPropRx.pQueue = &dataQueue;
52:
53:     RF_cmdPropRx.rxConf.bIncludeCrc = 0x1; // Optional bytes to append
54:     RF_cmdPropRx.rxConf.bAppendRssi = 0x1;
55:     RF_cmdPropRx.rxConf.bAppendTimestamp = 0x1;
56:     RF_cmdPropRx.rxConf.bAppendStatus = 0x1;
57:
58:     RF_cmdPropRx.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)
59:
60:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
61:                        &rfParams);
62:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
63:     RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRx, RF_PriorityNormal,
64:               &callback, RF_EventRxEntryDone);
65:     while(1);
66: }
67:
```

```
68: //---------------------------------------------------------------------------------------------
69: // Callback for Receiving Standard Packet Format with CMD_PROP_RX (1 Length Byte)
70: //---------------------------------------------------------------------------------------------
71: void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
72: {
73:     if(e & RF_EventRxEntryDone)
74:     {
75:         currentDataEntry = RFQueue_getDataEntry();
76:
77:         packetLength = *(uint8_t*)(&currentDataEntry->data);
78:         packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
79:
80:         memcpy(packet, packetDataPointer, (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
81:
82:         crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
83:                  (uint16_t)(packet[packetLength + 1] << 0));
84:
85:         rssi = packet[packetLength + 2];
86:
87:         timestamp = ((uint32_t)(packet[packetLength + 3] <<  0) +
88:                      (uint32_t)(packet[packetLength + 4] <<  8) +
89:                      (uint32_t)(packet[packetLength + 5] << 16) +
90:                      (uint32_t)(packet[packetLength + 6] << 24));
91:
92:         status = packet[packetLength + 7];
93:
94:         RFQueue_nextEntry();
95:     }
96: }
```

In this code example, no length filtering is implemented, and all optional bytes that can be appended, are appended. If the only packet to be received is the packet shown in Figure 2-1, MAX_LENGTH can be changed from 255 to 3, causing all packets with length byte larger than 3 to be discarded (since rxConf.bAutoFlushIgnored = 1).

2.1.3 TX using CMD_PROP_TX_ADV and Standard Packet Format (1 Length Byte)

In some cases, the user would want to use the advanced command to transmit a packet, even if the packet format is the standard one. This could be because they eventually want to transmit a longer preamble than what is supported by the standard TX command, for example in a Sniff Mode application.

