Application Note

如何修改从 SysConfig 或 SmartRF™ Studio 导入/导出的 TX 和 RX 命令

Siri Johnsrud

摘要

本文档介绍了使 simplelink_cc13xx_cc26xx_sdk_x_xx_xx_xx [1]中的 rfPacketRx 和 rfPacketTx 示例能够同时使用标准和高级 TX 和 RX 命令（一个和两个长度字节）所需的必要更改以及如何修改代码以便能够使用 802.15.4g 格式接收和发送数据包。

使用一个长度字节时，有效载荷限制为 255 字节。当使用两个长度字节时，有效载荷限制为 4093 字节，而当使用 802.15.4g 格式时，长度字段为 11 位，有效载荷限制为 2045 字节。

如要发送或接收长度超过 4093 字节的数据包，必须使用无限数据包长度。本应用手册未讨论此内容。

内容

1 简介 2
2 使用标准命令导出的 PHY 设置 3
2.1 标准数据包格式（1 个长度字节） 3
2.2 标准数据包格式（2 个长度字节） 10
3 使用高级命令导出的 TX 和 RX 设置 15
3.1 高级数据包格式 16
3.2 标准数据包格式（1 个长度字节） 18
3.3 标准数据包格式（2 个长度字节） 21
4 参考资料 25

商标

1 简介

SmartRF™ Studio (2) 和 SysConfig (3) 都可以导出/导入特定器件所有已特性化 PHY 的寄存器设置。对于专有 PHY 组，可以使用标准 TX 和 RX 命令（CMD_PROP_TX 和 CMD_PROP_RX）或高级命令（CMD_PROP_TX_ADV 和 CMD_PROP_RX_ADV）导出/导入设置。基础 RX 和 TX SLA (Simplelink Academy) 演示了如何从 SysConfig 导入设置和/或从 SmartRF Studio 导出设置。即使两个命令集在支持的数据包格式上具备很高的灵活性，但默认的代码导出/导入仅实现了两种不同的数据包格式。图 1-1 展示了为两种不同情况配置的数据包格式。

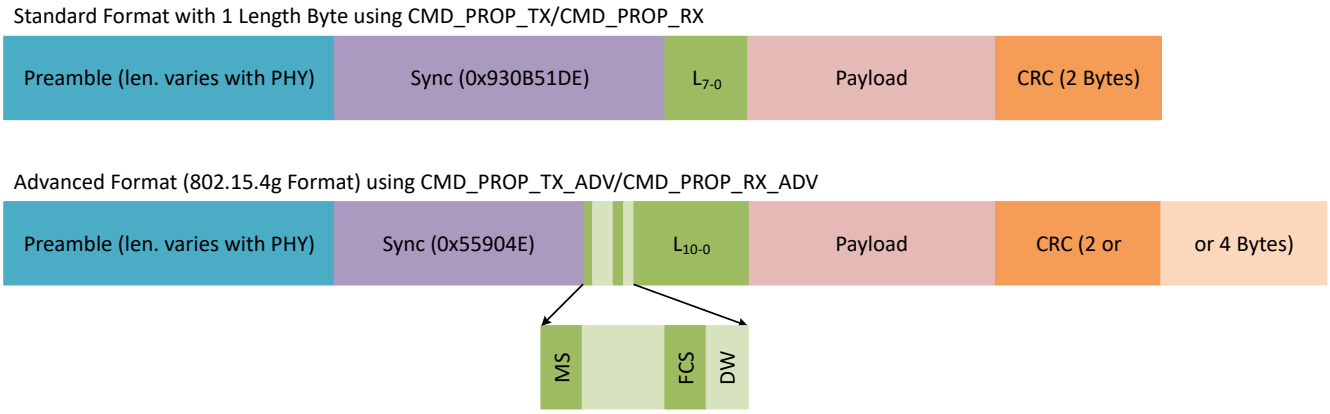

图 1-1. 标准与高级数据包格式

标准数据包格式也可通过高级命令实现，并且能够轻松将单长度字节改为双长度字节；默认采用高级数据包格式的 PHY，也可修改为发送和接收标准数据包格式的数据包（同时支持 1 和 2 个长度字节）。表 1-1 汇总了本应用手册所论述的各类组合方式。

