Sure-Fi Radio Module Command Set

Intro:

At the most basic level, all communication with the Sure-Fi Module will be done through two UART connections on the side of the board. Each command should be in the format shown below which is an attention (ATTN) character which should always be 0x7E (a ~ in ASCII) or 0x7C (a | in ASCII), followed by one byte of command (CMD), one byte of length (LEN), and LEN bytes of payload. If LEN is 0x00 then no bytes of payload should follow.

UART packets going to the module are referred to as "Commands" while packets coming back from either interface are referred to as "Responses". The Commands that can be sent to the module are detailed below as well as all Responses you can expect to receive from each command. There are two separate UART interfaces to the module, one leads to the Sure-Fi Radio and one leads to the on-board Bluetooth chip which can be used for communication to any host BLE device. Each interface has it's own Commands and Responses.

Unless in the process of sending a command, all other information that does not match the specified ATTN character will be dropped by the module. If the ATTN, CMD, and LEN bytes are received and then none or only part of the payload are received before a break of 10ms or more the packet will be dropped by the module and a SureRsp_UartTimeout will be sent. If an unrecognized command is sent to the module then a SureRsp_Failure with the SureError_Unsupported error code will be sent.

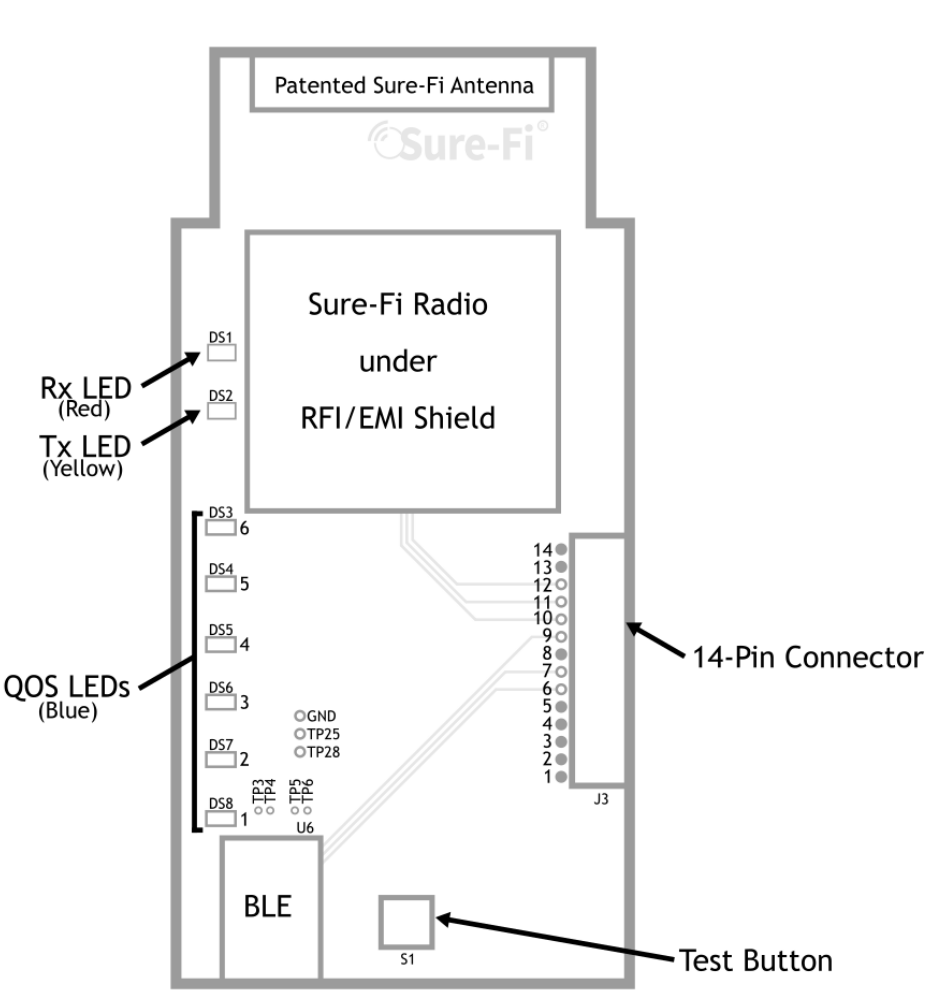
The Bluetooth chip supports two different ATTN characters: 0x7E and 0x7C. Commands prefixed with 0x7E will be echoed through the Module Service to the connected host device. Data written to the service from the host BLE device will be echoed to the application with the same ATTN character. Commands prefixed with 0x7C control the Bluetooth chip and it's operation and should follow the Bluetooth Commands set forth below. Likewise the Bluetooth Responses will come with the 0x7C ATTN character.

All the Commands, Responses, and other related enumerations and types are defined in sureFiModule.h which can be copied and included in your C or C++ project directly or used as a reference.

UART Command/Response Format

Byte	0	1	2	3+
Name	ATTN	CMD	LEN	PAYLOAD
Example	0x7E	0x40	0x00	-

Block Diagram



14-Pin Connector

Pin	Label	Direction	Description
1	GND		
2	GND		
3	GND		
4	+5V	IN	+5 VDC
5	GND		
6	U2RX	IN	BLE UART
7	U2TX	OUT	BLE UART
8	GND		
9	GPIO1	OUT	BLE Connected
10	GPIO0	OUT	Radio Busy/Interrupt
11	U1TX	OUT	Radio UART
12	U1RX	IN	Radio UART
13	GND		
14	GND		

Sure-Fi Radio Commands

Command Name	HEX Value	Payload Size (bytes)
Run-Time Commands		
SureCmd_DefaultSettings	0x30	0
SureCmd_ClearFlags	0x31	1
SureCmd_WriteConfig	0x32	1
SureCmd_SetIntEnableBits	0x33	4
SureCmd_Reset	0x34	0
SureCmd_Sleep	0x35	0
SureCmd_QosLightshow	0x36	0
SureCmd_TransmitData	0x37	0 – 62
SureCmd_StartEncrption	0x38	0
SureCmd_StopEncryption	0x39	0
SureCmd_ShowQualityOfService	0x3A	0
Get Information Commands		
SureCmd_GetStatus	0x40	0
SureCmd_GetIntEnableBits	0x41	0
SureCmd_GetModuleVersion	0x42	0
SureCmd_GetPacketTimeOnAir	0x43	0
SureCmd_GetRandomNumber	0x44	0
SureCmd_GetPacket	0x45	0
SureCmd_GetAckPacket	0x46	0
SureCmd_GetReceiveInfo	0x47	0
SureCmd_GetTransmitInfo	0x48	0
SureCmd_GetRegisteredSerial	0x49	0
Set Setting Commands		
SureCmd_SetAllSettings	0x50	14
SureCmd_SetRadioMode	0x51	1 or 3
SureCmd_SetFhssTable	0x52	1
SureCmd_SetReceiveUID	0x53	0 - 8
SureCmd_SetTransmitUID	0x54	0 - 8
SureCmd_SetReceivePacketSize	0x55	1
SureCmd_SetRadioPolarity	0x56	1
SureCmd_SetTransmitPower	0x57	1
SureCmd_SetAckData	0x58	0 – 62
SureCmd_SetTableHoppingEnabled	0x59	1
-	-	-
SureCmd_SetQosConfig	0x60	1
SureCmd_SetIndications	0x61	3
SureCmd_SetQuietMode	0x62	1
SureCmd_SetButtonConfig	0x63	1
SureCmd_SetAcksEnabled	0x64	1
SureCmd_SetNumRetries	0x65	1
Get Setting Commands		
SureCmd_GetAllSettings	0x70	0
SureCmd_GetRadioMode	0x71	0
SureCmd_GetFhssTable	0x72	0
SureCmd_GetReceiveUID	0x73	0
SureCmd_GetTransmitUID	0x74	0
SureCmd_GetReceivePacketSize	0x75	0
SureCmd_GetRadioPolarity	0x76	0
SureCmd_GetTransmitPower	0x77	0
SureCmd_GetAckData	0x78	0

SureCmd_GetTableHoppingEnabled	0x79	0
-	-	-
SureCmd_GetQosConfig	0x80	0
SureCmd_GetIndications	0x81	0
SureCmd_GetQuietMode	0x82	0
SureCmd_GetButtonConfig	0x83	0
SureCmd_GetAcksEnabled	0x84	0
SureCmd_GetNumRetries	0x85	0

Sure-Fi Radio Responses

Command Name	HEX Value	Payload Size (bytes)
Information Responses		
SureRsp_Status	0x40	4
SureRsp_IntEnableBits	0x41	4
SureRsp_ModuleVersion	0x42	11
SureRsp_PacketTimeOnAir	0x43	4
SureRsp_RandomNumber	0x44	4
SureRsp_Packet	0x45	1-62
SureRsp_AckPacket	0x46	1-62
SureRsp_ReceiveInfo	0x47	4
SureRsp_TransmitInfo	0x48	7
SureRsp_RegisteredSerial	0x49	1-31
Success/Failure Responses		
SureRsp_Success	0x50	1
SureRsp_Failure	0x51	2
SureRsp_UartTimeout	0x52	3
Get Setting Responses		
SureRsp_AllSettings	0x70	14
SureRsp_RadioMode	0x71	1 or 3
SureRsp_FhssTable	0x72	1
SureRsp_ReceiveUID	0x73	0-8
SureRsp_TransmitUID	0x74	0-8
SureRsp_ReceivePacketSize	0x75	1
SureRsp_RadioPolarity	0x76	1
SureRsp_TransmitPower	0x77	1
SureRsp_AckData	0x78	0-62
SureRsp_TableHoppingEnabled	0x79	1
-	-	-
SureRsp_QosConfig	0x80	1
SureRsp_Indications	0x81	3
SureRsp_QuietMode	0x82	1
SureRsp_ButtonConfig	0x83	1
SureRsp_AcksEnabled	0x84	1
SureRsp_NumRetries	0x85	1

Bluetooth Commands

Command Name	HEX Value	Payload Size (bytes)
Run-Time Commands		
BleCmd_StartAdvertising	0x30	0
BleCmd_StopAdvertising	0x31	0
BleCmd_CloseConnection	0x32	0
BleCmd_StartDfuMode	0x33	0
BleCmd_ReadExmem	0x34	5

BleCmd_WriteExmem	0x35	5-255
BleCmd_ClearResetFlag	0x36	5
BleCmd_ClearConnAttemptFlag	0x37	0
Get Information Commands		
BleCmd_GetFirmwareVersion	0x40	0
BleCmd_GetStatus	0x41	0
BleCmd_GetMacAddress	0x42	0
Set Setting Commands		
BleCmd_SetStatusUpdateBits	0x50	1
BleCmd_SetAdvertisingData	0x51	0-19
BleCmd_SetAdvertisingName	0x52	1-22
BleCmd_SetTemporaryData	0x53	0-255
BleCmd_SetGpioConfiguration	0x54	3
BleCmd_SetGpioValue	0x55	2
BleCmd_SetGpioUpdateEnabled	0x56	2
BleCmd_SetRejectConnections	0x57	1
Get Setting Commands		
BleCmd_GetStatusUpdateBits	0x70	0
BleCmd_GetAdvertisingData	0x71	0
BleCmd_GetAdvertisingName	0x72	0
BleCmd_GetTemporaryData	0x73	0
BleCmd_GetGpioConfiguration	0x74	0
BleCmd_GetGpioValue	0x75	0
BleCmd_GetGpioUpdateEnabled	0x76	0
BleCmd_GetRejectConnections	0x77	0

Bluetooth Responses

Command Name	HEX Value	Payload Size (bytes)
Run-Time Responses		
BleRsp_DfuNeedAdvData	0x30	4
BleRsp_ExmemData	0x31	5-255
Information Responses		
BleRsp_FirmwareVersion	0x40	4
BleRsp_Status	0x41	1
BleRsp_MacAddress	0x42	6
Success/Failure Responses		
BleRsp_Success	0x50	1
BleRsp_Failure	0x51	2
BleRsp_UartTimeout	0x52	3
BleRsp_BleWriteTimeout	0x53	3
Settings Responses		
BleRsp_StatusUpdateBits	0x70	1
BleRsp_AdvertisingData	0x71	0-19
BleRsp_AdvertisingName	0x72	1-22
BleRsp_TemporaryData	0x73	0-255
BleRsp_GpioConfiguration	0x74	3
BleRsp_GpioValue	0x75	2
BleRsp_GpioUpdateEnabled	0x76	2
BleRsp_RejectConnections	0x77	1

Information Sections

Sure-Fi Radio Commands (Run-Time)

SureCmd_DefaultSettings (0x30)

Description: Sets all of the settings on the module back to default. This includes the ConfigFlags byte in the Status Register and IntEnableBits.

