AN1365: Using RAIL for IEEE 802.15.4 Applications with EFR32

This version of AN1365 has been deprecated. For the latest version, see docs.silabs.com.

****

This document describes using the Flex SDK for 802.15.4 development on EFR32™ Wireless parts. It includes features and limitations, using the radio configurator and supplied software components, and a description of the examples.

Proprietary is supported on all EFR32FG devices. For others, check the device's datasheet under Ordering Information > Protocol Stack to see if Proprietary is supported. In Proprietary SDK version 2.7.x, Connect is not supported on EFR32xG22.

KEY POINTS

- RAIL accelerated 802.15.4 features and limitations.
- Usage of the Radio Configurator for creating 802.15.4 compatible PHYs.
- Examples for demonstrating features.

1 Overview

RAIL provides numerous HW-accelerated IEEE 802.15.4 compliant MAC features, such as format filtering, address filtering, auto ACKing, and filtering based on the frame type. The MAC features are available if the radio is configured via official IEEE 802.15.4 PHYs loaded by certain RAIL APIs.

However, as RAIL exhibits only a limited subset of the standard PHYs defined by the 802.15.4 PHY layer specification, it is possible to define additional PHYs in the Radio Configurator interface compatible with the IEEE 802.15.4 standard on which the MAC features can also be enabled.

2 Configure 802.15.4 PHYs in RAIL

This section describes how to configure EFR32 radios with an IEEE 802.15.4 compatible PHY. Other PHYs might not be available in RAIL, but can be selected in the Radio Configurator.

Note: For the available PHYs and 802.15.4 specific RAIL features on the selected radio platform generation, please consult:

- the device's datasheet
- the Radio Configuration GUI
- the RAIL's documentation

2.1 2.4 GHz 250 kbps O-QPSK PHYs

The 2.4 GHz 250 kbps O-QPSK PHY is the most commonly used instance of the IEEE 802.15.4 PHYs supported in RAIL as an embedded configuration. Zigbee and Thread applications (indoor PAN network protocols) also configure this PHY when initializing the stack.

To configure this PHY, call the `RAIL_IEEE802154_Config2p4GHzRadio()` RAIL API. This procedure shall take the place of `RAIL_ConfigChannels()`.

This PHY initializes the radio for 2.4 GHz operation as defined by IEEE 802.15.4-2011 section 8.1.2.2.

This PHY uses a 1-byte header (PHR) where the 7 LSB bits determine the number of bytes in the PSDU (payload data).

RAIL supports this PHY along with each combination of the following options:

- Antenna Diversity
- Radio Coexistence
- FEM

Configure these PHYs via the `RAIL_IEEE802154_Config2p4GHzRadio[AntDiv][Coex][Fem]()` API variants.

These PHYs are highly optimized and validated configurations not available in the Radio Configurator.

Note: In these PHY configurations, only channels 11-26 are available. This PHY should not be confused with the SUN O-QPSK PHY. The SubGHz O-QPSK PHYs are documented in a subsequent section.

2.1.1 Connect 2.4 GHz O-QPSK PHYs

The Radio Configurator provides the `Connect 2.4GHz OQPSK 2Mcps 250kbps` PHY, which is mostly compatible with 802.15.4, but might not meet all requirements, and might fail regulatory requirements (such as FCC). This PHY has worse RF performance on most EFR32 radio generations than the PHYs introduced in the previous section.

However, configuring via the Radio Configurator can be useful if the user has some experience with the 802.15.4 packet format, but doesn't need to comply with the standard, as these can be customized in many ways (DSSS configuration, custom datarate, different syncword, and preamble configuration, channel number, and frequency assignment, etc.).

You can find more information in UG435.02: Using Silicon Labs Connect v3.x with IEEE 802.15.4.

2.2 Wi-SUN and SUN PHYs

The Wi-SUN and SUN PHYs are available in the Radio Configurator. However, some of the PHYs below might have an alternative configuration available via RAIL APIs. Use the PHYs from the Radio Configurator to be fully compatible with the standard.

2.2.1 SUN FSK

These PHYs can be found under the `Base` profile -> `IEEE 802.15.4 SUN FSK *`.

The SUN FSK PHYs supported in the Radio Configurator always use `phySunFskSfd = 0` SFD configuration.

2.2.1.1 SubGHz GB PHYs

The following PHYs are being used most commonly in UK metering standards, for example, Zigbee. These can be configured via RAIL API calls:

- `RAIL_IEEE802154_ConfigGB863MHzRadio()`
- `RAIL_IEEE802154_ConfigGB915MHzRadio()`

Note: These PHYs provide quasi SUN-FSK configuration but are not fully compliant with that standard (there are limitations on the PHY header's construction, e.g., the maximum payload length is limited to 127 bytes in these PHYs).