表 1-1. 命令类型和数据包格式的组合

导入/导出命令类型	数据包格式	使用的命令类型	模式	最大有效载荷长度	部分
标准 CMD_PROP_TX CMD_PROP_RX	标准（1 个长度字节）	标准	TX	255	2.1.1
		标准	RX	255	2.1.2
		高级	TX	255	2.1.3
		高级	RX	255	2.1.4
	标准（2 个长度字节）	高级	TX	4093	2.2.1
		高级	RX	4093	2.2.2
高级 CMD_PROP_TX_ADV CMD_PROP_RX_ADV	高级 (802.15.4g)	高级	TX	2043 或 2045[1]	3.1.1
		高级	RX	2047[2]	3.1.2
	标准（1 个长度字节）	高级	TX	255	3.2.1
		高级	RX	255	3.2.2
	标准（2 个长度字节）	高级	TX	4093	3.3.1
		高级	RX	4093	3.3.2

[1] 最大有效载荷长度取决于 FSC 类型（2 字节或 4 字节 CRC）
[2] RX 侧的最大长度包括 CRC 字节数（2 或 4）

2 使用标准命令导出的 PHY 设置

在导出/导入使用标准 TX 和 RX 命令的 PHY 设置时，这些命令本身支持标准数据包格式（具有 1 个长度字节），无需修改 rfPacketTx 和 rfPacketRx 示例即可与这些 PHY 一起运行。以下各节 (2.1 - 2.2) 介绍了如何使用两种命令类型（标准和高级）实现标准数据包格式（1 个长度字节）以及如何修改标准数据包格式以包含双长度字节而非单长度字节。

2.1 标准数据包格式（1 个长度字节）

假设您要发送 3 个字节的有效载荷（0x01、0x02、0x03）。

该数据包在空中传输的格式如图 2-1 所示。

图 2-1. 标准数据包格式（1 个长度字节）

2.1.1 使用 CMD_PROP_TX 和标准数据包格式（1 个长度字节）的 TX

编写 rfPacketTx 示例是为了支持此格式，除了有效载荷长度和内容外，无需更改代码示例。但以下代码片段做了部分优化修改，以提升可读性。以下代码展示了如何修改 rfPacketTx.c 以实现每 0.5s 发送一次所需的有效载荷。

```
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: }
```

使用标准 TX 命令时，最大有效载荷长度为 255，并且长度字节 (PAYLOAD_LENGTH) 必须写入命令的.pktLen 字段。

2.1.2 使用 CMD_PROP_RX 和标准数据包格式（1 个长度字节）的 RX

为了能够使用 CMD_PROP_RX 接收中图 1-1 所示的标准数据包格式的数据包，可以使用以下代码（此处代码与原始 rfPacketRx 相比略有修改，以提高可读性并简化调试）。

```
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: }
```

本代码示例未实现长度过滤功能，且所有可附加的可选字节均已添加。如果要接收的唯一数据包是图 2-1 中所示的数据包，可将 MAX_LENGTH 从 255 修改为 3，所有长度字节大于 3 的数据包将被自动丢弃（因为 rxConf.bAutoFlushIgnored =1）。

2.1.3 使用 CMD_PROP_TX_ADV 命令和标准数据包格式（1 个长度字节）的 TX

在某些情况下，即便采用标准数据包格式，用户仍希望使用高级命令发送数据包。这可能是因为用户需要发送标准 TX 命令所不支持的更长前导码，侦听模式应用就是典型场景。

SysConfig (3) 和 SmartRF Studio (2) 均支持导入和导出 PHY 默认配置以外的其他命令。图 2-2 展示了如何使用 SysConfig 选择要导入到项目中的额外命令，而且图 2-3 展示了如何使用 SmartRF Studio 进行代码导出以完成此操作。