Both SysConfig (3) and SmartRF Studio (2) support importing and exporting other commands than what is used for a PHY by default. Figure 2-2 shows how to select extra commands to be imported to a project using SysConfig, and Figure 2-3 shows how this is done with code exports using SmartRF Studio.

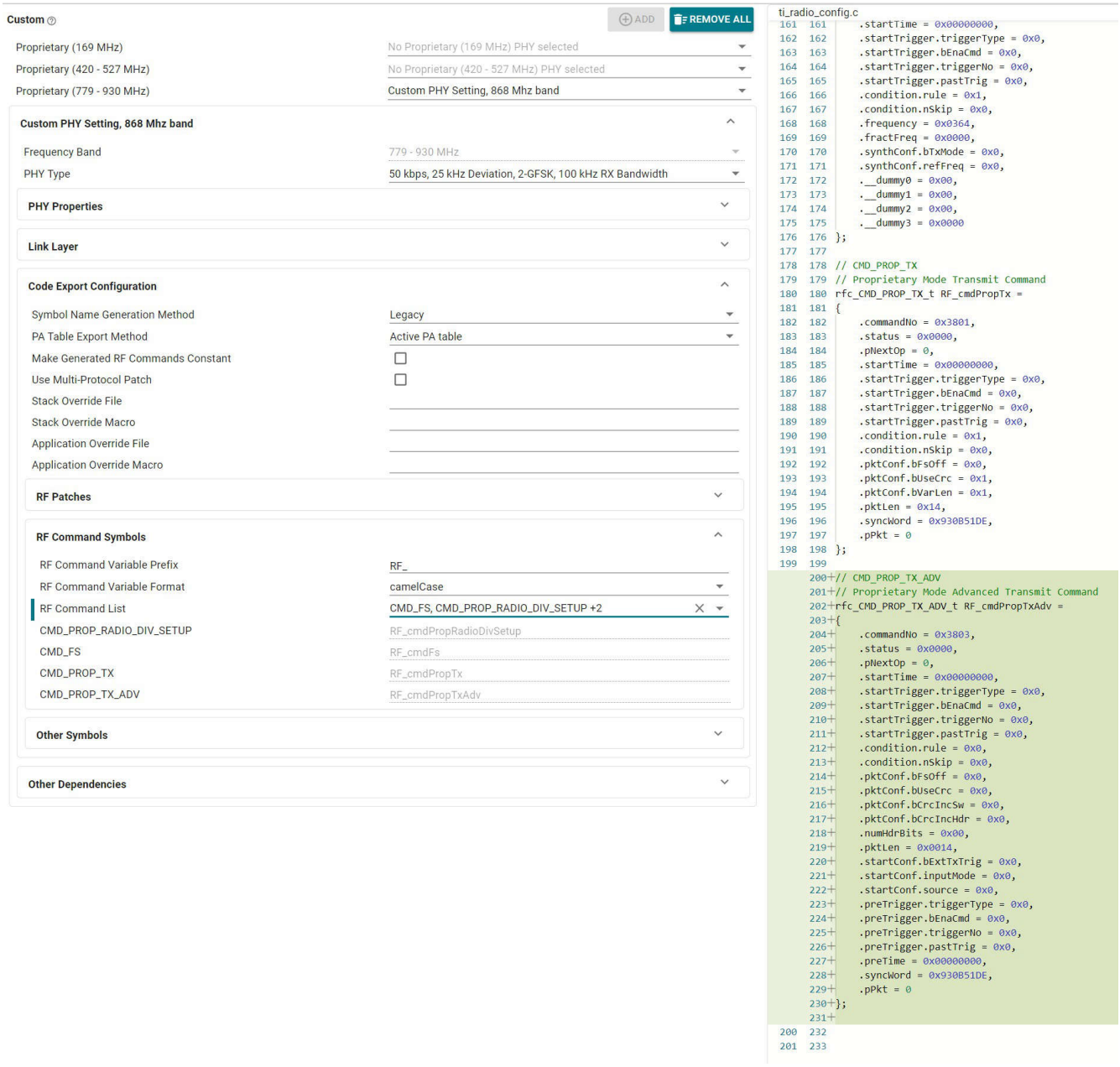

Figure 2-2. Importing the Advanced TX Command in addition to the Standard Command in SysConfig

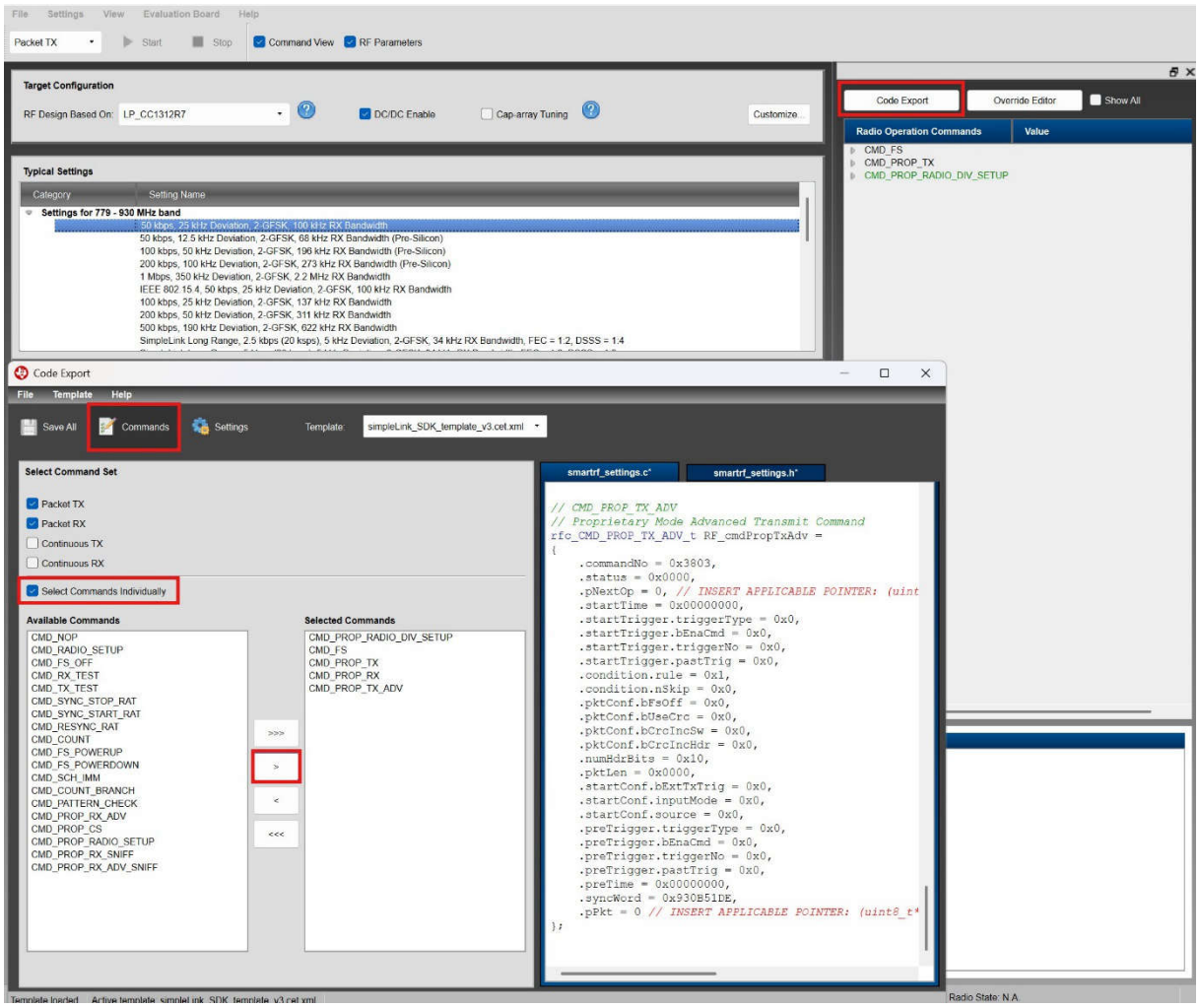

Figure 2-3. Exporting the Advanced TX Command in addition to the Standard Command in SmartRF Studio

SmartRF Studio and SysConfig are not giving the exact same settings when adding an extra command like this, so the code example is written to cover both cases. When using the advanced TX command, the length byte must be manually written to the packet together with the payload, and the .pktLen field of the command must include both the length field and the payload length. A code example is shown below.

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format with CMD_PROP_TX_ADV (1 Length Byte)
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 255 bytes
7:  #define LENGTH_FIELD   1
8:
9:  uint8_t packet[LENGTH_FIELD + PAYLOAD_LENGTH];
10:
11: static RF_Object rfObject;
12: static RF_Handle rfHandle;
13:
14: void *mainThread(void *arg0)
15: {
16:     RF_Params rfParams;
17:     RF_Params_init(&rfParams);
18:
19:     RF_cmdPropTxAdv.numHdrBits = 0x0; // Settings that must change to support
20:                                       // the standard packet format
21:
22:     RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH; // Application specific settings
23:     RF_cmdPropTxAdv.pPkt = packet;
24:
25:     // Settings to modify if going from a PHY that uses the standard TX command
26:     // to use the advanced TX command
27:     RF_cmdPropTxAdv.condition.rule = 0x1;
28:     RF_cmdPropTxAdv.pktConf.bUseCrc = 0x1;
29:
30:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
31:                        &rfParams);
31:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
32:
33:     while(1)
34:     {
35:         //-----------------------------------------------------------------------------------
36:         // Could be placed outside the while(1) since the packet does not change
37:         packet[0] = PAYLOAD_LENGTH;
38:
39:         for (uint16_t i = 1; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
40:         {
41:             packet[i] = i;
42:         }
43:         //-----------------------------------------------------------------------------------
44:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
45:         RF_yield(rfHandle);
46:         usleep(500000);
47:     }
48: }
```