The available channels for the 863 MHz PHY ranges between 0xE0..0xFA, while for the 915 MHz PHY, these are 0x80..0x9A, 0xA0..0xA8, 0xC0..0xDA.

2.2.2 SUN O-QPSK

These PHYs can be found under the `SUN OQPSK` Profile.

2.2.3 Wi-SUN specific PHYs

- The HAN PHYs can be found under the `Wi-SUN HAN` Profile.
- The FAN PHYs can be found under the `Wi-SUN FAN 1.0` Profile or `Wi-SUN FAN 1.1` Profile.

2.3 PHYs are using the 802.15.4 frame format

Using the PHYs described in the following section, the radio might not be fully 802.15.4 compatible and could fail on regulatory and/or protocol certification. The 802.15.4 frame format is a well-known and common specification. Using such a radio configuration might accelerate the product development cycle, and it makes available the 802.15.4 specific RAIL APIs.

If the PHY you want to use is supported by RAIL, Silicon Labs recommends using it from there, as these are validated and fine-tuned through config options that might not be available on the radio configurator.

If you select any PHY from the `Connect` Profile and `Long Range` Profile in the Radio Configurator, most of the payload-related customization becomes unavailable because these PHYs ensure 802.15.4 frame format compatibility, but overall, these PHYs are not fully compatible with the standard.

However, in cases where you need more customization options available in the Radio Configurator, you might have set up 802.15.4 frame format compatibility as it is discussed in the following section.

2.4 Build Custom PHYs in the Radio Configurator

In cases where you need full control over the PHY customization, you need to set it up directly in the Radio Configurator.

To set up the base of the config, follow these steps:

- Disable all `Advanced` options.
- Set up modulation and frequencies as per the requirements of the standard.
- Configure channel spacing and channel numbering according to the standard.
- Preamble and Sync word (SFD) depends on the PHY format; set it up accordingly.
- Use LSB first everywhere unless stated otherwise. (*)

(*) SUN-FSK is an exception, which defines the length field MSB first. The length can be handled by the hardware, but you must keep in mind that you have to reverse the endianness of these bytes for both RX and TX in the application. Use the following code to set and get the length of the packet:

```
// length helper functions for Wi-SUN 15.4 frame formats
void setLength(uint8_t *payload, uint16_t length){
  payload[1] = __RBIT(length & 0xFF) >> 24;
  payload[0] = __RBIT(length >> 8) >> 24;
}
```

```
  uint16_t getLength(uint8_t *payload){
  uint16_t length = __RBIT(payload[1]) >> 24;
  length |= (__RBIT(payload[0]) >> 16) & 0x0700;
  return length;
}
```

2.4.1 Variable length setup for 1Byte PHR with 7bit length

On 2.4 GHz O-QPSK, this is covered by the embedded config. Otherwise:

- Check `Header Enable`
- Set `Header Size` to `1`
- Set `Frame Length Encoding` to `VARIABLE_LENGTH`
- Set `Frame Bit Endian` to `LSB_FIRST`
- Check `Length Includes CRC Bytes`
- Set `Minimum Length` to `0`
- Set `Maximum Length` to `127`
- Set `Variable Length Bit Size` to `7`
- Set `Variable Frame Length Adjust` to `0`
- Set `Variable Length Bit Endian` to `LSB_FIRST`
- Set `Variable Length Bit Location` to `0`

2.4.2 Variable length setup for 2Byte PHR and 11bit length (SUN FSK)

- Check `Header Enable`
- Set `Header Size` to `2`
- Set `Frame Length Encoding` to `VARIABLE_LENGTH`
- Set `Frame Bit Endian` to `LSB_FIRST`
- Check `Length Includes CRC Bytes`
- Set `Minimum Length` to `0`
- Set `Maximum Length` to `2048`
- Set `Variable Length Bit Size` to `11`
- Set `Variable Frame Length Adjust` to `0`
- Set `Variable Length Bit Endian` to `MSB_FIRST`
- Set `Variable Length Byte Endian` to `MSB_FIRST`
- Set `Variable Length Bit Location` to `0`

Note: The default RX FIFO size is 512B. You might want to increase that through `RAIL_SetRxFifo()` or use FIFO mode to download the packet in chunks.