Payload: N/A

Response: This command always responds with SureRsp_Success.

Example: [0x7E, 0x30, 0x00]

SureCmd_ClearFlags (0x31)

Description: Clear flags in the ClearableFlags byte in the Status Register using the mask given in the payload.

Payload: 1 byte binary mask. Having a bit set to 1 will clear the flag in the corresponding location in ClearableFlags register. Having a bit set to 0 will leave the flag bit unaffected in the register.

Response: If successful the command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: The command contained less than 1 byte payload
- **SureError_PayloadTooLarge**: The command contained more than 1 byte payload

Example: [0x7E, 0x31, 0x01, 0xFF]

SureCmd_WriteConfig (0x32)

Description: Write the ConfigFlags byte in the Status Register. See the Status Register section for more information on what each bit does.

Payload: 1 byte value to assign to the ConfigFlags byte in the status register

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: The command contained less than 1 byte payload
- **SureError_PayloadTooLarge**: The command contained more than 1 byte payload

Example: [0x7E, 0x32, 0x01, 0x02]

SureCmd_SetIntEnableBits (0x33)

Description: Write the IntEnableBits register which control which bits the module monitors for changes. If a change occurs in the Status Register on any of the bits that you have set to 1 the module will notify you of the status change. If InterruptDriven = 1 then the module will drive the interrupt pin active until you read the status register using

SureCmd_GetStatus. If InterruptDriven = 0 then the module will automatically send a SureRsp_Status whenever a interrupt enable bit changes.

Payload: 4 bytes corrisponding to the 4 bytes in the Status Register. Each bit can be enabled (1) or disabled (0).

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: The command contained less than 4 bytes payload
- **SureError_PayloadTooLarge**: The command contained more than 4 bytes payload

Example: [0x7E, 0x33, 0x04, 0xFF, 0xFF, 0xFF, 0xFF]

SureCmd_Reset (0x34)

Description: Forces the radio microcontroller to do a Software Reset.

Payload: N/A

Response: No response is given for this command. However, the module will perform it's regular startup process which includes sending the SureRsp_Status with the WasReset flag set.

Example: [0x7E, 0x34, 0x00]

SureCmd_Sleep (0x35)

Description: Puts the radio into sleep mode. This command is currently unsupported and will come in later firmware versions.

Payload: N/A

Response: Always responds with SureRsp_Failure with the error code set to SureError_Unsupported.

Example: [0x7E, 0x35, 0x00]

SureCmd_QosLightshow (0x36)

Description: Starts a lightshow on the QOS lights which can be used for debug purposes or as an indication to the user. As long as the lightshow is being shown the DoingLightshow bit in the Status Register will be set. There is no way to stop the lightshow after it has been started.

Payload: N/A

Response: This command always responds with SureRsp_Success.

Example: [0x7E, 0x36, 0x00]

SureCmd_TransmitData (0x37)

Description: Requests data to be sent over the Sure-Fi radio using the current settings. The length of the payload has to match the ReceivePacketSize minus the TransmitUID length otherwise the command will fail. The Busy bit in the Status Register must be 0 otherwise this command will fail. If this command succeeds the module will go into Transmit mode for a period of time dependant on the settings chosen. You will not be able to call SureCmd_TransmitData again until it is finished. Upon completion of the a transmission, successful or not, the TransmitFinished flag in the Status Register will be set to 1. The result of the transmission can then be obtained using SureCmd_GetTransmitInfo. If an acknowledgment

was received as part of the sending process and the acknowledgment contained data then the AckPacketReady flag in the Status Register will be set to 1. The acknowledgment data can be obtained using SureCmd_GetAckPacket.

Payload: 0 – 62 bytes of payload to be sent across the radio module. If successful the receiving end will get this payload using SureCmd_GetPacket.

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: The payload size + TransmitUID length was less than ReceivePacketSize
- **SureError_PayloadTooLarge**: The payload size + TransmitUID length was more than ReceivePacketSize
- **SureError_Busy**: The radio is busy and cannot perform a transaction right now

Example: [0x7E, 0x37, 0x0A, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99]

SureCmd_StartEncryption (0x38)

Description: Instructs the module to start the Encryption process. Because encryption requires a packet transaction to begin this command follows all the same criteria as SureCmd_TransmitData. If encryption was already started OR if another module has already started encryption with this module (EncryptionActive is set in the Status Register) then SureCmd_StartEncryption will simply respond with SureError_AlreadyStarted. This may be part of normal operation since both sides might be trying to start encryption when powered on. Both sides should attempt to start encryption at a random time after startup in order to prevent radio collisions (though these collisions should usually be handled by retries).

Encryption also requires certain criteria to be met before it will start. The TransmitUID and ReceiveUID must NOT be 0 bytes in length. The ReceivePacketSize must be a multiple of 16 bytes minus 2 bytes for module overhead (Ex. 14 bytes, 30 bytes, etc.). Acknowledgments must be enabled.

Payload: N/A

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_InvalidSettings**: One of the criteria for Encryption has not been met. Check that your settings meet the criteria listed above.
- **SureError_Busy**: The radio is busy and cannot perform a transaction right now
- **SureError_AlreadyStarted**: Encryption is already active

Example: [0x7E, 0x38, 0x00]

SureCmd_StopEncryption (0x39)

Description: Stops the current encryption process, putting the module back into the default encryption state.

Payload: N/A

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_NotStarted**: Encryption was not active

Example: [0x7E, 0x39, 0x00]

SureCmd_ShowQualityOfService (0x3A)

Description: Show the Quality of Service indication using the Quality of Service (QOS) lights on side of the module. The indication shows 1 - 6 LEDs depending on the RSSI value of the last packet or acknowledgment received. The indication lasts for 1000ms and the ShowingQos bit in the Status Register will be set during this time. This indication may also be shown automatically depending on what the QosConfig is set to.

Payload: N/A

Response: This command always responds with SureRsp_Success.

Example: [0x7E, 0x3A, 0x00]

Sure-Fi Radio Commands (Get Information)

SureCmd_GetStatus (0x40)

Description: Retrieves the current Status Register value. See Status Register section for more information about what the status bits mean. If InterruptDriven = 1 and the interrupt pin is currently active it will be deactivated when this command is received.

Payload: N/A

Response: This command will always respond with SureRsp_Status.

Example: [0x7E, 0x40, 0x00]

SureCmd_GetIntEnableBits (0x41)

Description: Retrieves the current IntEnableBits Register value. These are the same bits that were set by SureCmd_SetIntEnableBits.

Payload: N/A

Response: This command always responds with SureRsp_IntEnableBits.

Example: [0x7E, 0x41, 0x00]

SureCmd_GetModuleVersion (0x42)

Description: Retrieves the module version information. See the Version Numbers section for more information about what the version numbers mean.

Payload: N/A

Response: This command always responds with SureRsp_ModuleVersion.

Example: [0x7E, 0x42, 0x00]

SureCmd_GetPacketTimeOnAir (0x43)

Description: Retrieves the amount of time a single packet will take with the current settings on the module.

Payload: N/A

Response: This command always responds with SureRsp_PacketTimeOnAir.

Example: [0x7E, 0x43, 0x00]

SureCmd_GetRandomNumber (0x44)

Description: Retrieves a random number from the radio which is seeded from the radio receiver white noise. This can be a useful source if the application has no means of generating a true random. It is recommended that the application use the number as a seed for it's own pseudo random number generator rather than a random source by itself.

Payload: N/A

Response: This command always responds with SureRsp_RandomNumber.

Example: [0x7E, 0x44, 0x00]

SureCmd_GetPacket (0x45)

Description: Retrieves the data from last successful packet that was received by the Sure-Fi radio. This should usually be called whenever the RxPacketReady flag gets set in the Status Register.

Payload: N/A

Response: On success the command responds with SureRsp_Packet. Otherwise it sends SureRsp_Failure with one of the following error codes:

- **SureError_Busy:** The module has not yet received a valid packet since startup

Example: [0x7E, 0x45, 0x00]

SureCmd_GetAckPacket (0x46)

Description: Retrieves the data from last acknowledgment that contained data. This should usually be called whenever the AckPacketReady flag gets set in the Status Register.

Payload: N/A

Response: On success the command responds with SureRsp_AckPacket. Otherwise it sends SureRsp_Failure with one of the following error codes:

- **SureError_Busy:** The module has not yet received an acknowledgment containing data since startup

Example: [0x7E, 0x46, 0x00]

SureCmd_GetReceiveInfo (0x47)

Description: Retrieves the information about the last successfully received packet.

Payload: N/A

Response: This command always responds with SureRsp_ReceiveInfo.

Example: [0x7E, 0x47, 0x00]

SureCmd_GetTransmitInfo (0x48)

Description: Retrieves the information about the last transmission that has finished.

Payload: N/A

Response: This command always responds with SureRsp_TransmitInfo.

Example: [0x7E, 0x48, 0x00]

SureCmd_GetRegisteredSerial (0x49)

Description: Retrieves the Registered Serial String of the module. This string is unique to each module. The application can use it for whatever purposes it sees fit but it is mainly available for internal use at Sure-Fi.

Payload: N/A

Response: This command always responds with SureRsp_RegisteredSerial.

Example: [0x7E, 0x48, 0x00]

Sure-Fi Radio Commands (Set Settings)

SureCmd_SetAllSettings (0x50)

Description: Sets most of the settings using a single command. This function is useful for configuring the radio on startup so you can easily set all settings at the same time. The payload must contain 14 bytes that are described in the ModuleSettings_t section. The command does not support setting the ReceiveUID, TransmitUID, or AckData since these are all variable length. You must send those commands separately. You also cannot set RadioMode to RadioMode_Custom using the SetAllSettings command. You must call SureCmd_SetRadioMode separately if you want to set it to this value.

Payload: 14 bytes of payload containing the various setting values. See the ModuleSettings_t section for more information.

Response: If any of the settings are invalid this command will return a SureRsp_Failure as if the application had called the corresponding command specifically. Therefore this command might return multiple failure responses if multiple invalid settings were sent. If all settings were updated successfully and no Failure responses were returned then this command responds with SureRsp_Success with SureCmd_SetAllSettings in the payload.

Example: [0x7E, 0x50, 0x0E, 0x02, 0x20, 0x0A, 0x02, 0x1F, 0x01, 0x06, 0x00, 0x00, 0x00, 0x00, 0x12, 0x01, 0x02]

SureCmd_SetRadioMode (0x51)

Description: Changes the RadioMode of the module. The RadioMode affects the distance and data rate along with various other physical aspects of the radio signal. The details of all the radio options can be found in the Radio Mode section.

