SNIC Interface Specification

Version: 1.7

October 17, 2012

Release Record

Version Number	Release Date	Comments
Version 0.0	10/6/2011	• Initial Draft.
Version 1.0	1/26/2012	• Initial release
Version 1.2	3/5/2012	• Added WiFi network status indication and data ACK configuration
Version 1.3	3/21/2012	• Add WiFi get RSSI, get status, and scan
Version 1.4	5/4/2012	• Add soft AP and NVM related changes
Version 1.5	7/24/2012	• Restore and AP on off changes
Version 1.6	8/8/2012	• IO and Peripheral API
Version 1.7	10/17/12	• Add SPI interface

THE TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. Introduction

SN8200 is a complete low power embedded wireless solution to address the connectivity demand in M2M applications. It integrates an ARM Cortex M3 micro-controller, Wi-Fi BB/MAC/RF IC, RF front end, flash memory, clock, and on-board antenna into a small form factor module. The SN8200 Serial Network Interface Controller (SNIC) is a complete platform for easy Internet connectivity. The SNIC contains a firmware running onboard SN8200 to support wireless network configuration, TCP/IP network stack, WiFi driver and I/O peripherals driver. It provides to the embedded network application a socket interface over a serial bus, enabling the creation of wireless IP-capable nodes in a simple and straight forward manner. It supports dual APSTA mode that simplifies WLAN connection setup using an embedded HTTP web server onboard the SN8200 module. The customer application may communicate with the SN8200 module via either the UART or SPI interface. Each interface has its unique frame structure as described in the ensuing sections. This document provides the SNIC specification for interfacing the SN8200 with the customer application through either the UART or SPI interface.

1.1 Terms

- The term APSTA mode refers to the capability to simultaneously support both WiFi AP and STA modes of operation.
- The term IP address in this document refers to IPv4 IP address.
- The term Socket in this document refers to a one-byte ID that represents a multi-byte socket in SN82xx.

2. UART serial interface protocol

The customer application communicates with the SN8200 module via the UART interface. The default configuration using the following character format with no flow control:

- 921.6 Kbps
- 8 data bits
- no parity
- 1 stop bit.

But the interface configuration may be different. In that case, use the SNIC Monitor to modify the FW parameters to match that configuration. See the "SN8200 SNIC EVK User Guide" for details on the different UART configurations supported by SN8200.

Message exchange between the customer application and SN8200 is accomplished through a simple frame structures. The frame structure is as shown in Figure 1.

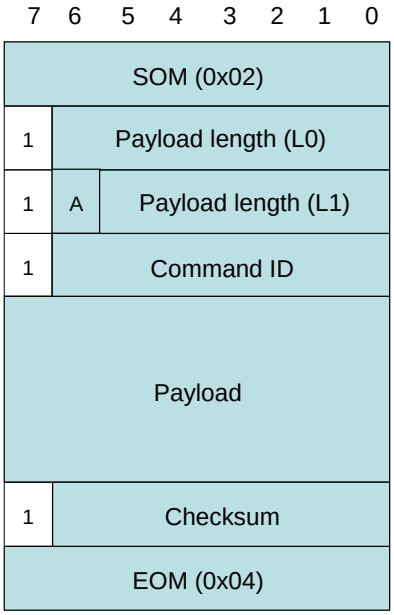

Figure 1 UART frame format

Each frame is delineated by a Start of Message (SOM) and End of Message (EOM) byte. The rest of the fields are as follows:

- Special values are SOM (0x02), EOM (0x04) and ESC (0x10). The application payload octets are remapped such that any special value is mapped to a couplet of ESC + that value added to 128, as shown in Figure 2.
- Payload length (L1:L0): octet length of application payload including any escape characters. L0 stands for bit0 to bit6, and L1 stands for bit7 to bit12 of the payload length.
- Command ID: specifies types of payload

- A: 1 if ACK required, 0 if no ACK is required. If A=1, then the receiver must send ACK upon a successful validation of the frame. A frame requiring acknowledgement must be acknowledged before any other non-response structured frame may be transmitted, i.e., a command response is always permitted.
- Checksum: sum of L0, A | L1, command ID, and application payload octets.

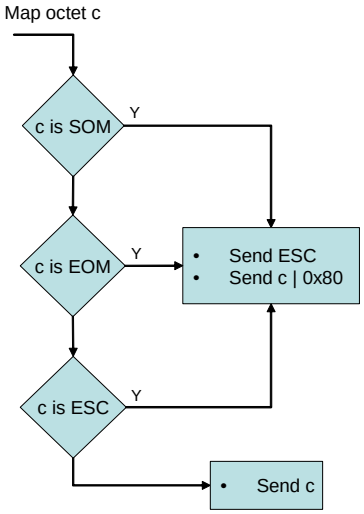

Figure 2 Mapping of payload octets for UART frames transmission

2.1 UART message acknowledgement

A transmitter sets A=1 when sending a frame requiring an acknowledgement. Once such a message is sent, the transmitter starts a timer Ttx (500ms). A transmitter must not send any UART frames (even that with A=0) until it has either received an ACK, NAK or Ttx has timedout.