2.1.4 RX using CMD_PROP_RX_ADV and Standard Packet Format (1 Length Byte)

If we want to receive a packet with standard packet format using the advanced RX command instead of the standard RX command, this is also possible, and the command can be added in the same way as described for the advanced TX command in Section 2.1.3. In the following, a code example is showning how a standard packet can be received using the advanced RX command. Also here, MAX_LENGTH can be changed from 255 to 3 to enable packet length filtering.

```
1:   //---------------------------------------------------------------------------------------
2:   // Receive Standard Packet Format with CMD_PROP_RX_ADV (1 Length Byte)
3:   //---------------------------------------------------------------------------------------
4:   // Defines
5:   #define DATA_ENTRY_HEADER_SIZE  8   // Constant header size of a Generic Data Entry
6:   #define NUM_DATA_ENTRIES        2   // NOTE: Only two data entries supported
7:   #define CRC                     2   // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
8:   #define RSSI                    1   // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
9:   #define TIMESTAMP               4   // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
10:  #define STATUS                  1   // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
11:  #define LENGTH_FIELD            1   // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
12:  #define MAX_LENGTH              255 // Max length the radio will accept
13:  #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
14:
15:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
16:  uint8_t packetLength;                                          // packetLength
17:  static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
18:
19:  static RF_Object rfObject;
20:  static RF_Handle rfHandle;
21:
22:  static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
23:                                   NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
24:  static dataQueue_t dataQueue;
25:  static rfc_dataEntryGeneral_t* currentDataEntry;
26:  static uint8_t* packetDataPointer;
27:  rfc_propRxOutput_t rxStatistics;
28:  uint16_t crc16;
29:  int8_t rssi;
30:  uint32_t timestamp;
31:  uint8_t status;
32:
33:  void *mainThread(void *arg0)
34:  {
35:      RF_Params rfParams;
36:      RF_Params_init(&rfParams);
37:
38:      if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
39:                             NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
40:      {
41:          while(1);
42:      }
43:
44:      RF_cmdPropRxAdv.pktConf.bCrcIncHdr = 0x1; // Settings that must change to support
45:      RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x8; // the standard packet format
46:      RF_cmdPropRxAdv.hdrConf.numLenBits = 0x8;
47:
48:      RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1; // Application specific settings
49:      RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
50:      RF_cmdPropRxAdv.rxConf.bAutoFlushIgnored = 0x1;
51:      RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
52:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
53:      RF_cmdPropRxAdv.pQueue = &dataQueue;
54:
55:      // Settings to modify if going from a PHY that uses the standard RX command
56:      // to use the advanced RX command
57:      RF_cmdPropRxAdv.condition.rule = 0x1;
58:      RF_cmdPropRxAdv.pktConf.bUseCrc = 0x1;
59:      RF_cmdPropRxAdv.rxConf.bIncludeHdr = 0x1;
60:      RF_cmdPropRxAdv.endTrigger.triggerType = 0x1;
61:
62:      RF_cmdPropRxAdv.rxConf.bIncludeCrc = 0x1; // Optional bytes to append
63:      RF_cmdPropRxAdv.rxConf.bAppendRssi = 0x1;
64:      RF_cmdPropRxAdv.rxConf.bAppendTimestamp = 0x1;
65:      RF_cmdPropRxAdv.rxConf.bAppendStatus = 0x1;
66:
67:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)
68:
69:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
70:                         &rfParams);
71:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
72:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
73:                &callback, RF_EventRxEntryDone);
74:      while(1);
75:  }
76:
```

```
77:  //-------------------------------------------------------------------------------------------
78:  // Callback for Receiving Standard Packet Format with CMD_PROP_RX_ADV (1 Length Byte)
79:  //-------------------------------------------------------------------------------------------
80:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
81:  {
82:      if(e & RF_EventRxEntryDone)
83:      {
84:          currentDataEntry = RFQueue_getDataEntry();
85:
86:         packetLength = *(uint8_t*)(&currentDataEntry->data);
87:         packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
88:
89:         memcpy(packet, packetDataPointer, (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
90:
91:         crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
92:                  (uint16_t)(packet[packetLength + 1] << 0));
93:
94:         rssi = packet[packetLength + 2];
95:
96:         timestamp = ((uint32_t)(packet[packetLength + 3] << 0 ) +
97:                      (uint32_t)(packet[packetLength + 4] << 8 ) +
98:                      (uint32_t)(packet[packetLength + 5] << 16) +
99:                      (uint32_t)(packet[packetLength + 6] << 24));
100:
101:        status = packet[packetLength + 7];
102:
103:        RFQueue_nextEntry();
104:    }
105: }
```

2.2 Standard Packet Format (2 Length Bytes)

Another reason for wanting to change from the standard commands to the advanced ones could be that one wants to transmit and receive packets that are longer than 255 bytes and hence needs more than one byte for the length field.

With a two bytes long length field, the packet used for the previous examples, would look like shown in Figure 2-4.

Figure 2-4. Standard Packet Format (2 Length Bytes)

2.2.1 TX using CMD_PROP_TX_ADV and Standard Packet Format (2 Length Bytes)

There are no changes needed to the settings compared to what is shown in the example in Section 2.1.3.

The only changes needed are the following:

- Alter LENGTH_FIELD from 1 to 2
- Add both length bytes to the packet

Necessary changes are found on line 7, and line 38-48.