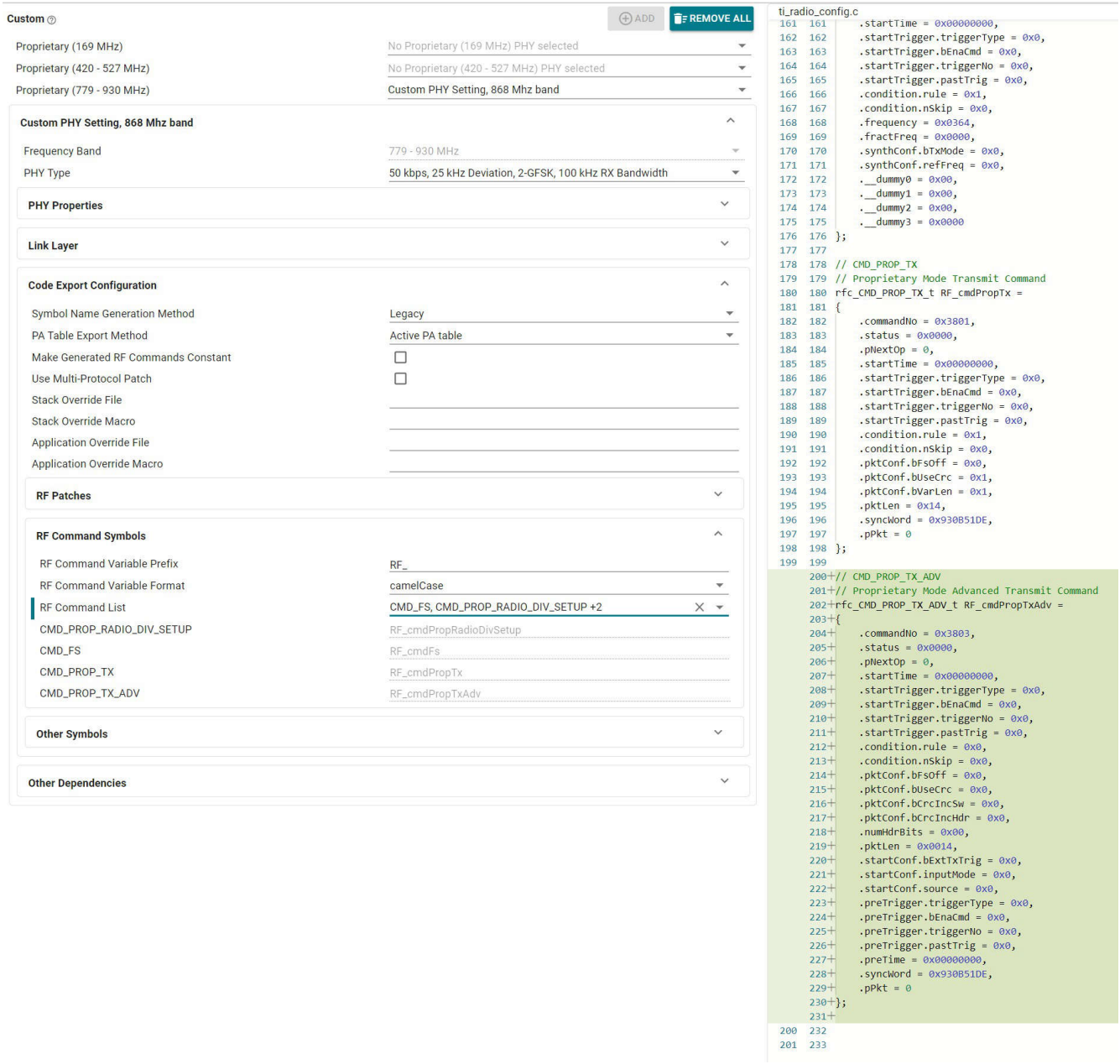

图 2-2. 在 SysConfig 中同时导入标准命令与高级 TX 命令

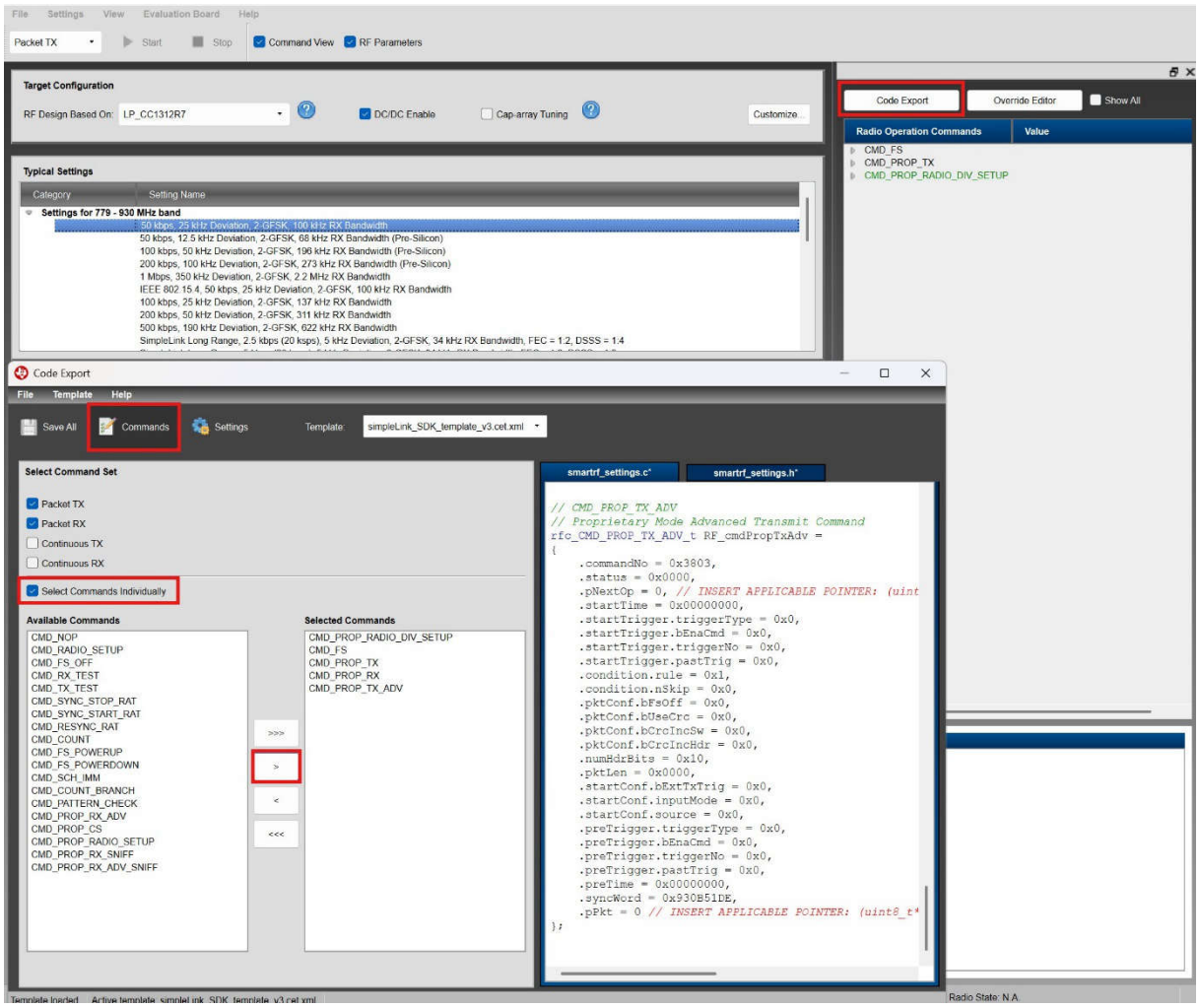

图 2-3. 在 SmartRF Studio 中同时导出标准命令与高级 TX 命令

通过这种方式添加额外命令时，SmartRF Studio 与 SysConfig 生成的配置并不完全相同，因此示例代码的编写兼容了两种工具的输出结果。使用高级 TX 命令时，长度字节必须与有效载荷一起手动写入数据包，并且命令的 .pktLen 字段必须同时包含长度字段和有效载荷长度。代码示例如下所示。

```
//--------------------------------------------------------------------------------------------
// Transmit Standard Packet Format with CMD_PROP_TX_ADV (1 Length Byte)
//--------------------------------------------------------------------------------------------

// Defines
#define PAYLOAD_LENGTH 3 // Max 255 bytes
#define LENGTH_FIELD   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.numHdrBits = 0x0; // Settings that must change to support
                                      // the standard packet format

    RF_cmdPropTxAdv.pktLen = LENGTH_FIELD + PAYLOAD_LENGTH; // Application specific settings
    RF_cmdPropTxAdv.pPkt = packet;

    // Settings to modify if going from a PHY that uses the standard TX command
    // to use the advanced TX command
    RF_cmdPropTxAdv.condition.rule = 0x1;
    RF_cmdPropTxAdv.pktConf.bUseCrc = 0x1;

    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
        packet[0] = PAYLOAD_LENGTH;

        for (uint16_t i = 1; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
        {
            packet[i] = i;
        }
        //--------------------------------------------------------------------------------------
        RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
        RF_yield(rfHandle);
        usleep(500000);
    }
}
```