Payload: For all regular RadioModes the payload must be 1 byte containing a value that corresponds to one of the modes. If you send RadioMode_Custom, however, the payload must be 3 bytes total. The second byte must be some SpreadingFactorOption, and the third byte must be some BandwidthOption.

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte (or 3 bytes if RadioMode_Custom is sent)
- **SureError_PayloadTooLarge**: Payload was more than 1 byte (or 3 bytes if RadioMode_Custom is sent)
- **SureError_InvalidValue**: One of the values sent did not match any of the accepted values.

Example: [0x7E, 0x51, 0x01, 0x02]

SureCmd_SetFhssTable (0x52)

Description: Sets the Freqquency Hopping Spread Spectrum (FHSS) table to use when transmitting packets. In order for two modules to communicate they must be on the same FHSS table. At the same time, two modules that are on seperate hopping tables should not interfere with each other when transmitting at the same time. This is a achieved through a variety of other options based off the FHSS table chosen by the application. For this reason, the FHSS table acts as a sort of "Pairing Channel" between devices. There will be less collisions if every pair of modules are on different hopping tables. See the RadioModes section for more information about how the FHSS table and the radio settings are related.

Payload: 1 byte between 0 - 215 (0x00 - 0xD7)

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte
- **SureError_InvalidValue**: The chosen table was more than 215 (0xD7)

Example: [0x7E, 0x52, 0x01, 0x20]

SureCmd_SetReceiveUID (0x53)

Description: Sets the Receive UID of this module to any desired array of bytes. Every packet that is received successfully on the module is checked against this UID to ensure the packet is for this module. The UID can be any number of bytes up to 8 bytes total. A UID length of 0 bytes effectively disables any checks on the packet to effectively making the module receive all packets that are transmitted with compatible radio settings.

If two modules are near each other and have the same Receive UID and acknowledgments are enabled they will acknowledge the same packet at the same time and collide with each other effectively making it impossible for the transmitting module to tell if the packets are making it through. If you need packets to go to multiple destinations with a single transmission then acknowledgments should be disabled on the receiving modules. You can, however, set the TransmitUID and ReceiveUID to the same value on two modules with acknowledgments and have them communicate just fine.

Payload: 0 – 8 bytes of Receive UID

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooLarge**: Payload was more than 8 bytes

Example: [0x7E, 0x53, 0x03, 0x12, 0x34, 0x56]

SureCmd_SetTransmitUID (0x54)

Description: Sets the TransmitUID for all outgoing packets to any desired array of bytes. The Transmit UID is used in tandem with the ReceiveUID of another module to successful perform acknowledgments. Generally the TransmitUID and ReceiveUID should be the same length since the TransmitUID of one side has to match the ReceiveUID of the other side in order for the packet to succeed. However, this is not enforced and in rare cases there are applications that will run with a different length TransmitUID and ReceiveUID.

Payload: 0 – 8 bytes of Transmit UID

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooLarge**: Payload was more than 8 bytes

Example: [0x7E, 0x54, 0x03, 0x12, 0x34, 0x56]

SureCmd_SetReceivePacketSize (0x55)

Description: Sets the receive packet size of the radio. The ReceivePacketSize can be any number between 1 and 62 bytes. Both the sending and receiving sides must have the same ReceivePacketSize in order for the packet to succeed. The ReceivePacketSize must be greater than the ReceieveUID and TransmitUID lengths. The larger the packets are the more time it will take to complete transactions, however larger packet sizes will give you higher overall throughput. The

module does not support variable packet lengths and therefore the trade-off of larger packet sizes are that smaller amounts of data take longer to send. The ReceivePacketSize also affects how large your payload can be when calling SureCmd_TransmitData. The module does not currently support sending different sized packets than receiving.

Payload: 1-byte setting the number of bytes to be sent or received in all packets

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte

Example: [0x7E, 0x55, 0x01, 0x0E]

SureCmd_SetRadioPolarity (0x56)

Description: Sets the polarity of the module. It can be set to 0x00 to disable it (default setting), or it can be set to 0x01 (Up) or 0x02 (Down). If the polarity is set to something other than 0x00 then one side must set the polarity to 0x01 and the other side to 0x02 in order for the communications to work. The polarity is meant to be used with systems where there are two distinct sides (Like a central and remote bridge) and helps reduce traffic between units of the same side especially when they are in close proximity (like many central units in a single room). It essentially provides a way for units of the same polarity to not receive each other's transmissions.

Payload: 1-byte either 0x00 (disabled), 0x01 (Up), or 0x02 (Down)

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte
- **SureError_InvalidValue**: A value greater than 0x02 was passed

Example: [0x7E, 0x56, 0x01, 0x02]

SureCmd_SetTransmitPower (0x57)

Description: Sets the power to be used during transmissions. This can be set to any whole value between 0dBm (1mWatt) and 30dBm (1Watt). The default setting is 1 Watt since this will give you the maximum range and penetration.

Payload: 1-byte between 0x01 (0dBm) and 0x1F (30dBm)

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte
- **SureError_InvalidValue**: Either 0x00 or a value greater than 0x1F was passed

Example: [0x7E, 0x57, 0x01, 0x1F]

SureCmd_SetAckData (0x58)

Description: Sets the acknowledgment data to be sent in any acknowledgments the module receives. This command is similar to SureCmd_TransmitData in that the payload has to be the ReceivePacketSize - TransmitUID length. However, AckData can be set at any time even when transmitting a message or in the process of acknowledging a packet since it is a passive setting. Due to timing constraints on the radio the module does not support setting the AckData immediately following the receive of a packet in order to send data back in the acknowledgment based off the contents of the packet. Ack data is useful for information that can be actively updated by the application that the other side might want to know at any time. In this way the other side can transmit a message when it wants this information and process the AckData when it comes back. If the AckData is set to 0 length the acknowledgments do not trigger the AckPacketReceived status bit in the Status Register on the transmitting module.

Payload: 0 – 62 bytes of data to be transmitted when an acknowledgment is sent. The payload size must be ReceivePacketSize - TransmitUID length otherwise this command will respond with a SureRsp_Failure.

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was not 0 bytes and shorter than ReceivePacketSize - TransmitUID length
- **SureError_PayloadTooLarge**: Payload was not 0 bytes and longer than ReceivePacketSize - TransmitUID length

Example: [0x7E, 0x58, 0x0A, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99]

SureCmd_SetTableHoppingEnabled (0x59)

Description: Enables or disables Table Hopping functionality on the module. Table hopping is another feature on top of Frequency Hopping that allows us to change which FHSS table we are using every few packets at the cost of a little synchronization delay (the length of which depends on the number of retries set). The main purpose of changing tables is to allow more transmissions from a single module within the FCC limit of 20 seconds. See the Table Hopping section for more information on how this works.

Payload: 1 byte either 0x00 for disabled or 0x01 for enabled

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte

Example: [0x7E, 0x59, 0x01, 0x01]

SureCmd_SetQosConfig (0x60)

Description: Sets the configuration of the Quality of Service (QOS) indication that happens when packets are received. This indication is meant to be used to show the user a representation of the RSSI of a received packet (acknowledgments can be used as well). This configuration settings tells the module in what scenarios it should automatically show the QOS indication. Regardless of the setting here you can also tell the module to show the QOS indication at any time using the SureCmd_ShowQualityOfService command. The QOS indication is a display of 1-6 LEDs based off the RSSI of the last packet. The indication lasts for 1000ms and overrides any Indications, set by SureCmd_SetIndications, during that time. See the Quality of Service section for a breakdown of how RSSI values match up to the 6 LEDs.

Payload: 1 byte corresponding to one of the following options:

- **QosConfig_Manual** (0x01): Never show the QOS automatically
- **QosConfig_OnReceive** (0x02):
- **QosConfig_OnTransmit** (0x03):

- **QosConfig_OnReceiveAndTransmit** (0x04):
- **QosConfig_OnAckData** (0x05):
- **QosConfig_OnReceiveAndAckData** (0x06):

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte
- **SureError_InvalidValue**: Either 0x00 or a value greater than 0x06 was passed

Example: [0x7E, 0x60, 0x01, 0x06]

SureCmd_SetIndications (0x61)

Description: Sets the behavior of the 6 QOS lights while idling. This can be used to show many different sorts of indications to the user for any purpose the application sees fit. The command sets the indication mode of all 6 LEDs at the same time.

Payload: 3 bytes (6 nibbles) with each nibble corresponding to a single LED. The first byte is for LED 1 and 2 (lowest 2 LEDs on the module) where the lower nibble is LED1 and the upper nibble is LED2. The other two bytes follow the same pattern. The example below sets LED1 to option 1, LED2 to option 2, etc. Each nibble can be one of the following values:

- **Indication_Off** (0x0): The LED is off
- **Indication_On** (0x1): The LED is on
- **Indication_Blink1Hz** (0x2): The LED blinks with 50% duty cycle at 1Hz
- **Indication_Blink2Hz** (0x3):The LED blinks with 50% duty cycle at 2Hz
- **Indication_Pattern1** (0x4): The LED blinks 1 time every 2000ms
- **Indication_Pattern2** (0x5): The LED blinks 2 times every 2000ms
- **Indication_Pattern3** (0x6): The LED blinks 3 times every 2000ms
- **Indication_Pattern4** (0x7): The LED blinks 4 times every 2000ms

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 3 bytes
- **SureError_PayloadTooLarge**: Payload was more than 3 bytes
- **SureError_InvalidValue**: Either 0x0 or a value greater than 0x7 was passed for one of the LEDs

Example: [0x7E, 0x61, 0x03, 0x21, 0x43, 0x65]

SureCmd_SetQuietMode (0x62)

Description: Enables or disables quiet mode on the module. Quiet mode is an override for AcksEnabled and ButtonConfig and is intended to be used when the Bluetooth is performing some critical operation. If quiet mode is enabled the module will not acknowledge packets and will not perform any action if the button is pressed. This prevents the Sure-Fi radio from performing an unexpected transmission which would require the Bluetooth module to

disconnect. Quiet mode does not prevent transmissions performed through SureCmd_TransmitData or SureCmd_StartEncryption. The application should refrain from sending this command if the Bluetooth needs to stay connected.

Payload: 1 byte either 0x00 for disabled or 0x01 for enabled

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte

Example: [0x7E, 0x62, 0x01, 0x00]

SureCmd_SetButtonConfig (0x63)

Description: Sets the button configuration which is a combination of two distinct options into a single configuration byte. The upper nibble sets the button hold time which is the number of seconds the button is required to be held down before the ButtonHeld flag gets set in the Status Register. The lower nibble sets the button configuration which handles what should be done when the button is pressed and released before the button held time has been satisfied.

Payload: 1 byte where upper and lower nibble are two distinct options. The upper nibble should be any value between 0x1 and 0xF which defines the number of seconds the button is required to be held before the ButtonHeld flag is set in the Status Register. The lower nibble defines what the button should do when it is pressed and released before the hold time is reached. It can be one of the following values:

- **ButtonConfig_NoAction** (0x1): No action is taken automatically by the module. This is useful when you would like to respond to the button press in some other way. The button press can still be detected using the ButtonDown or ButtonPressed bits in the Status Register.
- **ButtonConfig_SendZeros** (0x2): A packet containing the current TransmitUID and a payload of 0x00 bytes is sent if the radio is not already busy.
- **ButtonConfig_SendAckData** (0x3): A packet containing the current TransmitUID and the current AckData is sent if the radio is not already busy. If AckData is set to 0 bytes (the default) then this option performs the same as ButtonConfig_SendZeros.