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format with CMD_PROP_TX_ADV (2 Length Bytes)
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 4093 bytes
7:  #define LENGTH_FIELD   2 // Changed from 1
8:
9:  uint8_t packet[LENGTH_FIELD + PAYLOAD_LENGTH];
10:
11: static RF_Object rfObject;
12: static RF_Handle rfHandle;
13:
14: void *mainThread(void *arg0)
15: {
16:     RF_Params rfParams;
17:     RF_Params_init(&rfParams);
18:
19:     RF_cmdPropTxAdv.numHdrBits = 0x0; // Settings that must change to support
20:                                       // the standard packet format
21:
22:     RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH; // Application specific
23:     RF_cmdPropTxAdv.pPkt = packet;                          // settings
24:
25:     // Settings to modify if going from a PHY that uses the standard TX command
26:     // to use the advanced TX command
27:     RF_cmdPropTxAdv.condition.rule = 0x1;
28:     RF_cmdPropTxAdv.pktConf.bUseCrc = 0x1;
29:
30:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
31:                        &rfParams);
32:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
33:
34:     while(1)
35:     {
36:         //-----------------------------------------------------------------------------------
37:         // Could be placed outside the while(1) since the packet does not change
38: #ifdef SLR_MODE
39:         packet[0] = (uint8_t)(PAYLOAD_LENGTH);
40:         packet[1] = (uint8_t)(PAYLOAD_LENGTH >> 8);
41: #else
42:         packet[0] = (uint8_t)(PAYLOAD_LENGTH >> 8);
43:         packet[1] = (uint8_t)(PAYLOAD_LENGTH);
44: #endif
45:         for (uint16_t i = 2; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
46:         {
47:             packet[i] = i - 1;
48:         }
49:         //-----------------------------------------------------------------------------------
50:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
51:         RF_yield(rfHandle);
52:         usleep(500000);
53:     }
54: }
```

Please note that if you are using one of the SimpleLink Long Range (SLR) PHYs or the Legacy Long Range PHY (CC13x0 only), the two length bytes need to be written in the opposite order compared to when using any of the other proprietary PHYs.

2.2.2 RX Using CMD_PROP_RX_ADV and Standard Packet Format (2 Length Bytes)

To receive the standard packet format with 2 length bytes, changes are needed to the settings compared to what is shown in the example in Section 2.1.4. Also, some defines and variables need to be changed, together with the application code in the callback. Necessary changes are found on line 11 and 12, line 15 and 16, line 46 and 47, and line 86-89.

```
1:   //---------------------------------------------------------------------------------------
2:   // Receive Standard Packet Format with CMD_PROP_RX_ADV (2 Length Bytes)
3:   //---------------------------------------------------------------------------------------
4:   // Defines
5:   #define DATA_ENTRY_HEADER_SIZE  8    // Constant header size of a Generic Data Entry
6:   #define NUM_DATA_ENTRIES        2    // NOTE: Only two data entries supported
7:   #define CRC                     2    // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
8:   #define RSSI                    1    // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
9:   #define TIMESTAMP               4    // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
10:  #define STATUS                  1    // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
11:  #define LENGTH_FIELD            2    // Changed from 1
12:  #define MAX_LENGTH              4093 // Changed from 255
13:  #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
14:
15:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
16:  uint16_t packetLength; // Changed from uint8_t                    packetLength
17:
18:  static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
19:  static RF_Object rfObject;
20:  static RF_Handle rfHandle;
21:
22:  static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
23:                                   NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
24:  static dataQueue_t dataQueue;
25:  static rfc_dataEntryGeneral_t* currentDataEntry;
26:  static uint8_t* packetDataPointer;
27:  rfc_propRxOutput_t rxStatistics;
28:  uint16_t crc16;
29:  int8_t rssi;
30:  uint32_t timestamp;
31:  uint8_t status;
32:
33:  void *mainThread(void *arg0)
34:  {
35:      RF_Params rfParams;
36:      RF_Params_init(&rfParams);
37:
38:      if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
39:                             NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
40:      {
41:          while(1);
42:      }
43:
44:      RF_cmdPropRxAdv.pktConf.bCrcIncHdr = 0x1;  // Settings that must change
45:                                                 // to support the standard format
46:      RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x10; // Changed from 0x8
47:      RF_cmdPropRxAdv.hdrConf.numLenBits = 0x10; // Changed from 0x8
48:
49:      RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1;   // Application specific settings
50:      RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
51:      RF_cmdPropRxAdv.rxConf.bAutoFlushIgnored = 0x1;
52:      RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
53:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
54:      RF_cmdPropRxAdv.pQueue = &dataQueue;
55:
56:      // Settings to modify if going from a PHY that uses the standard RX command
57:     // to use the advanced RX command
58:      RF_cmdPropRxAdv.condition.rule = 0x1;
59:      RF_cmdPropRxAdv.pktConf.bUseCrc = 0x1;
60:      RF_cmdPropRxAdv.rxConf.bIncludeHdr = 0x1;
61:      RF_cmdPropRxAdv.endTrigger.triggerType = 0x1;
62:
63:      RF_cmdPropRxAdv.rxConf.bIncludeCrc = 0x1; // Optional bytes to append
64:      RF_cmdPropRxAdv.rxConf.bAppendRssi = 0x1;
65:      RF_cmdPropRxAdv.rxConf.bAppendTimestamp = 0x1;
66:      RF_cmdPropRxAdv.rxConf.bAppendStatus = 0x1;
67:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)
68:
69:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
70:                         &rfParams);
71:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
72:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
73:                &callback, RF_EventRxEntryDone);
74:      while(1);
75:  }
76:
```