2.4.3 CRC configuration

2.4.3.1 2B (CCIT-16)

- Set `CRC Input Bit Endian` to `LSB_FIRST`
- Uncheck `CRC Input Padding`
- Set `CRC Polynomial` to `CCITT_16`
- Set `CRC Seed` to `00`
- Set `CRC Output Bit Endian` to `MSB_FIRST`

- Set `CRC Byte Endian` to `MSB_FIRST`
- Uncheck CRC Invert
- Uncheck `CRC Header`

2.4.3.2 4B (ANSI X3.66-1979)

- Set `CRC Input Bit Endian` to `LSB_FIRST`
- Check `CRC Input Padding`
- Set `CRC Polynomial` to `ANSIX366_1979`
- Set `CRC Seed` to `0xFFFFFFFF`
- Set `CRC Output Bit Endian` to `MSB_FIRST`
- Set `CRC Byte Endian` to `MSB_FIRST`
- Check `CRC Invert`
- Uncheck `CRC Header`

2.4.3.3 Whitening (2B PHR only)

- Set `Whitening Output Bit` to `0`
- Set `Whitening Seed` to `0x01F0`
- Set `Whitening Polynomial` to `PN9`
- Uncheck `Whiten Header`

2.5 Best Practices for Configuring the PHY

- If the selected PHY is available via RAIL API calls, use that (except SUN and Wi-SUN PHYs).
- If the selected PHY is available via the Radio Configurator, use that.
- If the selected PHY isn't created yet, you can configure it on your own on the Radio Configurator GUI. Note that in this case, the protocol configurator might not meet the specification.

3 802.15.4 HW Accelerated MAC Features

The APIs introduced in the following chapters can work with any proprietary PHY (just like any PHY defined under the `Connect` Profile in the Radio Configurator) which meets the 802.15.4 frame format specification. This is the only requirement of the 802.15.4 PHY layer to use the protocol-specific RAIL APIs. The Connect SDK is a great example of how to use the protocol specific features on PHYs that are not fully 802.15.4 compatible.

The hardware does support address and frame filtering, Auto-ACK, and support to handle the frame pending bit in the ACK, implemented in RAIL. To enable it, use the `RAIL_IEEE802154_Init()` API call and the other calls related to address filtering or frame pending.

You can configure the Address Filtering and Auto-Ack features via the Init API, though those features have individual configuration APIs introduced in the following chapters.

3.1 Address Filtering

To configure address filtering, call `RAIL_IEEE802154_SetAddresses()` with a structure containing all addresses, or call the individual `RAIL_IEEE802154_SetPanId()`, `RAIL_IEEE802154_SetShortAddress()`, and `RAIL_IEEE802154_SetLongAddress()` APIs.

You can set up the addresses in the `RAIL_IEEE802154_Init()` API, too.

Configuring the addresses packet reception will be immediately aborted if the configured Address Filtering does not match.

RAIL supports intra-PAN (network) communication using multiple addresses.

Note that IEEE 802.15.4 Address Filter is not the same as the generic RAIL's Address Filter feature.

3.1.1 Best Practices for Address Filtering

Broadcast addresses are supported by default without any additional configuration and do not consume extra address slots.

Set unused addresses to the broadcast address (`0xFFFF`) to avoid false detections.

Do not change addresses during reception. Reconfiguring the Address Filter while receiving a packet may result in undesired behavior.

For more details, see the Detailed Description at IEEE 802.15.4 documentation and `RAIL_IEEE802154_AddrConfig_t` type's documentation.

3.2 Auto-Ack and Frame Pending

Auto-ACK is controlled by the `ackConfig` and `timings` fields passed to `RAIL_IEEE802154_Init()`. After initialization, they may be controlled using the normal `Auto-ACK` and `State Transitions` APIs.

The `ackTimeout` field only covers sync word detection of the received packet. Therefore, the ACK's PHY header and payload time shall be accommodated to get the correct `ackTimeout` setting. Find an example at RAIL's IEEE 802.15.4 – Detailed Description.

To listen for an ACK packet, you must transmit the packet with the `RAIL_TX_OPTION_WAIT_FOR_ACK` option.

The ACK frame includes the same sequence number as the frame that it responds to.

The AR bit in the Frame Control Field specifies whether an acknowledgment is required from the receiver device. Upon a reception, the receiver sends an ACK frame only if the frame passes the filtering. If this bit is 0, the receiver does not send an ACK packet. Usually, the AR is located at the 5th position of the FCF.

The Frame Pending bit is used for data polling. The end device polls the coordinator with a data request command. If the coordinator has a message to the end device, it responds with an ACK packet with the frame pending bit set in the FCF.

When in IEEE 802.15.4 mode, the ACK will generally have a 5-byte length (FCF, sequence number and FCS from the MAC fields, but have no payload), its Frame Type will be ACK, its Frame Version 0 (2003), and its Frame Pending bit will be false unless the `RAIL_EVENT_IEEE802154_DATA_REQUEST_COMMAND` event is triggered, in which case, it will default to the `RAIL_IEEE802154_Config_t::defaultFramePendingInOutgoingAcks` setting. If the default Frame Pending setting is incorrect, the app must call `RAIL_IEEE802154_ToggleFramePending()` (formerly

RAIL_IEEE802154_SetFramePending()) while handling the RAIL_EVENT_IEEE802154_DATA_REQUEST_COMMAND event.