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 bytes
- **SureError_PayloadTooLarge**: Payload was more than 1 bytes
- **SureError_ValueTooLow**: The upper nibble was 0x0
- **SureError_InvalidValue**: The lower nibble was either 0x0 or greater than 0x3

Example: [0x7E, 0x63, 0x01, 0x12]

SureCmd_SetAcksEnabled (0x64)

Description: Enables or disables acknowledgments on the radio. Acknowledgments are sent for all packets that are received by this module so that the sending party can confirm the packet made it successfully. If acknowledgments are disabled then the module will not respond to packets but it will still receive them like normal. With acknowledgments disabled the module will also not wait for acknowledgments when transmitting packets and will always assume the packet succeeded on the first try effectively disabling retries. The application is then required to determine if the packet

was successful or handle the unreliability of the transport itself. This can be useful for certain applications that need more control over their packets or lower overhead and latency.

Payload: 1 byte either 0x00 for disabled or 0x01 for enabled

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte

Example: [0x7E, 0x64, 0x01, 0x01]

SureCmd_SetNumRetries (0x65)

Description: Sets the number of retries that will be done if the first try fails. ("retry" refers to the transmission of a packet after the first "try" that was initiated by SureCmd_TransmitData). Have retries set to a value greater than 0 helps with collisions that can occur with other Sure-Fi Modules or 900MHz devices. A value of 1 or 2 retries is recommended for most applications. The maximum value is determined by the maximum number of packets we can transmit within a 20 second period on a single FHSS table. See the Radio Mode and Table Hopping sections for more information about FCC limits.

Payload: 1 byte representing the desired number of retries. This value can be 0 but must not be more than the maximum number of retries allowed (which varies based off other radio settings).

Response: If successful this command will respond with SureRsp_Success. Otherwise it will send SureRsp_Failure with one of the following error codes:

- **SureError_PayloadTooSmall**: Payload was less than 1 byte
- **SureError_PayloadTooLarge**: Payload was more than 1 byte
- **SureError_ValueTooHigh**: The number of retries requested was more than the maximum number of retries allowed

Example: [0x7E, 0x65, 0x01, 0x02]

Sure-Fi Radio Commands (Get Settings)

SureCmd_GetAllSettings (0x70)

Description: Retrieves all the settings that can be sent in SureCmd_SetAllSettings in the same format they would be sent in that command.

Payload: N/A

Response: This command always responds with SureRsp_AllSettings.

Example: [0x7E, 0x70, 0x00]

SureCmd_GetRadioMode (0x71)

Description: Retrieves the current RadioMode as it was sent in SureCmd_SetRadioMode.

Payload: N/A

Response: This command always responds with SureRsp_RadioMode.

Example: [0x7E, 0x71, 0x00]

SureCmd_GetFhssTable (0x72)

Description: Retrieves the current FHSS table as it was sent in SureCmd_SetFhssTable.

Payload: N/A

Response: This command always responds with SureRsp_FhssTable.

Example: [0x7E, 0x72, 0x00]

SureCmd_GetReceiveUID (0x73)

Description: Retrieves the current ReceiveUID as it was sent in SureCmd_SetReceiveUID.

Payload: N/A

Response: This command always responds with SureRsp_ReceiveUID.

Example: [0x7E, 0x73, 0x00]

SureCmd_GetTransmitUID (0x74)

Description: Retrieves the current TransmitUID as it was sent in SureCmd_SetTransmitUID.

Payload: N/A

Response: This command always responds with SureRsp_TransmitUID.

Example: [0x7E, 0x74, 0x00]

SureCmd_GetReceivePacketSize (0x75)

Description: Retrieves the current ReceivePacketSize as it was sent in SureCmd_SetReceivePacketSize.

Payload: N/A

Response: This command always responds with SureRsp_ReceivePacketSize.

Example: [0x7E, 0x75, 0x00]

SureCmd_GetRadioPolarity (0x76)

Description: Retrieves the current RadioPolarity as it was sent in SureCmd_SetRadioPolarity.

Payload: N/A

Response: This command always responds with SureRsp_RadioPolarity.

Example: [0x7E, 0x76, 0x00]

SureCmd_GetTransmitPower (0x77)

Description: Retrieves the current TransmitPower as it was sent in SureCmd_SetTransmitPower.

Payload: N/A

Response: This command always responds with SureRsp_TransmitPower.

Example: [0x7E, 0x77, 0x00]

SureCmd_GetAckData (0x78)

Description: Retrieves the current AckData as it was sent in SureCmd_SetAckData.

Payload: N/A

Response: This command always responds with SureRsp_AckData.

Example: [0x7E, 0x78, 0x00]

SureCmd_GetTableHoppingEnabled (0x79)

Description: Retrieves the current TableHoppingEnabled as it was sent in SureCmd_SetTableHoppingEnabled.

Payload: N/A

Response: This command always responds with SureRsp_TableHoppingEnabled.

Example: [0x7E, 0x79, 0x00]

SureCmd_GetQosConfig (0x80)

Description: Retrieves the current QosConfig as it was sent in SureCmd_SetQosConfig.

Payload: N/A

Response: This command always responds with SureRsp_QosConfig.

Example: [0x7E, 0x80, 0x00]

SureCmd_GetIndications (0x81)

Description: Retrieves the current Indications as it was sent in SureCmd_SetIndications.

Payload: N/A
Response: This command always responds with SureRsp_Indications.

Example: [0x7E, 0x81, 0x00]

SureCmd_GetQuietMode (0x82)

Description: Retrieves the current QuietMode as it was sent in SureCmd_SetQuietMode.

Payload: N/A

Response: This command always responds with SureRsp_QuietMode.

Example: [0x7E, 0x82, 0x00]

SureCmd_GetButtonConfig (0x83)

Description: Retrieves the current ButtonConfig as it was sent in SureCmd_SetButtonConfig.

Payload: N/A

Response: This command always responds with SureRsp_ButtonConfig.

Example: [0x7E, 0x83, 0x00]

SureCmd_GetAcksEnabled (0x84)

Description: Retrieves the current AcksEnabled as it was sent in SureCmd_SetAcksEnabled.

Payload: N/A

Response: This command always responds with SureRsp_AcksEnabled.

Example: [0x7E, 0x84, 0x00]

SureCmd_GetNumRetries (0x85)

Description: Retrieves the current NumRetries as it was sent in SureCmd_SetNumRetries.

Payload: N/A

Response: This command always responds with SureRsp_NumRetries.

Example: [0x7E, 0x85, 0x00]

Sure-Fi Radio Responses (Information)

SureRsp_Status (0x40)

Description: Sent by SureCmd_GetStatus. This response is also sent automatically when a status bit changes and InterruptDriven = 0. See the Status Register section for more information on what each bit in the status means.

Payload: 4 bytes of the status register. The first byte is the StateFlags, followed by OtherFlags, ClearableFlags and then ConfigFlags.

Example: [0x7E, 0x40, 0x04, 81, 00, 00, 12]

SureRsp_IntEnableBits (0x41)

Description: Sent by SureCmd_GetIntEnableBits. The payload matches the same format that was sent in SureCmd_SetIntEnableBits.

Payload: 4 bytes that correspond to the IntEnableBits register. Each bit corresponds to a bit in the Status Register and enables or disables (1 = enabled, 0 = disabled) notifications for that bit when it's value changes.

Example: [0x7E, 0x41, 0x04, 0xBF, 0x01, 0x8F, 0x0F]

SureRsp_ModuleVersion (0x42)

Description: Sent by SureCmd_GetModuleVersion. The payload contains firmware and hardware version information for the module.

Payload: 11 bytes. The structure of these bytes follow the ModuleVersion_t structure defined in sureFiModule.h. The bytes are as follows:

[0]: Firmware Version Major

[1]: Firmware Version Minor

[2-3]: Firmware Version Build (unsigned, Little Endian byte order)

[4]: Hardware Version Major

[5]: Hardware Version Minor

[6-9]: Microcontroller ID (unsigned, Little Endian byte order)

[10]: Microcontroll Revision

Example: [0x7E, 0x42, 0x0B, 0x02, 0x00, 0x42, 0x01, 0x01, 0x01, 0x53, 0xA0, 0x71, 0x07, 0x02]

SureRsp_PacketTimeOnAir (0x43)

Description: Sent by SureCmd_GetPacketTimeOnAir. The payload contains the time that a single packet will take (in milliseconds) with the current radio settings.

Payload: 2-byte unsigned integer (Little Endian byte order) that contains the time for a single packet in milliseconds.

Example: [0x7E, 0x43, 0x02, 0xBD, 0x00]

SureRsp_RandomNumber (0x44)

Description: Sent by SureCmd_GetRandomNumber. The payload contains a random number sources from the radio receiver white noise. The number is meant to be used as a seed for the applications pseudo-random number generator and not as a constant source of evenly distributed random numbers.

Payload: 4-byte random number

Example: [0x7E, 0x44, 0x04, 0xF2, 0x1A, 0x15, 0x35]

SureRsp_Packet (0x45)

Description: Sent by SureCmd_GetPacket. The payload contains the payload that was sent from another module to this one. The payload size should match the ReceivePacketSize - Receive UID length.

Payload: 0 – 62 bytes of radio payload from another module.

Example: [0x7E, 0x45, 0x0A, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99]

SureRsp_AckPacket (0x46)

Description: Sent by SureCmd_GetAckPacket. The payload contains the payload that was loaded into another module using SureCmd_SetAckData and was received as an acknowledgment to a packet that was sent using this module. The payload size should match the ReceivePacketSize - Receive UID length.

Payload: 0 – 62 bytes of radio payload from the last received acknowledgment that contained data

Example: [0x7E, 0x46, 0x0A, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99]

SureRsp_ReceiveInfo (0x47)

Description: Sent by SureCmd_GetReceiveInfo. The receive info contains the RSSI and SNR values for the last successfully received packet. The success byte is included for backwards compatibility.

Payload: 4 bytes. The format of these bytes follows the ReceiveInfo_t structure defined in sureFiModule.h. The bytes are as follows:

[0]: Success/Failure byte (included for backwards compatibility)

[1-2]: The Receive Signal Strength Indicator (RSSI) of the packet. The value is a 16-bit signed integer (Little Endian byte order).

[3]: The Signal to Noise Ratio (SNR) of the packet. The value is an 8-bit signed integer

Example: [0x7E, 0x47, 0x04, 0x01, 0xFF, 0xDE, 0xFE]

SureRsp_TransmitInfo (0x48)

Description: Sent by SureCmd_GetTransmitInfo. The transmit info contains the success/failure byte that tells you whether or not the transmission was successful. It also contains the RSSI and SNR of a received acknowledgment for the packet (if any). The transmit info also contains information about the number of retries that were attempted and the max number the module would attempt for this transmission.

Payload: 7 bytes. The format of these bytes follows the TransmitInfo_t structure defined in sureFiModule.h. The bytes are as follows:

[0] Success/Failure byte (0x01 for success, 0x00 for failure)

[1-2]: The Receive Signal Strength Indicator (RSSI) of the acknowledgment (or 0 if no acknowledgment was received). The value is a 16-bit signed integer (Little Endian byte order).

[3]: The Signal to Noise Ratio (SNR) of the acknowledgment (or 0 if no acknowledgment was received). The value is an 8-bit signed integer

[4]: The number of retries attempted by the module

[5]: The max number of retries the module would attempt for this transmission.

[6]: The length of the data in the acknowledgment (included for backwards compatibility)

Example: [0x7E, 0x48, 0x07, 0x01, 0xFF, 0xDE, 0xFE, 0x00, 0x02, 0x0A]

SureRsp_RegisteredSerial (0x49)

Description: Sent by SureCmd_GetRegisteredSerial. The payload contains a string of ASCII characters that represent the unique serial number of the module. The application can use this string however it pleases but it is mostly provided for internal use at Sure-Fi.