```
77:  //---------------------------------------------------------------------------------------
79:  // Callback for Receiving Standard Packet Format with CMD_PROP_RX_ADV (2 Length Bytes)
79:  //---------------------------------------------------------------------------------------
80:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
81:  {
82:      if(e & RF_EventRxEntryDone)
83:      {
84:          currentDataEntry = RFQueue_getDataEntry();
85:
86:          packetLength = ((uint16_t)(*(uint8_t*)(&currentDataEntry->data + 1) << 8) |
87:                          (uint16_t)(*(uint8_t*)(&currentDataEntry->data + 0) << 0));
88:          // Changed from
89:          // packetLength = *(uint8_t*)(&currentDataEntry->data);
90:
91:          packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
92:
93:          memcpy(packet, packetDataPointer,
94:                 (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
95:
96:          crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
97:                   (uint16_t)(packet[packetLength + 1] << 0));
98:
99:          rssi = packet[packetLength + 2];
100:
101:         timestamp = ((uint32_t)(packet[packetLength + 3] << 0 ) +
102:                      (uint32_t)(packet[packetLength + 4] << 8 ) +
103:                      (uint32_t)(packet[packetLength + 5] << 16) +
104:                      (uint32_t)(packet[packetLength + 6] << 24));
105:
106:         status = packet[packetLength + 7];
107:
108:         RFQueue_nextEntry();
109:     }
110: }
```

3 TX and RX Settings Exported with the Advanced Commands

All PHYs that are, by default, exported/imported with the CMD_PROP_TX_ADV or CMD_PROP_RX_ADV commands use a packet format specified by IEEE 802.15.4g. This format is shown in Figure 3-1.

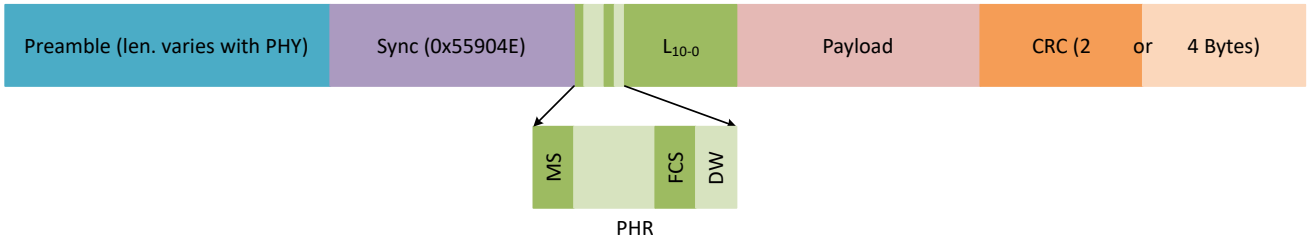

Figure 3-1. Advanced Packet Format

The sync word (0x55904E) is followed by a 2 bytes long header (PHR) containing the following fields:

PHR[15] MS: Mode Switch (always set to 0)

PHR[14:13] Reserved (Don't care)

PHR[12] FCS: FCS Type (0: 4 bytes CRC, 1: 2 bytes CRC)

PHR[11] DW: Data whitening (0: Disable, 1: Enable)

PHR[10:0] Length

The length field should include the CRC bytes (2 or 4 bytes depending on the FCS field in the PHR), meaning that if you want to transmit the 3 bytes long payload used in the previous example, the length in the header should be 5 or 7. If you for example want to use a 4 bytes long CRC (FCS = 0) plus whitening (DW = 1), the header would be 0x0807.

3.1 Advanced Packet Format

The following two sections show how the rfPacketTx and rfPacketRX examples can be modified to transmit and receive a packet using the advanced packet format. In the TX example, we assume that we want to use a 4 bytes long CRC plus whitening.

3.1.1 TX using CMD_PROP_TX_ADV and Advanced Packet Format

The header is written to the packet with least significant byte first, so the packet that the CMD_PROP_TX_ADV command should transmit, would be 0x07, 0x08, 0x01, 0x02, 0x03.

Since the length field is 11 bits the max number of payload bytes are 2045 when using a 2 byte long CRC, and 2043 when the CRC is 4 bytes long. The following shows code that can be used to transmit a packet using the advanced packet format.

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Advanced Packet Format with CMD_PROP_TX_ADV
3:  //---------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 2045 or 2043, depending on FSC type
7:  #define HEADER_FIELD   2
8:
9:  // #define _802_15_4_G_HEADER   0x00 // MS = 0, FCS = 0 (4 bytes CRC), DW = 0 (Whitening Off)
10: #define _802_15_4_G_HEADER      0x08 // MS = 0, FCS = 0 (4 bytes CRC), DW = 1 (Whitening On)
11: // #define _802_15_4_G_HEADER   0x10 // MS = 0, FCS = 1 (2 bytes CRC), DW = 0 (Whitening Off)
12: // #define _802_15_4_G_HEADER   0x18 // MS = 0, FCS = 1 (2 bytes CRC), DW = 1 (Whitening On)
13:
14: uint8_t packet[HEADER_FIELD + PAYLOAD_LENGTH];
15: uint16_t header;
16:
17: static RF_Object rfObject;
18: static RF_Handle rfHandle;
19:
20: void *mainThread(void *arg0)
21: {
22:     RF_Params rfParams;
23:     RF_Params_init(&rfParams);
24:
25:     RF_cmdPropTxAdv.startTrigger.triggerType = 0x0; // Application specific settings
26:     RF_cmdPropTxAdv.startTrigger.pastTrig = 0x0;
27:     RF_cmdPropTxAdv.pktLen = HEADER_FIELD + PAYLOAD_LENGTH;
28:     RF_cmdPropTxAdv.preTrigger.triggerType = 0x0;
29:     RF_cmdPropTxAdv.preTrigger.pastTrig = 0x0;
30:     RF_cmdPropTxAdv.pPkt = packet;
31:
32:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
33:                        &rfParams);
34:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
35:
36:     while(1)
37:     {
38:         //-----------------------------------------------------------------------------------
39:         // Could be placed outside the while(1) since the packet does not change
40:         if((_802_15_4_G_HEADER == 0x00) || (_802_15_4_G_HEADER == 0x08))
41:         {
42:             header = (_802_15_4_G_HEADER << 8) + ((4 + PAYLOAD_LENGTH) & 0x07FF);
43:         }
44:         else
45:         {
46:             header = (_802_15_4_G_HEADER << 8) + ((2 + PAYLOAD_LENGTH) & 0x07FF);
47:         }
48:         packet[0] = (uint8_t)(header);
49:         packet[1] = (uint8_t)(header >> 8);
50:
51:         for (uint16_t i = 2; i < (HEADER_FIELD + PAYLOAD_LENGTH); i++)
52:         {
53:             packet[i] = i - 1;
54:         }
55:         //-----------------------------------------------------------------------------------
56:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
57:         RF_yield(rfHandle);
58:         usleep(500000);
59:     }
60: }
```