2.1.4 使用 CMD_PROP_RX_ADV 和标准数据包格式（1 个长度字节）的 RX

如果我们要使用高级 RX 命令而不是标准 RX 命令接收标准数据包格式的数据包，该方案也可行，并且命令添加方式与第 2.1.3 节所述高级 TX 命令的添加方式一致。以下代码示例展示了如何使用高级 RX 命令接收标准数据包。此外，可以将 MAX_LENGTH 从 255 更改为 3，以启用数据包长度过滤。

```
//---------------------------------------------------------------------------------------
// Receive Standard Packet Format with CMD_PROP_RX_ADV (1 Length Byte)
//---------------------------------------------------------------------------------------
// 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.bCrcIncHdr = 0x1; // Settings that must change to support
    RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x8; // the standard packet format
    RF_cmdPropRxAdv.hdrConf.numLenBits = 0x8;

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

    // Settings to modify if going from a PHY that uses the standard RX command
    // to use the advanced RX command
    RF_cmdPropRxAdv.condition.rule = 0x1;
    RF_cmdPropRxAdv.pktConf.bUseCrc = 0x1;
    RF_cmdPropRxAdv.rxConf.bIncludeHdr = 0x1;
    RF_cmdPropRxAdv.endTrigger.triggerType = 0x1;

    RF_cmdPropRxAdv.rxConf.bIncludeCrc = 0x1; // Optional bytes to append
    RF_cmdPropRxAdv.rxConf.bAppendRssi = 0x1;
    RF_cmdPropRxAdv.rxConf.bAppendTimestamp = 0x1;
    RF_cmdPropRxAdv.rxConf.bAppendStatus = 0x1;

    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);
}
```

```
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 标准数据包格式（2 个长度字节）

从标准命令切换为高级命令的另一个原因可能是，用户希望发送和接收超过 255 字节的数据包，因此长度字段需要超过一个字节。

采用双字节长度字段时，前述示例所用的数据包如图 2-4 所示。

图 2-4. 标准数据包格式（2 个长度字节）

2.2.1 使用 CMD_PROP_TX_ADV 和标准数据包格式（2 个长度字节）的 TX

与第 2.1.3 节中示例中所示设置相比，无需更改设置。

仅需执行以下修改：

- 将 LENGTH_FIELD 从 1 修改为 2
- 将两个长度字节添加到数据包中

需修改的代码位于在第 7 行和第 38-48 行。

```
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: }
```

请注意，如果您正在使用 SimpleLink 远距离 (SLR) PHY 或传统远距离 PHY（仅限 CC13x0），则这两个长度字节的写入顺序需要与使用其他任何专有 PHY 时相反。

2.2.2 使用 CMD_PROP_RX_ADV 和标准数据包格式（2 个长度字节）的 RX

若要接收 2 个长度字节的标准数据包格式，需要修改相关设置，不能直接使用第 2.1.4 节中示例所示的设置。此外，需要更改一些定义和变量以及回调中的应用代码。需修改的代码位于第 11 行和第 12 行、第 15 行和第 16 行、第 46 行和第 47 行以及第 86-89 行。

```
//---------------------------------------------------------------------------------------
// Receive Standard Packet Format with CMD_PROP_RX_ADV (2 Length Bytes)
//---------------------------------------------------------------------------------------
// 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            2    // Changed from 1
#define MAX_LENGTH              4093 // Changed from 255
#define NUM_APPENDED_BYTES      LENGTH_FIELD + CRC + RSSI + TIMESTAMP + STATUS

uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - LENGTH_FIELD]; // Length stored in
uint16_t packetLength; // Changed from uint8_t                    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.bCrcIncHdr = 0x1;  // Settings that must change
                                               // to support the standard format
    RF_cmdPropRxAdv.hdrConf.numHdrBits = 0x10; // Changed from 0x8
    RF_cmdPropRxAdv.hdrConf.numLenBits = 0x10; // Changed from 0x8

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

    // Settings to modify if going from a PHY that uses the standard RX command
   // to use the advanced RX command
    RF_cmdPropRxAdv.condition.rule = 0x1;
    RF_cmdPropRxAdv.pktConf.bUseCrc = 0x1;
    RF_cmdPropRxAdv.rxConf.bIncludeHdr = 0x1;
    RF_cmdPropRxAdv.endTrigger.triggerType = 0x1;