This event must be turned on by the user and will fire whenever a data request is being received or fully received so that the stack can determine if there is pending data. Note that if the default Frame Pending bit needs to be changed, it must be done quickly. Otherwise, the ACK may already have been transmitted with the default setting. To trigger this event earlier, during the packet reception, use the RAIL_IEEE802154_EnableEarlyFramePending() API.

Check the return code of RAIL_IEEE802154_ToggleFramePending() to be sure that the Frame Pending bit was changed in time.

The RAIL_EVENT_IEEE802154_DATA_REQUEST_COMMAND is triggered only for Data Request MAC command frames requesting ACK. To trigger this event also for Data type frames call the RAIL_IEEE802154_EnableDataFramePending() API.

3.3 Enhanced ACK

The RAIL_IEEE802154_E_OPTION_ENH_ACK option allows the radio to receive packets with Frame Version 2 in the MAC header, as RAIL normally accepts packets with Frame Version 0 or 1. Enabling this option is required for handling Enhanced ACK frames.

RAIL assembles Immediate ACK frames automatically, but not Enhanced ACK frames. A dedicated RAIL event, RAIL_EVENT_IEEE802154_DATA_REQUEST_COMMAND, is implemented to parse a partly-received frame to determine the type of ACK needed to provide enough time to generate the ENH-ACK frame.

Note that this feature does not enable receiving Multipurpose frames. It can be enabled via the RAIL_IEEE802154_AcceptFrames() API.

Note: Thread protocol always sets the FP bit to 1.

3.4 Implicit Broadcast

The RAIL_IEEE802154_E_OPTION_IMPLICIT_BROADCAST option enables the MAC implicit broadcast feature. If the Frame Version is 2 and the frame lacks the destination PAN ID and address, then the packet is treated like a broadcast packet. If this option is disabled, such a packet is usually filtered.

3.5 Runtime PHY Reconfiguration with SUN FSK and Wi-SUN PHYs

3.5.1 2B Header Options

Call RAIL_IEEE802154_ConfigGOptions() with the RAIL_IEEE802154_G_OPTION_GB868 option to support dynamic CRC and whitening reconfiguration coded in a 2B PHR PHY.

In this mode, RAIL automatically reads the PHR field of the transmitted and received packet, and reconfigures the radio (i.e., enables/disables whitening for the rest of the payload, and selects 2/4-bytes CRC configuration) according to the FCS and WHITENING bitfields during the ongoing operation.

Though the API's name is a bit misleading, this enables all the 802.15.4g available features.

3.5.2 Dynamic FEC

To enable/disable 802.15.4g dynamic FEC feature use the RAIL_IEEE802154_G_OPTION_DYNFEC option.

SFD field (implemented as the syncword) indicates whether the rest of the packet is FEC encoded or not. As the SFD field is configured as the syncword on the EFR32 radio platform, it enables dual syncword detection to detect whether FEC should be enabled or not.

Address Filtering and FEC work only simultaneously on radio platforms where RAIL_IEEE802154_SUPPORTS_G_DYNFEC is defined. That means on former chips, Promiscuous mode should be enabled to support FEC.

Dynamic FEC configuration is supported on SUN FSK PHYs only.

Find more information at docs.silabs.com.

3.5.3 Mode Switch

The mode switch mechanism enables a device using the MR-FSK PHY to change its symbol rate and/or modulation scheme on a packet-by-packet basis. This feature can be enabled/disabled with the `RAIL_IEEE802154_G_OPTION_WISUN_MODESWITCH` option.

A specific mode switch PPDU is used to inform the receiver of the mode switch and specifies the new PHY mode of the following PPDU.

Switching to a new PHY mode with a higher rate typically by sending/receiving a specific Mode Switch packet that indicates the incoming new PHY mode. The Mode Switch packet is an FSK-modulated 2-byte PHY header with no payload.

Find more information at docs.silabs.com.

Note that this is the Mode Switch feature defined by Wi-SUN and not the 802.15.4g standard.

4 Examples

The following examples demonstrate basic functionalities of the 802.15.4 RAIL features. Usually, these tests require 2 RAILtest nodes, one operating as a transmitter and the other as the receiver. Issue the commands on both nodes if not stated otherwise.

For the sake of simplicity, we demonstrate the 802.15.4 features using the in-built 2.4 GHz 250 kbps O-QPSK and the SubGHz GB PHYs.