Payload: 1-31 bytes consisting of all ASCII characters. There is no null terminator byte sent in the packet.

Example: [0x7E, 0x49, 0x0E, 0x54, 0x45, 0x31, 0x30, 0x31, 0x34, 0x30, 0x33, 0x30, 0x31, 0x32, 0x35, 0x31, 0x36]

Sure-Fi Responses (Success/Failure)

SureRsp_Success (0x50)

Description: Sent by a wide variety of commands that do not need to respond with information but the application might want to know if they were successful or not. Because this response is used for multiple commands, the command that was sent is put in the payload of this response. In this way you can detect the success of any command or block until a command has been successfully loaded into the module.

Payload: 1 byte containing a copy of the command that was sent.

Example: [0x7E, 0x50, 0x01, 0x37]

SureRsp_Failure (0x51)

Description: Sent by a wide variety of commands that can fail due to improper input or improper timing. Because this response is used for multiple commands, the command that was sent is put in the payload of this response. In many cases there is nothing the application can do about a failure of a command except to maybe reset the state of something. The success and failure responses are provided largely as a way for programmer to debug invalid calls to the module. Therefore having a clear way to see when errors happen will help development for the module immensely.

Payload: 2 bytes. The first byte is a copy of the command that was sent. The second byte is an error code. The error codes that can come in this packet are mentioned by the command that is being sent and the conditions for each code are listed there. However, the error codes used throughout the module all correspond to a code in the following list:

- **SureError_ValueTooLow** (0x01): One of the payload values was below the minimum
- **SureError_ValueTooHigh** (0x02): One of the payload value was above the maximum
- **SureError_InvalidValue** (0x03): One of the payload values did not match any known options
- **SureError_PayloadTooLarge** (0x04): The payload of the command contained too many bytes
- **SureError_PayloadTooSmall** (0x05): The payload of the command did not contain enough bytes
- **SureError_Busy** (0x06): The module is busy and cannot perform the requested action
- **SureError_InvalidSettings** (0x07): The current settings of the module do not pass the criteria for this command
- **SureError_NotFccApproved** (0x08): Performing the requested action would not be within FCC limits
- **SureError_AlreadyStarted** (0x09): The requested action has already begun or cannot be started right now
- **SureError_Unsupported** (0x0A): A response to any command that is unknown to the module.

Example: [0x7E, 0x50, 0x01, 0x37]

SureRsp_UartTimeout (0x52)

Description: Sent when the module receives an ATTN character, followed by a CMD and LENGTH, but then during the transfer of the payload a byte is not received for more than 10ms. This response is mostly helpful for debugging

improperly formatted commands sent by the application. It also keeps the module from waiting indefinitely for a number of bytes when the length is accidentally a larger number than expected.

Payload: 3 bytes. The first byte is a copy of the CMD byte that was received. The second byte is the received LEN byte. The third byte is the number of bytes of payload actually received before the command processing timed out.

Example: [0x7E, 0x52, 0x03, 0x37, 0x0B, 0x0A]

Sure-Fi Radio Responses (Settings)

SureRsp_AllSettings (0x70)

Description: Sent byte SureCmd_GetAllSettings. The format is the same as the payload of SureCmd_SetAllSettings.

Payload: 14 bytes of payload containing the various setting values. See the Module Settings section for more information.

Example: [0x7E, 0x70, 0x0E, 0x02, 0x20, 0x0A, 0x02, 0x1F, 0x01, 0x06, 0x00, 0x00, 0x00, 0x00, 0x12, 0x01, 0x02]

SureRsp_RadioMode (0x71)

Description: Sent byte SureCmd_GetRadioMode. The format is the same as the payload of SureCmd_SetRadioMode.

Payload: 1 or 3 bytes. The first byte is always the radio mode. If the radio mode is custom then the payload will contain 2 more bytes. The next byte is the SpreadingFactor and the last byte is the Bandwidth. See the Radio Modes section for more information. See the Radio Modes section for more information.

Example: [0x7E, 0x71, 0x03, 0x07, 0x04, 0x04]

SureRsp_FhssTable (0x72)

Description: Sent byte SureCmd_GetFhssTable. The format is the same as the payload of SureCmd_SetFhssTable.

Payload: 1-Byte containing the current hopping table that is set

Example: [0x7E, 0x72, 0x01, 0x20]

SureRsp_ReceiveUID (0x73)

Description: Sent byte SureCmd_GetReceiveUID. The format is the same as the payload of SureCmd_SetReceiveUID.

Payload: 0 – 8 bytes containing the current ReceiveUID

Example: [0x7E, 0x73, 0x03, 0x12, 0x34, 0x56]

SureRsp_TransmitUID (0x74)

Description: Sent byte SureCmd_GetTransmitUID. The format is the same as the payload of SureCmd_SetTransmitUID.

Payload: 0 – 8 bytes containing the current TransmitUID

Example: [0x7E, 0x74, 0x03, 0x65, 0x43, 0x21]

SureRsp_ReceivePacketSize (0x75)

Description: Sent byte SureCmd_GetReceivePacketSize. The format is the same as the payload of SureCmd_SetReceivePacketSize.

Payload: 1-byte containing current setting for ReceivePacketSize

Example: [0x7E, 0x75, 0x01, 0x0E]

SureRsp_RadioPolarity (0x76)

Description: Sent byte SureCmd_GetRadioPolarity. The format is the same as the payload of SureCmd_SetRadioPolarity.

Payload: 1-byte containing the current setting for RadioPolarity

Example: [0x7E, 0x76, 0x01, 0x02]

SureRsp_TransmitPower (0x77)

Description: Sent byte SureCmd_GetTransmitPower. The format is the same as the payload of SureCmd_SetTransmitPower.

Payload: 1-byte containing the current setting for TransmitPower

Example: [0x7E, 0x77, 0x01, 0x1F]

SureRsp_AckData (0x78)

Description: Sent byte SureCmd_GetAckData. The format is the same as the payload of SureCmd_SetAckData.

Payload: 0 – 62 bytes containing the current setting for AckData

Example: [0x7E, 0x78, 0x0A, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99]

SureRsp_TableHoppingEnabled (0x79)

Description: Sent byte SureCmd_GetTableHoppingEnabled. The format is the same as the payload of SureCmd_SetTableHoppingEnabled.

Payload: 1 bytes containing the current setting for TableHoppingEnabled. Either a 0x00 for disabled or 0x01 for enabled

Example: [0x7E, 0x79, 0x01, 0x01]

SureRsp_QosConfig (0x80)

Description: Sent byte SureCmd_GetQosConfig. The format is the same as the payload of SureCmd_SetQosConfig.

Payload: 1 byte containing the current setting for QosConfig

Example: [0x7E, 0x80, 0x01, 0x06]

SureRsp_Indications (0x81)

Description: Sent byte SureCmd_GetIndications. The format is the same as the payload of SureCmd_SetIndications.

Payload: 3 bytes. Each nibble contains the current indication set for each of the 6 LEDs to perform. See SureCmd_SetIndications for more information on this format.

Example: [0x7E, 0x81, 0x03, 0x21, 0x43, 0x65]

SureRsp_QuietMode (0x82)

Description: Sent byte SureCmd_GetQuietMode. The format is the same as the payload of SureCmd_SetQuietMode.

Payload: 1 byte containing the current setting for QuietMode. Either 0x00 for disabled or 0x01 for enabled.

Example: [0x7E, 0x82, 0x01, 0x00]

SureRsp_ButtonConfig (0x83)

Description: Sent byte SureCmd_GetButtonConfig. The format is the same as the payload of SureCmd_SetButtonConfig.

Payload: 1 byte containing the current setting for ButtonConfig. See SureCmd_SetButtonConfig for more information.

Example: [0x7E, 0x83, 0x01, 0x12]

SureRsp_AcksEnabled (0x84)

Description: Sent byte SureCmd_GetAcksEnabled. The format is the same as the payload of SureCmd_SetAcksEnabled.

Payload: 1 byte containing the current setting for AcksEnabled. Either 0x00 for disabled or 0x01 for enabled.

Example: [0x7E, 0x84, 0x01, 0x01]

SureRsp_NumRetries (0x85)

Description: Sent byte SureCmd_GetNumRetries. The format is the same as the payload of SureCmd_SetNumRetries.

Payload: 1 byte containing the current setting for NumRetries.

Example: [0x7E, 0x85, 0x01, 0x02]

Bluetooth Commands (Run-Time)

BleCmd_StartAdvertising (0x30)

Description: Tells the Bluetooth to start advertising with the current settings.

Payload: N/A

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_AlreadyStarted**: Advertising has already been started

Example: [0x7C, 0x30, 0x00]

BleCmd_StopAdvertising (0x31)

Description: Tells the Bluetooth to stop advertising.

Payload: N/A

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_NotStarted**: Advertising has already been started

Example: [0x7C, 0x31, 0x00]

BleCmd_CloseConnection (0x32)

Description: Tells the Bluetooth to force disconnect the connected host.

Payload: N/A

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_NotStarted**: The Bluetooth is not connected to a host
- **BleError_AlreadyStarted**: The disconnect process has already been started

Example: [0x7C, 0x32, 0x00]

BleCmd_StartDfuMode (0x33)

Description: Intended for internal use

BleCmd_ReadExmem (0x34)

Description: Intended for internal use

BleCmd_WriteExmem (0x35)

Description: Intended for internal use

BleCmd_ClearExmem (0x36)

Description: Intended for internal use

BleCmd_ClearResetFlag (0x37)

Description: Clears the WasReset flag in the BLE Status Register. See the BLE Status Register section for more information.

Payload: N/A

Response: This command always responds with BleRsp_Success.

Example: [0x7C, 0x37, 0x00]

BleCmd_ClearConnAttemptFlag (0x38)

Description: Clears the ConnectionAttempted flag in the BLE Status Register. See the BLE Status Register section for more information.

Payload: N/A

Response: This command always responds with BleRsp_Success.

Example: [0x7C, 0x38, 0x00]

Bluetooth Commands (Get Information)

BleCmd_GetFirmwareVersion (0x40)

Description: Gets the version number for the firmware on the Bluetooth chip. See the Version Numbers section for more information.

Payload: N/A

Response: This command always responds with BleRsp_FirmwareVersion.

Example: [0x7C, 0x40, 0x00]

BleCmd_GetStatus (0x41)

Description: Gets the status from the Bluetooth module.

Payload: N/A

Response: This command always responds with BleRsp_Status.

Example: [0x7C, 0x41, 0x00]

BleCmd_GetMacAddress (0x42)

Description: Gets the MAC address of the Bluetooth module.

Payload: N/A

Response: This command always responds with BleRsp_MacAddress.

Example: [0x7C, 0x42, 0x00]

Bluetooth Commands (Set Settings)

BleCmd_SetStatusUpdateBits (0x50)

Description: Write the StatusUpdateBits register which controls which bits the Bluetooth monitors for changes. If a change occurs in the BLE Status Register on any of the bits that you have set to 1 the Bluetooth send BleRsp_Status automatically.

Payload: 1 byte corresponding to the 1 byte in the BLE Status Register. Each bit can be enabled (1) or disabled (0).

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_PayloadTooSmall**: The command contained less than 1 byte payload
- **BleError_PayloadTooLarge**: The command contained more than 1 byte payload

Example: [0x7C, 0x50, 0x01, 0xFF]

BleCmd_SetAdvertisingData (0x51)

Description: Sets the data to be placed in the Manufacturer's Data when the Bluetooth sends out advertisements. See the BLE Advertising section for more information.