    RF_cmdPropRxAdv.rxConf.bIncludeCrc = 0x1; // Optional bytes to append
    RF_cmdPropRxAdv.rxConf.bAppendRssi = 0x1;
    RF_cmdPropRxAdv.rxConf.bAppendTimestamp = 0x1;
    RF_cmdPropRxAdv.rxConf.bAppendStatus = 0x1;
    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);
}

```

```
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 使用高级命令导出的 TX 和 RX 设置

默认情况下，所有通过 CMD_PROP_TX_ADV 或 CMD_PROP_RX_ADV 命令导出/导入的 PHY 都采用 IEEE 802.15.4g 规定的数据包格式。该格式如图 3-1 所示。

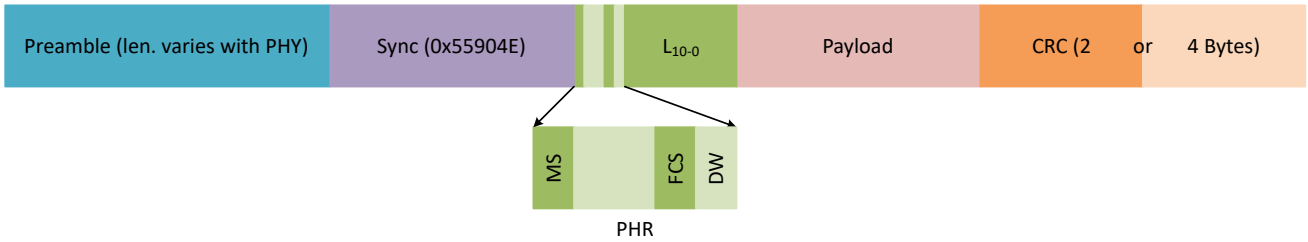

图 3-1. 高级数据包格式

同步字 (0x55904E) 后跟一个 2 字节长的标头 (PHR)，其中包含以下字段：

PHR[15] MS：模式开关（始终设置为 0）

PHR[14:13] 保留（无关）

PHR[12] FCS：FCS 类型（0：4 字节 CRC，1：2 字节 CRC）

PHR[11] DW：数据白化（0：禁用，1：启用）

PHR[10:0] 长度

长度字段应包括 CRC 字节（2 或 4 字节，具体取决于 PHR 中的 FCS 字段），即如果要发送前一个示例中使用的 3 字节长度的有效载荷，标头中的长度应是 5 或 7。例如，如果您想要使用 4 字节长度的 CRC (FCS = 0) 并开启数据白化 (DW = 1) 时，则标头将为 0x0807。

3.1 高级数据包格式

后续两节将说明如何修改 rfPacketTx 和 rfPacketRx 示例程序，实现高级数据包格式的数据包收发。在 TX 示例中，我们假设要使用 4 字节长度 CRC 并启用白化功能。

3.1.1 使用 CMD_PROP_TX_ADV 和高级数据包格式的 TX

标头以最低有效位优先的方式写入数据包，因此 CMD_PROP_TX_ADV 命令应发送的数据包将为 0x07、0x08、0x01、0x02、0x03。

由于长度字段为 11 位，因此当使用 2 字节长度 CRC 时，有效载荷字节数上限为 2045，当 CRC 为 4 字节长时，则为 2043。以下是可用于通过高级数据包格式发送数据包的代码。

```
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 使用 CMD_PROP_RX_ADV 和高级数据包格式的 RX

以下代码可用于使用高级数据包格式接收数据包。此代码仅可用于默认通过高级命令进行导出/导入的 PHY 设置。

```
1:   //---------------------------------------------------------------------------------------------
2:   // Receive Advanced Packet Format with CMD_PROP_RX_ADV
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                     4 // 4/2 (based on FCS) if .rxConf.bIncludeCrc = 0x1, else 0
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 HEADER_FIELD            2 // RF_cmdPropRx.rxConf.bIncludeHdr = 0x1
13:  #define MAX_LENGTH              2047 // Max length the radio will accept (incl. CRC bytes)
14:  #define NUM_APPENDED_BYTES      HEADER_FIELD + RSSI + TIMESTAMP + STATUS
15:
16:  uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - HEADER_FIELD]; // Header/Length stored in
17:  uint16_t packetLength;                                         // packetLength variable
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,
25:                                   MAX_LENGTH, 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:  uint32_t crc32;
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.rxConf.bAutoFlushCrcErr = 0x1;
50:      RF_cmdPropRxAdv.maxPktLen = MAX_LENGTH;
51:      RF_cmdPropRxAdv.pQueue = &dataQueue;
52:
53:      RF_cmdPropRxAdv.pOutput = (uint8_t*)&rxStatistics; // Optional (for debug)
54:
55:      rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
56:                         &rfParams);
57:      RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
58:      RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRxAdv, RF_PriorityNormal,
59:                &callback, RF_EventRxEntryDone);
60:      while(1);
61:  }
62:
```