4.1 Applying a PHY from RAIL

```
rx 0 // Configs can be loaded only in radio Idle state
config2p4GHz802154
// or
config863GHz802154
// or
config915MHz802154
rx 1 // Make the RX active
```

Transmitting a packet (`tx 1`) from TX node is received on the RX node.

The default payload of the RAILtest application is constructed so that the device will transmit a 16 bytes length frames. Therefore, loading the subGHz PHYs also changes the first two bytes of the payload (the PHR field) to comply with the default 16 byte length packet configuration of the RAILtest application, if the default payload has not been changed yet.

To set different payload lengths, see the next example.

Use the `getChannel` command to see which channel the radio is on.

4.2 Variable Length Payload

To change the payload length of the transmitted packet, the Frame Length (`FL`) field of the PHR must be changed. The Frame Length field specifies the total number of bytes in the PSDU (`PSDU_LEN`). RAIL automatically sets the length of the packet according to the Frame Length field on both TX and RX modes.

The length of the PHR (`PHR_LEN`) can be 1 or 2 bytes in these PHYs.

Keep in mind that CRC bytes must be included in the Frame Length field. 802.15.4 supports both 2 and 4 bytes length CRC sequences, so the length of the CRC (`CRC_LEN`) can be 2 or 4.

In the examples below, `PACKET_LEN` denotes the packet length RAIL uses (including the PHR but excluding the CRC), while `PSDU_LEN` is the value of the Frame Length field defined by 802.15.4. In other words, RAIL expects `PACKET_LEN` bytes in the TX FIFO before transmitting the packet.

`PACKET_LEN = PSDU_LEN + PHR_LEN – CRC_LEN.`

If `PACKET_LEN` is higher than the default RAILtest payload (16), the `setTxLength <MAX_PACKET_LEN>` must also be issued, where `MAX_PACKET_LEN` is the length of the maximum expected payload length to transmit. If `PACKET_LEN > MAX_PACKET_LEN` RAIL generates a `RAIL_EVENT_TX_UNDERFLOW` event resulting in packet abort as there are no more bytes stored in the TX FIFO to transmit.

For the 2.4 GHz PHY (1B PHR), the Frame Length field is the 7 LSB bits of the first byte.

```
setTxPayload 0x00 <FL[6:0]>
```

where `FL` can range between 4 - 127. The MSB bit shall be set to 0.

```
CRC_LEN = 2
PHR_LEN = 1

FL = PSDU_LEN = PACKET_LEN + 1
```

More examples:

```
rx 0
```

```
config2p4GHz802154

setTxPayload 0x00 0x04 // PACKET_LEN = 3; PSDU_LEN = 4; shortest payload
// or
setTxPayload 0x00 0x0F // PACKET_LEN = 14; PSDU_LEN = 15; default payload
// or
setTxPayload 0x00 0x11 // PACKET_LEN = 16; PSDU_LEN = 18; longest payload using default TX length
config
// or
setTxLength 17
setTxPayload 0x00 0x12 // PACKET_LEN = 18; PSDU_LEN = 19; adjust TX length to prevent underflow error

tx 1
```

For the SubGHz PHYs (2B header), the Frame Length field is the 3 MSB bits of the first byte and all the 8 bits of the second byte in MSB first order.

```
setTxPayload 0x00 <PHR[0:5] + FL[8:11] >> 5> <FL[0:7]>
```

where `FL` can range between 4 - 2047.

```
CRC_LEN = 2 // default configuration
PHR_LEN = 2

FL = PSDU_LEN = PACKET_LEN
```

More examples:

```
rx 0

config863GHz802154
// or
config915MHz802154

setTxPayload 0x00 0x18 0xA0 // PACKET_LEN = PSDU_LEN = 5; shortest payload
// or
setTxPayload 0x00 0x18 0x70 // PACKET_LEN = PSDU_LEN = 14; default payload
// or
setTxPayload 0x00 0x18 0x08 // PACKET_LEN = PSDU_LEN = 16; longest payload using default TX length
config
// or
settxlength 17
setTxPayload 0x00 0x18 0x88 // PACKET_LEN = PSDU_LEN = 17; adjust TX length to prevent underflow error
```

In summary, to transmit a packet with N bytes of payload (i.e., MHR + MAC payload), set:

- `FL` to `N + CRC_LEN`, and
- `MAX_PACKET_LEN` to `N + PHR_LEN`, if needed.

The default RX and TX buffer size of RAILtest is set to 512 bytes.

The 863 and 915 MHz inbuilt PHYs limits the payload length to 127 (`0xFE`).

Use the `printTxPacket` command to see the TX payload!