3.1.2 RX using CMD_PROP_RX_ADV and Advanced Packet Format

Below is code that can be used to receive a packet using the advanced packet format. This code can only be used for PHY settings that export/import the advanced commands by default.

```
//---------------------------------------------------------------------------------------------
// Receive Advanced Packet Format with CMD_PROP_RX_ADV
//---------------------------------------------------------------------------------------------

// Defines
#define DATA_ENTRY_HEADER_SIZE  8 // Constant header size of a Generic Data Entry
#define NUM_DATA_ENTRIES        2 // NOTE: Only two data entries supported
#define CRC                     4 // 4/2 (based on FCS) if .rxConf.bIncludeCrc = 0x1, else 0
#define RSSI                    1 // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
#define TIMESTAMP               4 // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
#define STATUS                  1 // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
#define HEADER_FIELD            2 // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
#define MAX_LENGTH              2047 // Max length the radio will accept (incl. CRC bytes)
#define NUM_APPENDED_BYTES      HEADER_FIELD + RSSI + TIMESTAMP + STATUS

uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - HEADER_FIELD]; // Header/Length stored in
uint16_t packetLength;                                          // packetLength variable

static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);

static RF_Object rfObject;
static RF_Handle rfHandle;

static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
                                 MAX_LENGTH, NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
static dataQueue_t dataQueue;
static rfc_dataEntryGeneral_t* currentDataEntry;
static uint8_t* packetDataPointer;
rfc_propRxOutput_t rxStatistics;
uint16_t crc16;
uint32_t crc32;
int8_t rssi;
uint32_t timestamp;
uint8_t status;

void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
                           NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
    {
        while(1);
    }

    RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1; // Application specific settings
    RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
    RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
    RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
    RF_cmdPropRxAdv.pQueue = &dataQueue;

    RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)

    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
                       &rfParams);
    RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
    RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
              &callback, RF_EventRxEntryDone);
    while(1);
}

```

```
63:  //-------------------------------------------------------------------------------------------
64:  // Callback for Receiving Advanced Packet Format with CMD_PROP_RX_ADV
65:  //-------------------------------------------------------------------------------------------
66:
67:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
68:  {
69:      if(e & RF_EventRxEntryDone)
70:      {
71:          currentDataEntry = RFQueue_getDataEntry();
72:
73:          uint16_t header = ((uint16_t)(*(uint8_t*)(&currentDataEntry->data + 1) << 8) |
74:                             (uint16_t)(*(uint8_t*)(&currentDataEntry->data + 0) << 0));
75:
76:          bool fcs = (bool)(0x1000 & header);
77:
78:          packetLength = 0x07FF & header;
79:
80:          packetDataPointer = (uint8_t*)(&currentDataEntry->data + HEADER_FIELD);
81:
82:          memcpy(packet, packetDataPointer,
83:                 (packetLength + NUM_APPENDED_BYTES - HEADER_FIELD));
84:
85:          // The FCS field in the header tell us if the CRC is 2 or 4 bytes long
86:          if(fcs) // FCS = 1 -> 2 bytes CRC
87:          {
88:              crc16 = ((uint16_t)(packet[packetLength - 2] <<  8) +
89:              (uint16_t)(packet[packetLength - 1] <<  0));
90:          }
91:          else // FCS = 0 -> 4 bytes CRC
92:          {
93:              crc32 = ((uint32_t)(packet[packetLength - 4] << 24) +
94:                       (uint32_t)(packet[packetLength - 3] << 16) +
95:                       (uint32_t)(packet[packetLength - 2] <<  8) +
96:                       (uint32_t)(packet[packetLength - 1] <<  0));
97:          }
98:
99:          rssi = packet[packetLength + 0];
100:
101:         timestamp = ((uint32_t)(packet[packetLength + 1] << 0 ) +
102:                      (uint32_t)(packet[packetLength + 2] << 8 ) +
103:                      (uint32_t)(packet[packetLength + 3] << 16) +
104:                      (uint32_t)(packet[packetLength + 4] << 24));
105:
106:         status = packet[packetLength + 5];
107:
108:         RFQueue_nextEntry();
109:     }
110: }
```

The maximum packet length supported for this format is 2047 bytes (this includes CRC (2 or 4 bytes)).

Please note that IEEE 802.15.4g does not specify the format of the payload (PSDU data). The PSDU is only described as a stream of bits, and whatever is inside is decided by the higher-level MAC specification.

In the code examples in Section 3.1.1 and Section 3.1.2, the payload is transmitted MSB first. To be compliant with SmartRF Studio (transmitting the payload LSB first by default), the code below can be used (on both RX and TX).

```
1:  //-------------------------------------------------------------------------------------------
2:  // Code for Converting Payload from MSB First to LSB First
3:  //-------------------------------------------------------------------------------------------
4:
5:  uint8_t lsbFirst(uint8_t b)
6:  {
7:      b = (b & 0xF0) >> 4 | (b & 0x0F) << 4;
8:      b = (b & 0xCC) >> 2 | (b & 0x33) << 2;
9:      b = (b & 0xAA) >> 1 | (b & 0x55) << 1;
10:     return b;
11: }
```

3.2 Standard Packet Format (1 Length Byte)

A user might want to use the standard packet format, even if SmartRF Studio and SysConfig exports/imports settings for the advanced packet format. Section 3.2.1 and 3.2.2 will show TX and RX code for how to support this with 1 length byte.

Please note that for both RX and TX it is necessary to also make changes to the setup command (CMS_PROP_RADIO_DIV_SETUP). The command needs to be configured for a 32 bites long sync word, and whitening must be turned off (line 29-30 (TX example) and line 60-61 (RX example)).

3.2.1 TX using CMD_PROP_TX_ADV and Standard Packet Format (1 Length Byte)

```
1:  //-------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format (1 Length Byte) with PHYs using CMD_PROP_TX_ADV by Default
3:  //-------------------------------------------------------------------------------------------
4:
5:  // Defines
6:  #define PAYLOAD_LENGTH 3 // Max 255 bytes
7:  #define LENGTH_FIELD   1
8:
9:  uint8_t packet[LENGTH_FIELD + PAYLOAD_LENGTH];
10:
11: static RF_Object rfObject;
12: static RF_Handle rfHandle;
13:
14: void *mainThread(void *arg0)
15: {
16:     RF_Params rfParams;
17:     RF_Params_init(&rfParams);
18:
19:     RF_cmdPropTxAdv.startTrigger.triggerType = 0x0; // Application specific settings
20:     RF_cmdPropTxAdv.startTrigger.pastTrig = 0x0;
21:     RF_cmdPropTxAdv.numHdrBits = 0x0;
22:     RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH;
23:     RF_cmdPropTxAdv.preTrigger.triggerType = 0x0;
24:     RF_cmdPropTxAdv.preTrigger.pastTrig = 0x0;
25:     RF_cmdPropTxAdv.syncWord = 0x930B51DE;
26:     RF_cmdPropTxAdv.pPkt = packet;
27:
28:     // Necessary changes to the Setup command to support the standard packet format
29:     RF_cmdPropRadioDivSetup.formatConf.nSwBits = 0x20; // 32 bits sync word
30:     RF_cmdPropRadioDivSetup.formatConf.whitenMode = 0x0; // No whitening
31:
32:     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
33:                        &rfParams);
34:     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
35:
36:     while(1)
37:     {
38:         //-----------------------------------------------------------------------------------
39:         // Could be placed outside the while(1) since the packet does not change
40:         packet[0] = PAYLOAD_LENGTH;
41:
42:         for (uint16_t i = 1; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
43:         {
44:             packet[i] = i;
45:         }
46:         //-----------------------------------------------------------------------------------
47:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
48:         RF_yield(rfHandle);
49:         usleep(500000);
50:     }
51: }
```