Payload: 0 - 19 bytes. The larger the advertising data is the smaller the advertised short name becomes.

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_PayloadTooLarge**: The command contained more than 19 bytes payload

Example: [0x7C, 0x51, 0x0C, 0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x20, 0x57, 0x6F, 0x72, 0x6C, 0x64, 0x21]

BleCmd_SetAdvertisingName (0x52)

Description: Sets the Full Name for the Bluetooth chip to send out in it's advertisements. Depending on how many bytes were passed to BleCmd_SetAdvertisingData the full name might not fit in the advertising packet. If this is the case a partial name will be advertised and the full name can be obtained by the host upon connection. See the BLE Advertising section for more information.

Payload: 0-22 bytes ASCII characters.

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_PayloadTooLarge**: The command contained more than 19 bytes payload

Example: [0x7C, 0x52, 0x10, 0x4C, 0x6F, 0x6E, 0x67, 0x20, 0x52, 0x61, 0x6E, 0x67, 0x65, 0x20, 0x52, 0x61, 0x64, 0x69, 0x6F]

BleCmd_SetTemporaryData (0x53)

Description: Passes some data to be saved in RAM on the Bluetooth chip for retrieval by the application later. This is only really useful in a handful of scenarios when the application is not able to keep information across logical barriers (like a software reset) and needs something external to temporarily save information. Note that this data is not guaranteed to persist since the Bluetooth could be reset during this time and this data is only stored in RAM on the Bluetooth chip.

Payload: 0-255 bytes of data.

Response: This command always responds with BleRsp_Success.

Example: [0x7C, 0x53, 0x0E, 0x48, 0x65, 0x6C, 0x6F, 0x20, 0x49, 0x6E, 0x74, 0x65, 0x72, 0x6E, 0x65, 0x74]

BleCmd_SetGpioConfiguration (0x54)

Description: Sets the direction and other configuration options for one of the extra GPIO that is connected to the Bluetooth chip. See the Bluetooth GPIO section for more information.

Payload: 3 bytes. The first byte should be the GPIO number. The second byte should be either BleGpioDir_Output (0x00) or BleGpioDir_Input (0x01). If configuring the pin as an output then the third byte should be the initial output value for the pin (0x00 for low and 0x01 for high). If configuring the pin as an input then the third byte should be either BleGpioPull_None (0x00), BleGpioPull_Up (0x01), or BleGpioPull_Down (0x02).

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_PayloadTooSmall**: The command contained less than 3 bytes payload
- **BleError_PayloadTooLarge**: The command contained more than 3 bytes payload
- **BleError_InvalidValue**: The direction byte was not BleGpioDir_Output or BleGpioDir_Input
- **BleError_Unsupported**: A GPIO with that number does not exist

Example: [0x7C, 0x54, 0x03, 0x19, 0x01, 0x02]

BleCmd_SetGpioValue (0x55)

Description: Sets the output value for a GPIO that has been configured as an output.

Payload: 2 bytes. The first byte should be the GPIO number. The second byte should be the value to output (0x00 for low, 0x01 for high).

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_PayloadTooSmall**: The command contained less than 2 bytes payload
- **BleError_PayloadTooLarge**: The command contained more than 2 bytes payload
- **BleError_InvalidSettings**: The GPIO has not been configured as an output
- **BleError_Unsupported**: A GPIO with that number does not exist

Example: [0x7C, 0x55, 0x02, 0x19, 0x01]

BleCmd_SetGpioUpdateEnabled (0x56)

Description: Enables or disables automatic updates for a GPIO that has been configured as an input. If updates are enabled for a GPIO then changes to the input value will automatically be sent to the application through a BleRsp_GpioValue.

Payload: 2 bytes. The first byte should be the GPIO number. The second byte should be 0x01 to enable updates or 0x00 to disable updates

Response: If successful this command will respond with BleRsp_Success. If enabling AutoUpdates this command also sends BleRsp_GpioValue immediately. If this command fails it will send BleRsp_Failure with one of the following error codes:

- **BleError_PayloadTooSmall**: The command contained less than 2 bytes payload
- **BleError_PayloadTooLarge**: The command contained more than 2 bytes payload
- **BleError_InvalidSettings**: The GPIO has not been configured as an input
- **BleError_Unsupported**: A GPIO with that number does not exist

Example: [0x7C, 0x56, 0x02, 0x19, 0x01]

BleCmd_SetRejectConnections (0x57)

Description: Sets whether or not new connections from a host device are allowed. When RejectConnections is enabled any incoming connections will be immediately closed. When this happens the ConnectionAttempted flag gets set in the BLE Status Register.

Payload: 1 byte with either 0x00 for disabled or 0x01 for enabled.

Response: If successful this command will respond with BleRsp_Success. Otherwise it will send BleRsp_Failure with one of the following error codes:

- **BleError_PayloadTooSmall**: The command contained less than 3 bytes payload
- **BleError_PayloadTooLarge**: The command contained more than 3 bytes payload

Example: [0x7C, 0x57, 0x01, 0x01]

Bluetooth Commands (Get Settings)

BleCmd_GetStatusUpdateBits (0x70)

Description: Gets the value of the StatusUpdateBits register that was set in BleCmd_SetStatusUpdateBits.

Payload: N/A

Response: This command always responds with BleRsp_StatusUpdateBits.

Example: [0x7C, 0x70, 0x00]

BleCmd_GetAdvertisingData (0x71)

Description: Gets the advertising data that was set in BleCmd_SetAdvertisingData.

Payload: N/A

Response: This command always responds with BleRsp_AdvertisingData.

Example: [0x7C, 0x71, 0x00]

BleCmd_GetAdvertisingName (0x72)

Description: Gets the advertising name that was set in BleCmd_SetAdvertisingName.

Payload: N/A

Response: This command always responds with BleRsp_AdvertisingName.

Example: [0x7C, 0x72, 0x00]

BleCmd_GetTemporaryData (0x73)

Description: Gets the temporary data that was set in BleCmd_SetTemporaryData.

Payload: N/A

Response: This command always responds with BleRsp_TemporaryData.

Example: [0x7C, 0x73, 0x00]

BleCmd_GetGpioConfiguration (0x74)

Description: Gets the configuration of one of the GPIO that was set in BleCmd_SetGpioConfiguration.

Payload: 1 byte GPIO number

Response: If this command succeeds it will respond with BleRsp_GpioConfiguration. Otherwise it will respond with BleRsp_Failure with one of the following error codes:

- **BleError_Unsupported**: A GPIO with that number does not exist

Example: [0x7C, 0x74, 0x00]

BleCmd_GetGpioValue (0x75)

Description: Gets the value of one of the previously configured GPIO. If the GPIO is configured as an input than this is the reading of the input. If the GPIO is configured as and output then this command is used to get the value that was previously written using BleCmd_SetGpioValue.

Payload: 1 byte GPIO number

Response: If this command succeeds it will respond with BleRsp_GpioValue. Otherwise it will respond with BleRsp_Failure with one of the following error codes:

- **BleError_Unsupported**: A GPIO with that number does not exist

Example: [0x7C, 0x75, 0x00]

BleCmd_GetGpioUpdateEnabled (0x76)

Description: Gets the current setting for auto updates for a single GPIO that was set in BleCmd_SetGpioUpdateEnabled.

Payload: 1 byte GPIO number

Response: If this command succeeds it will respond with BleRsp_GpioUpdateEnabled. Otherwise it will respond with BleRsp_Failure with one of the following error codes:

- **BleError_Unsupported**: A GPIO with that number does not exist

Example: [0x7C, 0x76, 0x00]

BleCmd_GetRejectConnections (0x77)

Description: Gets the current setting for rejections enabled that was set in BleCmd_SetRejectConnections.

Payload: N/A

Response: This command always responds with BleRsp_RejectConnections.

Example: [0x7C, 0x77, 0x00]

Bluetooth Responses (Run-Time)

BleRsp_DfuNeedAdvData (0x30)

Description: Intended for internal use

BleRsp_ExmemData (0x31)

Description: Intended for internal use

Bluetooth Responses (Information)

BleRsp_FirmwareVersion (0x40)

Description: Sent by BleCmd_GetFirmwareVersion. The format is described in the Version Numbers section.

Payload: 4 bytes. 1 byte major, 1 byte minor, 2 bytes build number (little-endian)

Example: [0x7C, 0x40, 0x04, 0x02, 0x01, 0xD2, 0x01]

BleRsp_Status (0x41)

Description: Sent by BleCmd_GetStatus or automatically when updates are enabled on bits that change. See the BLE Status Register section and BleCmd_SetStatusUpdateBits for more information.

Payload: 1 byte value of the BLE Status Register.

Example: [0x7C, 0x41, 0x01, 0x04]

BleRsp_MacAddress (0x42)

Description: Sent by BleCmd_GetMacAddress.

Payload: 6 byte mac address in little-endian order. This is the order that they are read out of the Bluetooth chip but is often the opposite order to how Bluetooth MAC addresses are normally displayed. For example the address in the example response below would normally be displayed as F4:39:AC:9B:3D:C2.

Example: [0x7C, 0x42, 0x06, 0xC2, 0x3D, 0x9B, 0xAC, 0x39, 0xF4]

Bluetooth Responses (Success/Failure)

BleRsp_Success (0x50)

Description: Sent by a wide variety of commands that do not need to respond with information but the application might want to know if they were successful or not. Because this response is used for multiple commands, the command that was sent is put in the payload of this response. In this way you can detect the success of any command or block until a command has been successfully loaded into the Bluetooth chip.

Payload: 1 byte copy of the command that was sent

Example: [0x7C, 0x50, 0x01, 0x30]

BleRsp_Failure (0x51)

Description: Sent by a wide variety of commands that can fail due to improper input or improper timing. Because this response is used for multiple commands, the command that was sent is put in the payload of this response. In many cases there is nothing the application can do about a failure of a command except to maybe reset the state of something. The success and failure responses are provided largely as a way for programmer to debug invalid calls to the Bluetooth chip. Therefore having a clear way to see when errors happen will help development for the module immensely.

Payload: 2 bytes. The first byte is a copy of the command that was sent. The second byte is an error code. The error codes that can come in this packet are mentioned by the command that is being sent and the conditions for each code are listed there. However, the error codes used throughout the module all correspond to a code in the following list:

- **BleError_ValueTooLow** (0x01): One of the payload values was below the minimum
- **BleError_ValueTooHigh** (0x02): One of the payload value was above the maximum
- **BleError_InvalidValue** (0x03): One of the payload values did not match any known options
- **BleError_PayloadTooLarge** (0x04): The payload of the command contained too many bytes
- **BleError_PayloadTooSmall** (0x05): The payload of the command did not contain enough bytes
- **BleError_Busy** (0x06): The module is busy and cannot perform the requested action
- **BleError_InvalidSettings** (0x07): The current settings of the module do not pass the criteria for this command
- **BleError_NotFccApproved** (0x08): Performing the requested action would not be within FCC limits
- **BleError_AlreadyStarted** (0x09): The requested action has already begun or cannot be started right now
- **BleError_Unsupported** (0x0A): A response to any command that is unknown to the module.
- **BleError_NotStarted** (0x0B): The requested action has already been stopped or never begun.

Example: [0x7C, 0x51, 0x02, 0x30, 0x09]

BleRsp_UartTimeout (0x52)

Description: Sent when the Bluetooth chip receives an ATTN character, followed by a CMD and LENGTH, but then during the transfer of the payload a byte is not received for more than 10ms. This response is mostly helpful for debugging improperly formatted commands sent by the application. It also keeps the Bluetooth chip from waiting indefinitely for a number of bytes when the length is accidentally a larger number than expected.