Use the `configRxOptions 0x1` to store the CRC on valid packet reception.

4.3 2. Address Filtering

In this example, we will use the 2.4 GHz PHY only.

4.3.1 Enable Address Filtering

HW accelerated MAC features, such as frame and address filtering, are enabled by enabling 802.15.4 mode using the `enable802154` command.

```
rx 0
config2p4GHz802154
enable802154 rx 100 192 1000
rx 1
```

The default packets transmitted after these commands won't be received on the RX node, because the 802.15.4 mode automatically enables filtering without any address defined (except broadcast messages).

We should format a valid broadcast message to get the packet received without defining any address.

4.3.2 Formatting Broadcast Message

Configure the Frame Contol Field as follows. Pay attention to the bit endianness.

- Frame Type = `3b000` (Beacon)
- Security Enabled = `1b0`
- Frame Pending = `1b0`
- AR = `1b0`
- PAN ID Compression = `1b0`
- Reserved = `1b0`

The first byte of the PSDU is `0x00`.

- Sequence Number Compression = `1b0`
- IE Present = `1b0`
- Destination Addressing Mode = `0b10` (Short, 16 bit)
- Frame Version = `1b0`
- Source address = `2b10` (Short, 16 bit)

The second byte is `0x88`.

The FCF field is followed by the Sequence Number. This is a random number; we can set it arbitrarily in this example:

- Sequence Number = `0x00`

The third byte is `0x00`.

Set the Addressing fields next to the Sequence Number starting by the Destination:

- Destination PAN ID = `0xFFFF` (Broadcast)
- Destination Address = `0xFFFF` (Broadcast)
- Source PAN ID = `0x1234`
- Source Address = `0x5678`

The next 8 bytes are `0xFF 0xFF 0xFF 0xFF 0x34 0x12 0x78 0x56`.

And finally, we add 1 byte of MAC payload to the packet:

- Payload data = `0x55`

The last byte configured in the TX payload is `0x55`.

The total length of the packet is 15 bytes including the header (i.e., the PHR field). Set the first byte according to Example 1.

- `FL = PSDU_LEN = 14 = 0x0E` (11B MHR + 1B payload + 2B CRC)

```
settxpayload 0 0x0E 0x00 0x88 0x00 0xFF 0xFF 0xFF 0xFF 0x34 0x12 0x78 0x56 0x55
             | PHR |   FCF   | SN | D. PAN  | D. Addr.| S. PAN  | S. Addr | Payl.|
```

This packet is now received without defining the address of the receiver.

4.4 Short Address Example

As the next step, we set up an address on the receiver node.

The `setPanId802154` command configures the PAN ID of the device and `setShortAddr802154` sets the Short Address.

To keep the example simple, we set the PAN ID and Address on the RX node to `0x5555` and `0xAAAA`, respectively.

```
// TX node

setPanId802154 0x1234
setShortAddr802154 0x5678

// RX node

setPanId802154 0x5555
setShortAddr802154 0xAAAA
```

Now we can send a message directly addressed to our receiver node, keeping all other parameters of the packet the same as before:

```
settxpayload 0 0x0E 0x00 0x88 0x00 0x55 0x55 0xAA 0xAA 0x34 0x12 0x78 0x56 0x55
```

Setting PAN ID and Short Address on the transmitter does not filter packets from transmitting.

You can check at this point that changing only 1 bit in the Destination Address or PAN ID makes the RX node not receive the packet:

```
                                              |
settxpayload 0 0x0E 0x00 0x88 0x00 0x55 0x57 0xAA 0xAA 0x34 0x12 0x78 0x56 0x55
// or
settxpayload 0 0x0E 0x00 0x88 0x00 0x55 0x55 0xAE 0xAA 0x34 0x12 0x78 0x56 0x55
                                                 |
```

However, RAIL supports multiple addresses and PAN IDs defined at the same time.

You can register a new address by passing a second, optional argument to the `setPanId802154` and `setShortAddr802154` commands.

To receive the previous two wrongly formatted packets, use the following commands:

```
// RX node

setPanId802154 0x5755 1
setShortAddr802154 0xAAAA 1

setPanId802154 0x5555 2
setShortAddr802154 0xAAAE 2
```

Notice that the PAN ID and the address must be defined in pairs on a given index. The following packet still can't be received with the registered PAN ID and Short Address pairs:

```
settxpayload 0 0x0E 0x00 0x88 0x00 0x55 0x57 0xAE 0xAA 0x34 0x12 0x78 0x56 0x55
                                            |    |
```

4.5 Long Address Example

As a last address filtering example, receive a packet using the long address.

Set the Source address to `0x0123456789ABCDEF` with the `setLongAddr802154` command.