3.2.2 RX using CMD_PROP_RX_ADV and Standard Packet Format (1 Length Byte)

```
//---------------------------------------------------------------------------------------------
// Receive Standard Packet Format (1 Length Byte) with PHYs using CMD_PROP_RX_ADV by Default
//---------------------------------------------------------------------------------------------

// Defines
#define DATA_ENTRY_HEADER_SIZE  8   // Constant header size of a Generic Data Entry
#define NUM_DATA_ENTRIES        2   // NOTE: Only two data entries supported
#define CRC                     2   // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
#define RSSI                    1   // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
#define TIMESTAMP               4   // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
#define STATUS                  1   // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
#define LENGTH_FIELD            1   // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
#define MAX_LENGTH              255 // Max length the radio will accept
#define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS

uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
uint8_t packetLength;                                           // packetLength

static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);

static RF_Object rfObject;
static RF_Handle rfHandle;

static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
                                 NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
static dataQueue_t dataQueue;
static rfc_dataEntryGeneral_t* currentDataEntry;
static uint8_t* packetDataPointer;
rfc_propRxOutput_t rxStatistics;
uint16_t crc16;
int8_t rssi;
uint32_t timestamp;
uint8_t status;

void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
                           NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
    {
        while(1);
    }

    RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1; // Application specific settings
    RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
    RF_cmdPropRxAdv.pktConf.bCrcIncHdr = 0x1;
    RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
    RF_cmdPropRxAdv.syncWord0 = 0x930B51DE;
    RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
    RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x8;
    RF_cmdPropRxAdv.hdrConf.numLenBits = 0x8;
    RF_cmdPropRxAdv.lenOffset = 0x0;
    RF_cmdPropRxAdv.pQueue = &dataQueue;

    RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Opt. (for debug)

    // Necessary changes to the Setup command to support the standard packet format
    RF_cmdPropRadioDivSetup.formatConf.nSwBits = 0x20; // 32 bits sync word
    RF_cmdPropRadioDivSetup.formatConf.whitenMode = 0x0; // No whitening

    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
                       &rfParams);
    RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
    RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
              &callback, RF_EventRxEntryDone);
    while(1);
}

```

```
71:  //-------------------------------------------------------------------------------------------
72:  // Callback for Receiving Standard Packet Format (1 Length Byte) with PHYs using
73:  // CMD_PROP_RX_ADV by Default
74:  //-------------------------------------------------------------------------------------------
75:
76:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
77:  {
78:      if(e & RF_EventRxEntryDone)
79:      {
80:          currentDataEntry = RFQueue_getDataEntry();
81:
82:          packetLength = *(uint8_t*)(&currentDataEntry->data);
83:          packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
84:
85:          memcpy(packet, packetDataPointer,
86:                    (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
87:
88:          crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
89:                   (uint16_t)(packet[packetLength + 1] << 0));
90:
91:          rssi = packet[packetLength + 2];
92:
93:          timestamp = ((uint32_t)(packet[packetLength + 3] << 0 ) +
94:                       (uint32_t)(packet[packetLength + 4] << 8 ) +
95:                       (uint32_t)(packet[packetLength + 5] << 16) +
96:                       (uint32_t)(packet[packetLength + 6] << 24));
97:
98:          status = packet[packetLength + 7];
99:
100:         RFQueue_nextEntry();
101:     }
102: }
```

3.3 Standard Packet Format (2 Length Bytes)

Necessary changes when going from 1 to 2 length bytes (standard packet format for PHYs exported with the advanced format), are on the following lines for the code examples in Section 3.3.1 and 3.3.2.

- Transmit Standard Packet Format (2 Length Bytes) with PHYs using CMD_PROP_TX_ADV by Default
  - Line 6 and 7
  - Line 40-50
- Receive Standard Packet Format (2 Length Bytes) with PHYs using CMD_PROP_RX_ADV by Default
  - Line 12 and 13
  - Line 18
  - Line 53 and 54
- Callback for Receiving Standard Packet Format (2 Length Bytes) with PHYs using CMD_PROP_RX_ADV by Default
  - Line 83-86

3.3.1 TX using CMD_PROP_TX_ADV and Standard Packet Format (2 Length Bytes)

```
//---------------------------------------------------------------------------------------------
// Transmit Standard Packet Format (2 Length Bytes) with PHYs using CMD_PROP_TX_ADV by Default
//---------------------------------------------------------------------------------------------

// Defines
#define PAYLOAD_LENGTH 3 // Max 4093 bytes
#define LENGTH_FIELD   2 // Changed from 1

uint8_t packet[LENGTH_FIELD + PAYLOAD_LENGTH];

static RF_Object rfObject;
static RF_Handle rfHandle;

void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    RF_cmdPropTxAdv.startTrigger.triggerType = 0x0; // Application specific settings
    RF_cmdPropTxAdv.startTrigger.pastTrig = 0x0;
    RF_cmdPropTxAdv.numHdrBits = 0x0;
    RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH;
    RF_cmdPropTxAdv.preTrigger.triggerType = 0x0;
    RF_cmdPropTxAdv.preTrigger.pastTrig = 0x0;
    RF_cmdPropTxAdv.syncWord = 0x930B51DE;
    RF_cmdPropTxAdv.pPkt = packet;

    // Necessary changes to the Setup command to support the standard packet format
    RF_cmdPropRadioDivSetup.formatConf.nSwBits = 0x20; // 32 bits sync word
    RF_cmdPropRadioDivSetup.formatConf.whitenMode = 0x0;  // No whitening

    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
                       &rfParams);
    RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);

    while(1)
    {
        //-----------------------------------------------------------------------------------
        // Could be placed outside the while(1) since the packet does not change
#ifdef SLR_MODE
        packet[0] = (uint8_t)(PAYLOAD_LENGTH);
        packet[1] = (uint8_t)(PAYLOAD_LENGTH >> 8);
#else
        packet[0] = (uint8_t)(PAYLOAD_LENGTH >> 8);
        packet[1] = (uint8_t)(PAYLOAD_LENGTH);
#endif
        for (uint16_t i = 2; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
        {
            packet[i] = i - 1;
        }
        //-----------------------------------------------------------------------------------
        RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
        RF_yield(rfHandle);
        usleep(500000);
    }
}
```