Payload: 3 bytes. The first byte is a copy of the CMD byte that was received. The second byte is the received LEN byte. The third byte is the number of bytes of payload actually received before the command processing timed out.

Example: [0x7C, 0x52, 0x03, 0x51, 0x0A, 0x08]

BleRsp_BleWriteTimeout (0x53)

Description: Similar to BleRsp_UartTimeout this command goes to the application when the command processing timed out on data received through the TxData characteristic from the connected host. This can happen when the host tries to write an improperly formatted command. The timeout for writing to the Bluetooth characteristic is 2000ms.

Payload: 3 bytes. The first byte is a copy of the CMD byte that was received. The second byte is the received LEN byte. The third byte is the number of bytes of payload actually received before the command processing timed out.

Example: [0x7C, 0x53, 0x03, 0x10, 0x10, 0x0E]

Bluetooth Responses (Settings)

BleRsp_StatusUpdateBits (0x70)

Description: Sent by BleCmd_GetStatusUpdateBits. The format is the same as BleCmd_SetStatusUpdateBits.

Payload: 1 byte value of the StatusUpdate register.

Example: [0x7C, 0x70, 0x01, 0xFF]

BleRsp_AdvertisingData (0x71)

Description: Sent by BleCmd_GetAdvertisingData. The format is the same as BleCmd_SetAdvertisingData.

Payload: 0-19 bytes containing the advertising data.

Example: [0x7C, 0x71, 0x02, 0x12, 0x34]

BleRsp_AdvertisingName (0x72)

Description: Sent by BleCmd_GetAdvertisingName. The format is the same as BleCmd_SetAdvertisingName.

Payload: 0-22 bytes containing the ASCII advertising name

Example: [0x7C, 0x72, 0x0E, 0x53, 0x75, 0x72, 0x65, 0x2D, 0x46, 0x69, 0x20, 0x4D, 0x6F, 0x64, 0x75, 0x6C, 0x65]

BleRsp_TemporaryData (0x73)

Description: Sent by BleCmd_GetTemporaryData. The format is the same as BleCmd_SetTemporaryData.

Payload: 0-255 bytes containing the temporary data

Example: [0x7C, 0x73, 0x00]

BleRsp_GpioConfiguration (0x74)

Description: Sent by BleCmd_GetGpioConfiguration. The format is the same as BleCmd_SetGpioConfiguration.

Payload: 3 bytes. 1 byte for GPIO Number. 1 byte for Input (0x01) / Output (0x00) configuration. 1 byte for either Pull-Up/Down configuration in input mode or the current output value for output mode

Example: [0x7C, 0x74, 0x03, 0x19, 0x01, 0x02]

BleRsp_GpioValue (0x75)

Description: Sent by BleCmd_GetGpioValue. When the GPIO is configured as an output this responds with the data written using BleCmd_SetGpioValue. When the GPIO is configured as an input this responds with the current value of the pin.

Payload: 2 bytes. 1 byte for the GPIO Number. 1 byte for value of the pin (0x00 low, 0x01 high)

Example: [0x7C, 0x75, 0x01, 0x19, 0x00]

BleRsp_GpioUpdateEnabled (0x76)

Description: Sent by BleCmd_GetGpioUpdateEnabled. The format is the same as BleCmd_SetGpioUpdateEnabled.

Payload: 2 bytes. 1 byte for the GPIO Number. 1 byte either 0x00 for updates disabled or 0x01 for updates enabled.

Example: [0x7C, 0x76, 0x02, 0x19, 0x01]

BleRsp_RejectConnections (0x77)

Description: Sent by BleCmd_GetRejectConnections. The format is the same as BleCmd_SetRejectConnections.

Payload: 1 byte either 0x00 for RejectConnections disabled or 0x01 for RejectConnections enabled

Example: [0x7C, 0x77, 0x01, 0x00]

Sure-Fi Radio Structures

Radio Status Register

The Radio Status Register is the backbone of communication between the module and the application. Any event that the application should be aware of is indicated in the Radio Status.

The Radio Status Register contains 4 bytes of data with each bit indicating different information about the state of the Sure-Fi Radio. The status register can be read at any time using SureCmd_GetStatus. If the InterruptDriven bit in the config byte of the Status Register is 0 then the status will also be sent automatically by the radio when a change occurs on one of the bits that has the corresponding IntEnable bit set (See SureCmd_SetIntEnableBits). For best use it is recommended that the application keep a copy of the status locally in order to perform an XOR with any new status that is received. In this way the application can determine which bits have changed since the last received status and perform actions accordingly.

Each byte in the status is a grouping of bits that have an overall purpose. The first byte is the StateFlags byte. It contains bits that represent the overall state of the Sure-Fi radio. The second byte is the OtherFlags byte. It contains other flags that are read only and can be helpful information for the application to know. The third byte is the ClearableFlags byte. It contains flags that will only ever be set by the Sure-Fi radio and must be cleared by the application (either through SureCmd_ClearFlags or setting the AutoClearFlags bit). The fourth and last byte is the ConfigFlags byte. It contains bits that control the behavior of the module and they can be written using SureCmd_WriteConfig.

Name	Bit Value	Description
StateFlags Byte		
RadioState	0x0F	These four bits are combined to represent the state of the Sure-Fi Radio. The value will be one of the following: • **Initializing** (0b0000): The radio is not yet ready to perform any actions • **Receiving** (0b0001): The radio is awaiting either instruction from the application or a packet to be received • **Transmitting** (0b0010): The radio is currently transmitting a packet. • **WaitingForAck** (0b0011): The radio has finished transmitting and is waiting for an acknowledgment that the packet was transferred successfully. • **Acknowledging** (0b0100): The radio is acknowledging a received packet. • **Sleeping** (0b0101): The radio is in sleep mode
Busy	0x10	The radio is busy and cannot accept any command that requires the radio to transmit a packet. This value is also replicated on the GPIO0 if InterruptDriven = 0 so that the application can hold off transmitting commands more easily
ChangingTables	0x20	The radio is waiting to change to the next table. This only happens when TableHopping is enabled. The delay is caused by a synchronization of both sides to move to the next table. The busy bit will be set during this time and no packets can be transmitted.
RxInProgress	0x40	There is a packet being received on the radio. The busy bit will be set during this time. After RxInProgress goes back to zero there

		may not always be a packet ready to read. There are various reasons why the packet may not have been received successfully but most often you will the ChecksumError bit get set if the RxPacketReady bit does not get set.
OnBaseTable	0x80	This bit is set as long as we are on the base FHSS table (the one set in SureCmd_SetFhssTable). It will only be cleared if TableHopping is enabled and a table change has been triggered. See the Table Hopping section for more information about why this information is useful for the application to know.
OtherFlags Byte		
DoingLightshow	0x01	The QOS Lightshow is in progress. This lightshow is activated by sending SureCmd_QosLightshow.
ShowingQos	0x02	The Quality of Service indication is being shown. This is either activated by SureCmd_ShowQualityOfService or automatically depending on the QosConfig.
ButtonDown	0x04	This bit is a direct correlation to the state of the button on the module. It will be 1 while the button is down and 0 while the button is not down.
EncryptionActive	0x08	The encryption process is ongoing. See the Encryption section for more information on the encryption process.
SettingsPending	0x10	Settings that have been processed by the module are waiting to be applied. Settings are only applied when the radio is in Receive mode. This bit is provided largely for debug purposes and requires no action from the application.
Reserved	0xE0	These bits are reserved for later use
ClearableFlags Byte		
WasReset	0x01	This bit is set when the module goes through the startup process. Caused by either a PowerOn or SoftwareReset triggered by SureCmd_Reset. This bit will most often only get set at the same time the application powers on. However, there are certain scenarios the module might get reset when the application board does not and in these scenarios the application should take care to reset the module to the settings it needs to perform it's operations.
TransmitFinished	0x02	This bit is set when a transmission process has completed. The application should call SureCmd_GetTransmitInfo.
RxPacketReady	0x04	This bit is set when a packet has been received successfully. The application should call SureCmd_GetPacket and SureCmd_GetReceiveInfo.
AckPacketReady	0x08	This bit is set when an acknowledgment was received that has acknowledgment data. Most often this will be at the same time as the TransmitFinished flag is set. The application should call SureCmd_GetAckPacket.
ChecksumError	0x10	This bit is set when a packet was received but the data was corrupted. This can happen for various reasons. Most often it is caused by collisions between two different transmitters at the same time or by a very low signal strength where the noise floor overpowers the signal.
EncryptionRekey	0x20	An encryption rekey process was triggered by 3 failed transmissions in a row. This feature can be disabled using the

		AutoRekey bit in the ConfigFlags byte.
ButtonPressed	0x40	This bit is set when the button on the module was pressed and released before the ButtonHeld flag was triggered.
ButtonHeld	0x80	This bit is set when the button on the module was pressed and held for longer than the time set by SureCmd_SetButtonConfig. The default setting is 1 second. This bit gets set before the button is released.
ConfigFlags Byte		
InterruptDriven	0x01	This bit controls the way in which the application receives updates to the status register. When this bit is one the GPIO0 pin is used as an interrupt pin for the application. Whenever a bit changes that has interrupts enable the GPIO0 pin will be asserted until the application calls SureCmd_GetStatus. If this bit is 0 then the module sends SureRsp_Status updates automatically when changes occur and the GPIO0 pin is freed up to be used as a busy indication. The pin is asserted when the Busy bit is 1. The InterruptDriven bit is set to 0 on startup and SureRsp_Status is always sent on startup.
AutoClearFlags	0x02	If this bit is set to 1 then the ClearableFlags byte will get cleared automatically whenever the status has been read. This includes automatic updates when InterruptDriven = 0. This relieves the application of the need to send SureCmd_ClearFlags every time one of the ClearableFlags is set. This bit is set to 0 by default.
RxLedMode	0x04	This bit changes the way the Receive LED (the Red LED) works on the module. When this bit is 0 (the default) the receive LED goes on for a set period of time after a valid packet is received or after a valid acknowledgment with data is received. When this bit is 1 the receive LED goes on during the receive of any packet (valid or invalid). This is often confusing to the user but can be helpful for debugging various problems.
TxLedMode	0x08	This bit changes the way the Transmit LED (the Yellow LED) works on the module. When this bit is 0 (the default) the transmit LED goes on during a whole transmission process (activated by SureCmd_TransmitData, SureCmd_StartEncryption or a button press when ButtonConfig is set accordingly). This transmission process includes the time spent to transmit or retransmit (retry) a packet as well as waiting for acknowledgments. If this bit is set to 1 then the Transmit LED goes on during any packet transmission, including retries and acknowledgments. This is often confusing to the user but can be helpful for debugging various problems.
AutoRekey	0x10	This bit is used to enable or disable the automatic Encryption Rekey Process. The Rekey Process is activated whenever there are 3 failed transmissions in a row. If this happens then the module automatically transmits a packet similar to the one sent in StartEncryption to make sure the other module did not fall out of step with the encryption. This feature is included for backwards compatibility and should be unnecessary with the newest version of the module.

Sure-Fi Radio Modes

The Radio Modes are meant to provide the application with an easy choice regarding which Spreading Factor and Bandwidth to use. There are 6 radio modes available on the Sure-Fi Module along with the option to choose a Custom radio mode. The various modes allow the application to make a trade-off between data throughput and range with RadioMode_1 representing the longest range/lowest data rate and RadioMode_6 representing the shortest range/highest data rate. The RadioMode can be set at any time as long as the radio is idle but the RadioMode of the Receiving and Transmitting sides must match in order for any transactions to succeed. Some RadioModes use multiple Spreading Factors and Bandwidths in order to decrease collisions between systems. This choice is made based off the hopping table choice made by the application in SureCmd_SetFhssTable.