The TX packet shall be changed:

- Destination address = `2b11` (Long, 64 bit)

The second byte of the FCF is `0x8C`.

- Destination address is now 4 times longer (8 bytes)

The addressing fields (14 bytes) are: `0x55 0x55 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF 0x34 0x12 0x78 0x56`.

- We should add 6 bytes to the PHR:
- `FL = PSDU_LEN = 20 = 0x14` (17B MHR + 1B payload + 2B CRC)

```
settxpayload 0 0x14 0x00 0x8C 0x00 0x55 0x55 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF 0x34 0x12
             | PHR |   FCF   | SN | D. PAN  |     Destination Long Address          | S. PAN  |
0x78 0x56 0x55
 S. Addr | Payl.|

// or load it in two commands if the default argument length does not let the command pass in one run

settxpayload 0  0x14 0x00 0x8C 0x00 0x55 0x55 0x01 0x23 0x45 0x67
settxpayload 10 0x89 0xAB 0xCD 0xEF 0x34 0x12 0x78 0x56 0x55
```

Don't forget to increase the maximum packet length on the transmitter if needed:

```
setTxLength 19
```

As a complete example to send and receive a packet with a long address, use the following commands after resetting the devices:

```
// RX Node

rx 0
config2p4GHz802154
enable802154 rx 100 192 1000
rx 1
setPanId802154 0x5555
setLongAddr802154 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF

// TX Node

rx 0
config2p4GHz802154
enable802154 rx 100 192 1000
rx 1
settxpayload 0  0x14 0x00 0x8C 0x00 0x55 0x55 0x01 0x23 0x45 0x67
settxpayload 10 0x89 0xAB 0xCD 0xEF 0x34 0x12 0x78 0x56 0x55
settxlength 19
```

In the examples above, the transmitted frame was a Beacon type message. By default, enabling 802.15.4 mode, all frame types are enabled to receive. To filter out or reenable messages with different frame types, use the `acceptFrames` command.

The command's argument sets which frames can be received:

```
acceptFrames <MAC command> <ACK> <Data> <Beacon> [<Multipurpose>]
```

For example, after the following command, the receiver will reject data and Beacon type frames, but accept Ack and MAC command frames.

```
acceptFrames 1 1 0 0
```

4.5.1 Notes on Address Filtering

Broadcast messages can always be received regardless of the configured addresses.

Addresses can be deactivated by setting the address to the broadcast address (`0xFFFF`) at the corresponding index.

```
setPanId802154 0xFFFF <index>
setShortAddr802154 0xFFFF <index>
```

802.15.4 Address Filtering is fully disabled when the device is in promiscuous mode.

```
setPromiscuousMode 1
```

There is no constraint between the short and the long address and the PAN ID; you can set each of them independently at any index.

RAIL supports intra-PAN communication via multiple addresses defined using different PAN IDs.

4.6 Auto-ACK

Auto-ACK is automatically enabled by enabling 802.15.4 mode using the `enable802154` command.

```
rx 0
config2p4GHz802154
enable802154 rx 100 192 1000
rx 1
```

The third and the fourth parameter of the `enable802154` command sets the RX-to-TX turnaround time and the ACK timeout in microseconds, respectively.

Let's start with the TX packet and corresponding device configuration introduced in the previous example:

```
settxpayload 0 0x0E 0x00 0x88 0x00 0x55 0x55 0xAA 0xAA 0x34 0x12 0x78 0x56 0x55
```

To request an Acknowledge message, the first byte of the FCF field must be changed:

1. Request ACK packet by setting the AR bit (5th bit).
2. Beacon frames cannot be acknowledged: set Frame type (3 LS bits) to `3b001` (Data).

The first byte of the FCF shall be `0x21`:

```
settxpayload 0 0x0E 0x21 0x88 0x00 0x55 0x55 0xAA 0xAA 0x34 0x12 0x78 0x56 0x55
```

Also, RAIL must be informed that, after transmitting the packet, an ACK packet is expected to be received. To do that, issue the following command:

```
configTxOptions 1
```

See a complete example for testing Auto-ACK feature below:

```
// TX node

rx 0
config2p4GHz802154
enable802154 rx 100 192 1000
setPanId802154 0x1234
setShortAddr802154 0x5678
settxpayload 0 0x0E 0x21 0x88 0x00 0x55 0x55 0xAA 0xAA 0x34 0x12 0x78 0x56 0x55
configTxOptions 1
rx 1

// RX node

rx 0
config2p4GHz802154
enable802154 rx 100 192 1000
setPanId802154 0x5555
setShortAddr802154 0xAAAA
rx 1
```