3.3.2 RX using CMD_PROP_RX_ADV and Standard Packet Format (2 Length Bytes)

```
1:   //---------------------------------------------------------------------------------------------
2:   // Receive Standard Packet Format (2 Length Bytes) with PHYs using CMD_PROP_RX_ADV by Default
3:   //---------------------------------------------------------------------------------------------
4:
5:   // Defines
6:   #define DATA_ENTRY_HEADER_SIZE  8    // Constant header size of a Generic Data Entry
7:   #define NUM_DATA_ENTRIES        2    // NOTE: Only two data entries supported
8:   #define CRC                     2    // 2 if .rxConf.bIncludeCrc = 0x1, 0 otherwise
9:   #define RSSI                    1    // 1 if .rxConf.bAppendRssi = 0x1, 0 otherwise
10:  #define TIMESTAMP               4    // 4 if .rxConf.bAppendTimestamp = 0x1, 0 otherwise
11:  #define STATUS                  1    // 1 if .rxConf.bAppendStatus = 0x1, 0 otherwise
12:  #define LENGTH_FIELD            2    // Changed from 1
13:  #define MAX_LENGTH              4093 // Changed from 255
14:  #define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS
15:
16:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
17:                                                                  // packetLength
18:  uint16_t packetLength; // Changed from uint8_t
19:
20:  static void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e);
21:
22:  static RF_Object rfObject;
23:  static RF_Handle rfHandle;
24:
25:  static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES, MAX_LENGTH,
26:                                   NUM_APPENDED_BYTES)]__attribute__((aligned(4)));
27:  static dataQueue_t dataQueue;
28:  static rfc_dataEntryGeneral_t* currentDataEntry;
29:  static uint8_t* packetDataPointer;
30:  rfc_propRxOutput_t rxStatistics;
31:  uint16_t crc16;
32:  int8_t rssi;
33:  uint32_t timestamp;
34:  uint8_t status;
35:
36:  void *mainThread(void *arg0)
37:  {
38:      RF_Params rfParams;
39:      RF_Params_init(&rfParams);
40:
41:      if(RFQueue_defineQueue(&dataQueue, rxDataEntryBuffer, sizeof(rxDataEntryBuffer),
42:                             NUM_DATA_ENTRIES, MAX_LENGTH + NUM_APPENDED_BYTES))
43:      {
44:          while(1);
45:      }
46:
47:      RF_cmdPropRxAdv.pktConf.bRepeatOk = 0x1; // Application specific settings
48:      RF_cmdPropRxAdv.pktConf.bRepeatNok = 0x1;
49:      RF_cmdPropRxAdv.pktConf.bCrcIncHdr = 0x1;
50:      RF_cmdPropRxAdv.rxConf.bAutoFlushCrcErr = 0x1;
51:      RF_cmdPropRxAdv.syncWord0 = 0x930B51DE;
52:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
53:      RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x10; // Changed from 0x8
54:      RF_cmdPropRxAdv.hdrConf.numLenBits = 0x10; // Changed from 0x8
55:      RF_cmdPropRxAdv.lenOffset = 0x0;
56:      RF_cmdPropRxAdv.pQueue = &dataQueue;
57:
58:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debugging)
59:
60:      // Necessary changes to the Setup command to support the standard packet format
61:      RF_cmdPropRadioDivSetup.formatConf.nSwBits = 0x20; // 32 bits sync word
62:      RF_cmdPropRadioDivSetup.formatConf.whitenMode = 0x0; // No whitening
63:
64:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
65:                         &rfParams);
66:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
67:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
68:                &callback, RF_EventRxEntryDone);
69:      while(1);
70:  }
71:
```

```
72:  //---------------------------------------------------------------------------------------------
73:  // Callback for Receiving Standard Packet Format (2 Length Bytes) with PHYs
74:  // using CMD_PROP_RX_ADV by Default
75:  //---------------------------------------------------------------------------------------------
76:
77:  void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
78:  {
79:      if(e & RF_EventRxEntryDone)
80:      {
81:          currentDataEntry = RFQueue_getDataEntry();
82:
83:          packetLength = ((uint16_t)(*(uint8_t*)(&currentDataEntry->data + 1) << 8) |
84:                          (uint16_t)(*(uint8_t*)(&currentDataEntry->data + 0) << 0));
85:          // Changed from
86:          // packetLength = *(uint8_t*)(&currentDataEntry->data);
87:
88:          packetDataPointer = (uint8_t*)(&currentDataEntry->data + LENGTH_FIELD);
89:
90:          memcpy(packet, packetDataPointer,
91:                 (packetLength + NUM_APPENDED_BYTES - LENGTH_FIELD));
92:
93:          crc16 = ((uint16_t)(packet[packetLength + 0] << 8) +
94:                   (uint16_t)(packet[packetLength + 1] << 0));
95:
96:          rssi = packet[packetLength + 2];
97:
98:          timestamp = ((uint32_t)(packet[packetLength + 3] << 0 ) +
99:                       (uint32_t)(packet[packetLength + 4] << 8 ) +
100:                      (uint32_t)(packet[packetLength + 5] << 16) +
101:                      (uint32_t)(packet[packetLength + 6] << 24));
102:
103:         status = packet[packetLength + 7];
104:
105:         RFQueue_nextEntry();
106:     }
107: }
```

4 References

1. Texas Instruments, SIMPLELINK-LOWPOWER-F2-SDK, webpage.
2. Texas Instruments, SmartRF™ Studio, webpage.
3. Texas Instruments, SysConfig, webpage.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025