The actual data rate depends on many factors including what RadioMode is chosen, what hopping table is used, whether or not acknowledgments are enabled, whether table hopping is enabled, the number of retries chosen, and how many packets actually succeed during the retry process. The data rate given below is based off the best possible scenario with the least amount of overhead possible on the Sure-Fi Module and assuming all packets succeed. Actual data rates in most applications will be lower.

RadioMode	HEX Value	Utilized Spreading Factors and Bandwidths	Range	Max Data Rate
RadioMode_1	0x01	SF12 @250kHz SF11 @125kHz	Longest	225 bps
RadioMode_2	0x02	SF11 @250kHz SF10 @125kHz SF9 @62.5kHz	Long	450 bps
RadioMode_3	0x03	SF10 @250kHz SF9 @125kHz SF8 @62.5kHz	Medium	805 bps
RadioMode_4	0x04	SF9 @250kHz SF8 @125kHz SF7 @62.5kHz	Average	1,456 bps
RadioMode_5	0x05	SF8 @250kHz	Short	2,546 bps
RadioMode_6	0x06	SF7 @250kHz	Shortest	4,207 bps
RadioMode_Custom	0x07	-	-	-

Table Hopping

Table Hopping is a feature provided by the Sure-Fi Module to help data throughput in applications that need to send multiple packets in a row. The feature is disabled by default and can be enabled using SureCmd_SetTableHoppingEnabled.

The module has support for 72 different hopping tables which can be chosen using SureCmd_SetFhssTable. (If a value larger than 71 is given then the actual hopping table used is FhssChoice%72 where % is the modulus operator). Even though each table contains all 72 frequencies each one must start with it's preamble on the same frequency each time. This preamble is the major limitation on the number of packets that can be transmitted within a period of time due to FCC limitations.

With Table Hopping disabled, the module will automatically handle holding off the application using the Busy bit in the Status Register when this limit has been reached. This ensures FCC compliance but is not ideal for applications that need to send their data as fast as possible.

If table hopping is enabled the module will automatically handle moving between different hopping tables whenever this limit is close to being reached on the current preamble frequency. This is compatible with having multiple retries enabled. However, a higher number of retries results in more waiting during the table change and therefore less throughput. The Table Hopping mechanism is also not perfect and can get out of sync if all tries of a packet fail. This will result in one module continuing to transmit on the new table while the other module did not change tables. This will get corrected when the FCC time period is passed (10-20 seconds depending on the Spreading Factor used) and the module goes back to the base table. Because of this it is best to have at least one retry enabled in case a single packet fails.

For debug purposes two bits are provided in the Status Register that give you some insight into this process. The ChangingTables bit will be 1 whenever the table change process is in progress. During this time the busy bit will be 1 as well and the application will have to wait for the process to complete before sending SureCmd_TransmitData. The OnBaseTable bit will be 1 when the module has not changed tables. However, after a table change has occurred and during the whole time the module is on a table other than the table that was chosen in SureCmd_SetFhssTable this bit will be set to 1. If the table change was unsuccessful then all packets will fail until the module returns back to the base table at the end of the time period.

Quality of Service Lights

The Quality of Service Indication is an indication that can be shown on the module's LEDs to indicate to the user how good the signal strength is on received packets. This indication can be configured to show automatically using SureCmd_SetQosConfig or can be shown at any time using SureCmd_ShowQualityOfService. In both of these cases the quality of service indication overrides the indications from SureCmd_SetIndications for a period of 1 second. Depending on the RSSI value of the last received packet or acknowledgment the module will automatically calculate a value from 1 to 6 and light up that many LEDs on the module starting from the bottom-most LED. The number of LEDs to be shown for any RSSI value can be found using the table below.

Number of LEDs	RSSI Range
1	less than -124 dB
2	-124 dB to -113 dB
3	-114 dB to -103 dB
4	-104 dB to -93 dB
5	-94 dB to -83 dB
6	more than -84 dB

Version Numbers

The Sure-Fi Radio's version is 13 bytes and is defined as ModuleVersion_t in the header file. It is a combination of 3 separate pieces of information: the firmware version, the hardware version, and the micro controller version. The firmware version is 4 bytes total: 1 byte for major, 1 byte for minor, and 2 bytes for build number (little endian). Most application only ever need to look at the firmware version. The hardware version is 2 bytes total: 1 byte for major, and 1 byte for minor. The micro controller version is 5 bytes total: 4 bytes for device ID (little endian), and 1 byte for revision number. The micro controller version corresponds to version numbers designated by Microchip for their microprocessors.

The Bluetooth version is 4 bytes and is defined as FullVersion_t in the header file. This firmware version is the same structure as the Sure-Fi Radio's firmware version: 1 byte major, 1 byte minor, and 2 bytes build number (little endian).

BLE Status Register

The Bluetooth status register is 1 byte (8 bits) in size and contains status information about the Bluetooth chip on the Sure-Fi Module. The functionality of the Bluetooth status is very similar to that of the Radio Status Register when the InterruptMode bit is 0. Changes in the status register are automatically sent to the application in a BleRsp_Status updates on that bit were enabled using BleCmd_SetStatusUpdateBits. This structure is also defined in the header as BleStatus_t. The bits and their descriptions are given below:

Name	Bit Value	Description
WasReset	0x01	Indicates that the Bluetooth chip has been reset. This flag can be cleared using BleCmd_ClearResetFlag.
Connected	0x02	Indicates whether or not a host device is currently connected. (1 for connected, 0 for disconnected)
Advertising	0x04	Indicates whether or not the Bluetooth is advertising (1 for advertising, 0 for not advertising)
InDfuMode	0x08	Indicates that the Bluetooth chip is in Device Firmware Update mode. A third party application should never see this.
SureFiTxInProgress	0x10	Indicates whether or not the Sure-Fi Radio is transmitting. This is important for the Bluetooth since it cannot be connected or advertise while the Sure-Fi Radio is transmitting a packet. If the application sees this bit go high at the same time that the Connect bit goes to 0 then the host device was disconnected by a Sure-Fi Transmit.
ConnectionAttempted	0x20	This bit gets set whenever a host device tries to connect while RejectConnections has been enabled using BleCmd_SetRejectConnections. This bit must be cleared by the application using BleCmd_ClearConnAttemptFlag.

BLE Advertising

The Bluetooth chip must be configured and told to start advertising before any host will be able to discover or connect to it. It is recommended that the application send BleCmd_SetAdvertisingName, BleCmd_SetAdvertisingData, and BleCmd_StartAdvertising in order to get the Bluetooth to start advertising. If the advertising name or data is not set before calling BleCmd_StartAdvertising then the default values of "Sure-Fi Module" and [0x12, 0x34] are used. The advertising name and data can be changed at any time, even when the Bluetooth has started advertising. Advertising can be stopped and started whenever the application would like.

The advertising data is placed under manufacturer specific data in the Bluetooth advertising packet with the 2 byte company identifier 0xFFFF. The advertising data is prioritized over the advertising name when the advertising packet cannot contain both. With each new byte of advertising data the max advertising name length gets shorter by one byte. The advertising name can still be set to something longer than what can be advertised but the full name will only be able to be obtained when the host connects. If the advertising packet is not able to contain the whole name it as advertised as a short name. There are total of 20 bytes available in the advertising packet, therefore if the advertising data is set to the maximum length of 19 bytes there will only be a single byte left for the first character in the advertising name.

No services or UUIDs are advertised in the packet. The Module Service can only be discovered once a host has connected. Once a host connects to the Bluetooth chip advertisements will cease to go out but they are still enabled and if left enabled the Bluetooth chip will start advertising again automatically when the host disconnects.

In order to comply with FCC standards for collocated transmitters the Bluetooth chip is not able to be connected or advertising while the Sure-Fi Radio performs a transmission. If the Sure-Fi Radio starts transmitting the Bluetooth will

automatically disconnect any host and temporarily postpone any advertisements until the transmission is complete. For this reason it is recommended that the application enable quiet mode on the radio while a host is connected using SureCmd_SetQuietMode and refrain from sending SureCmd_TransmitData if the application wants the Bluetooth connection to stay alive.

Module Service

The Bluetooth chip is designed to be as easy to use as possible for basic communication over Bluetooth Low Energy. The firmware automatically sets up a custom "Module Service" which allows any host device to connect, send and receive commands, and read the advertising data that was set by the application. The command format follows the same structure as all other UART commands going to and from the module and uses the default ATTN character (0x7E). Bluetooth Bonding is not supported by the firmware, therefore it is recommended that the application take appropriate steps to ensure that the connected host is validated in some way and that sensitive information is encrypted before sending across the "Module Service".

The "Module Service" is a custom service with the UUID: E8BF000A-0EC5-2536-2143-2D155783CE78. The service contains 4 characteristics with the similar UUID but with the 3rd and 4th byte being a unique short UUID:

Name	Short UUID	Attributes	Description
Rx Data	000B	Read, Indicate, Long Read	This characteristic will be filled with commands received from the application intended for the Bluetooth host. Each command received is indicated separately which allows the application to be notified of every new command, even if the new command matches the data that was already in the characteristic. It is recommended that the host always subscribe to indications on this characteristic. The data in this characteristic will always be in the UART command format (ATTN, CMD, LEN, ...).
Tx Data	000C	Write, Write Without Response, Long Write	This characteristic used by the host device to send commands to the application. All data written the characteristic is put onto an internal queue and handled like any other data bus. Then each command following the UART command format (ATTN, CMD, LEN, ...) is popped off and sent over the UART to the application in the order that they were received. This means that multiple commands can be written in the same Bluetooth write operation or one command can be split into multiple write operations. However, since the characteristic supports long writes it is usually most efficient to send large commands (or group of commands) in a single write call. The max write length is 258 bytes.
Radio Status	000D	Read, Indicate	This characteristic contains a 1 byte boolean (0x00, or 0x01) indicating the "Busy" status of the Sure-Fi Radio. This can be useful for applications who wish to control the radio through the Bluetooth without an application board.
Advertising Data	000E	Read, Indicate	This characteristic contains a copy of the advertising data that goes in all advertising packets. This data can be set by the application and updated at any time. This characteristic can also be subscribed to so changes are immediately indicated to the host. See BleCmd_SetAdvertisingData.

Bluetooth GPIO

There are 6 test points (TP) connected to the Bluetooth chip which are available for the application to use for any purpose. These test points are label in the Block Diagram as TP3, TP4, TP5, TP6, TP25, and TP28. These are also how they are labeled on the module and how they are referred to in all GPIO commands. This is their "GPIO Number" (TP3 is 3, TP4, is 4, TP25 is 25, etc.). The numbers are also defined in the header file as BleGpio_TP3, BleGpio_TP4, etc. These are 3.3V logic pins directly connected to the Bluetooth micro controller. For full electrical characteristics see the Sure-Fi Module Datasheet. Each GPIO can be configured separately as an input or an output.

In input mode the application can enable internal Pull-Up or Pull-Down resistors when configuring the pin using BleCmd_SetGpioConfiguration. Automatic updates for each GPIO can also be enabled using BleCmd_SetGpioUpdateEnabled. When this updates are enabled the input is monitored by the Bluetooth firmware and changes are reported to the application automatically with BleRsp_GpioValue responses. If the pin is configured as an output then it's updates are automatically disabled. The value of the pin can also be read at any time by the application using BleCmd_GetGpioValue.

In output mode the application can set the output value using BleCmd_SetGpioValue. This value can also be read by the application using BleCmd_GetGpioValue.