Review the received ACK message's payload:

```
0x05 0x02 0x00 0x00
```

- `0x05`: This is the PHR and the Frame Length is 5
- `0x02 0x00`: the FCF field indicates that it is an ACK frame
- `0x00`: Sequence Number is identical to the packet which is acknowledged by this packet

To change the Sequence Number only, you can use the following command:

```
settxpayload 3 0x01
```

802.15.4 ACK frames are automatically assembled in RAIL, there is no need to set it up in the RX node.

4.6.1 Notes on Auto-ACK

Auto-ACK is fully disabled when the device is in promiscuous mode.

```
setPromiscuousMode 1
```

To disable or reenable auto ACK feature use the following command:

```
autoAckPause <RX> <TX>
```

Changing the ACK payload or its length prevents RAIL to assemble the packet. Also, printing the ACK packet will not result in printing the auto 15.4 ACK packet.

```
setAckPayload
setAckLength
printAckPacket
```

RAIL sets the Frame Pending bit in the ACK payload automatically.

4.7 Runtime PHY changes

4.7.1 Whitening and CRC Config

The SUN FSK PHYs define dynamic radio config changes (i.e., selecting 2/4 byte CRC - FCS type - or enabling/disabling whitening - DW - over the PSDU field) which is fully supported by RAIL. The radio is reconfigured on the fly during packet transmission and reception according to the PHR field.

This mode can be enabled by the `RAIL_IEEE802154_G_OPTION_GB868` option. Use the `set802154g` command to enable the `G_OPTION`s.

```
// TX node

rx 0
config915MHz802154
enable802154 rx 100 192 1000
set802154g 1
setPanId802154 0x1234
setShortAddr802154 0x5678
settxpayload 2 0x00 0x88 0x00 0x55 0x55 0xAA 0xAA 0x34 0x12 0x78 0x56 0x55
rx 1

settxpayload 0x00 0x00 0x70 // not-whitened & 4B FCS
settxpayload 0x00 0x08 0x70 // not-whitened & 42 FCS
settxpayload 0x00 0x10 0x70 // whitened & 4B FCS
settxpayload 0x00 0x18 0x70 // whitened & 2B FCS

// RX node

rx 0
config915MHz802154
enable802154 rx 100 192 1000
set802154g 1
setPanId802154 0x5555
setShortAddr802154 0xAAAA
rx 1
```

Alternatively, you can use the `set802154PHR` command once the radio is configured with a PHY capable of handling dynamic PHY reconfiguration.

For SUN FSK compatibility, the first argument should be 1. The second and third arguments set up which CRC is being used and whether whitening is enabled on the transmitted packet.

This command also sets up the frame length derived from the `setTxLength <PACKET_LEN>` config.

For the 2.4 GHz PHY, pass 0 as the first and omit the other arguments to set up the Frame Length in the PHR according to the configured TX packet length.

```
settxpayload 0x00 0x00 0x70 // not-whitened & 4B FCS
set802154PHR 1 0 0

settxpayload 0x00 0x08 0x70 // not-whitened & 42 FCS
set802154PHR 1 1 0

settxpayload 0x00 0x10 0x70 // whitened & 4B FCS
set802154PHR 1 0 1

settxpayload 0x00 0x18 0x70 // whitened & 2B FCS
set802154PHR 1 1 1
```

4.7.2 Dynamic FEC

SUN FSK protocol defines the dynamic FEC feature which is supported on certain EFR32 device generations. The SFD field type indicates whether FEC is enabled or not on the transmitted/received packet.

This mode can be enabled by the `RAIL_IEEE802154_G_OPTION_DYNFEC` option. Use the `set802154g` command to enable the `G_OPTION`s.

```
// TX node

rx 0
enable802154 rx 100 192 1000

//set_appropriate_channel
setconfigindex <Y>
setchannel <X>

set802154g 3
setPanId802154 0x1234
setShortAddr802154 0x5678
settxpayload 2 0x00 0x88 0x00 0x55 0x55 0xAA 0xAA 0x34 0x12 0x78 0x56 0x55
settxpayload 0x00 0x18 0x70 // whitened & 2B FCS
rx 1

// RX node

rx 0
setchannel <set_appropriate_channel>
enable802154 rx 100 192 1000
set802154g 3
setPanId802154 0x5555
setShortAddr802154 0xAAAA
rx 1
```

Transmit packet without FEC encoding (using syncword #0):

```
configTxOptions 0
tx 1
```

Transmit packet without FEC encoding (using syncword #1):

```
configTxOptions 0x4
```

```
tx 1
```

Using it on platforms where it is supported requires help from technical support to create an appropriate PHY.

The `set802154g` command requires an appropriate PHY configuration and idle state.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

www.silabs.com