```
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: }
```

此格式支持的最大数据包长度为 2047 字节（这包括 CRC（2 或 4 字节））。

请注意，IEEE 802.15.4g 未指定有效载荷（PSDU 数据）的格式。PSDU 仅被描述为位流，其内部数据格式由上层 MAC 规范决定。

在第 3.1.1 节和第 3.1.2 节中的代码示例中，有效载荷采用 MSB 优先的方式发送。为兼容 SmartRF Studio（默认情况下以 LSB 优先方式发送有效载荷），可以使用以下代码（RX 和 TX 端均适用）。

```
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 标准数据包格式（1 个长度字节）

即便 SmartRF Studio 和 SysConfig 导出/导入的是高级数据包格式设置，用户也可能需要使用标准数据包格式。第 3.2.1 节和 3.2.2 节将展示支持 1 个长度字节标准格式的 TX 和 RX 代码。

请注意，RX 和 TX 均需对设置命令 (CMS_PROP_RADIO_DIV_SETUP) 进行修改。该命令需要配置为 32 位长同步字，并且必须关闭白化（第 29-30 行（TX 示例）和第 60-61 行（RX 示例））。

3.2.1 使用 CMD_PROP_TX_ADV 命令和标准数据包格式（1 个长度字节）的 TX

```
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 使用 CMD_PROP_RX_ADV 和标准数据包格式（1 个长度字节）的 RX

```
//---------------------------------------------------------------------------------------------
// 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 标准数据包格式（2 个长度字节）

针对第 3.3.1 和 3.3.2 节中的代码示例，将从 1 个长度字节改为 2 个长度字节（以高级格式导出的 PHY 使用的标准数据包格式）时，需修改的代码位于以下行。

- 基于默认使用 CMD_PROP_TX_ADV 的 PHY，发送标准数据包格式（2 个长度字节）
  - 第 6 行和第 7 行
  - 第 40-50 行
- 基于默认使用 CMD_PROP_RX_ADV 的 PHY，接收标准数据包格式（2 个长度字节）
  - 第 12 行和第 13 行
  - 线路 18
  - 第 53 行和第 54 行
- 基于默认使用 CMD_PROP_RX_ADV 的 PHY，接收标准数据包格式的回调函数（2 个长度字节）
  - 第 83-86 行

3.3.1 使用 CMD_PROP_TX_ADV 和标准数据包格式 (2 个长度字节) 的 TX

```
1:  //---------------------------------------------------------------------------------------------
2:  // Transmit Standard Packet Format (2 Length Bytes) with PHYs using CMD_PROP_TX_ADV by Default
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.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: #ifdef SLR_MODE
41:         packet[0] = (uint8_t)(PAYLOAD_LENGTH);
42:         packet[1] = (uint8_t)(PAYLOAD_LENGTH >> 8);
43: #else
44:         packet[0] = (uint8_t)(PAYLOAD_LENGTH >> 8);
45:         packet[1] = (uint8_t)(PAYLOAD_LENGTH);
46: #endif
47:         for (uint16_t i = 2; i < (LENGTH_FIELD + PAYLOAD_LENGTH); i++)
48:         {
49:             packet[i] = i - 1;
50:         }
51:         //---------------------------------------------------------------------------------
52:         RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTxAdv, RF_PriorityNormal, NULL, 0);
53:         RF_yield(rfHandle);
54:         usleep(500000);
55:     }
56: }
```

3.3.2 使用 CMD_PROP_RX_ADV 和标准数据包格式（2 个长度字节）的 RX

```
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 参考资料

1. 德州仪器 (TI)，SIMPLELINK-LOWPOWER-F2-SDK，网页。
2. 德州仪器 (TI)，SmartRF™ Studio，网页。
3. 德州仪器 (TI)，SysConfig，网页。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月