Only frames received with A=1 is acknowledged. In that case, if a valid frame is reveived, the receiver composes the ACK frame (Figure 3) and sends it to the transmitter. When the transmitter recives an ACK, it stops Ttx and may send another UART frame.

The receiver validates the message after receiving the EOM. If the checksum does not match, the receiver composes the NAK frame (Figure 4) and sends it to the transmitter.

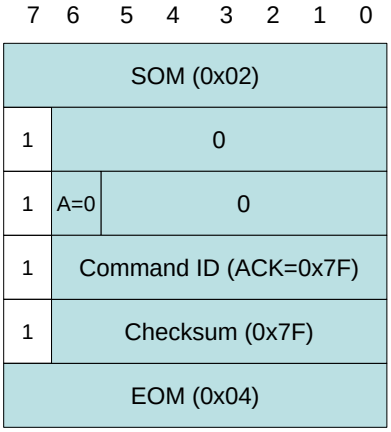

Figure 3 UART ACK frame format

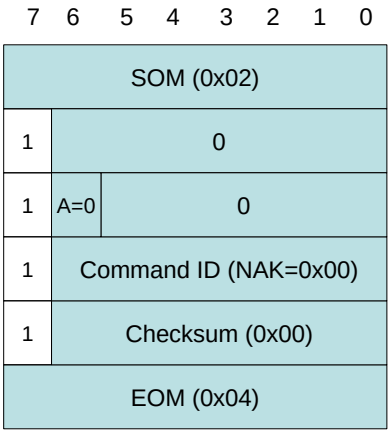

Figure 4 UART NAK frame format

2.2 UART message reception

The flow control for message reception is as shown in Figure 5. Invalid frames are silently discarded and NAK is generated as described in the previous section.

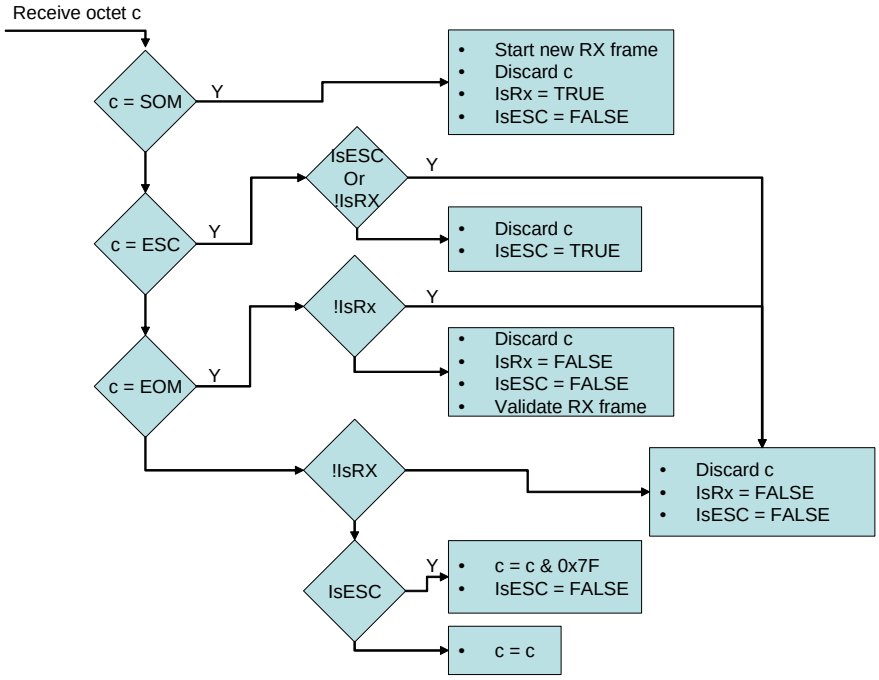

Figure 5 Flow of control for UART frame reception

2.3 Command frames

The command frames are defined in the payload as specified in the ensuing sections. The following are command IDs reserved by the serial protocol.

Command ID	Description
0x00	NAK
0x7F	ACK

Table 1 Reserved command IDs

2.4 Compatibility with future versions

We will be expanding the capabilities of the API. One such possible extension is to append new fields to the end of an existing message. For example, we may add DNS server list at the end of the SNIC_GET_DHCP_INFO_RSP message. Host application development should keep this in mind in general.

3. SPI serial interface protocol

The customer application communicates with the SN8200 module via the SPI interface (up to 18MBits/s). In the SNIC application, the SN8200 is the SPI slave. Since SN8200 is the SPI slave, it requires the master to provide the clock to send data to the host controller. When the SN8200 module has data, it indicates to the host by asserting a dedicated GPIO line (ALRT/). The signal transitions from high to low to signal the event, and returns to the high state prior to the completion of the current data transfer. If the host connects this signal to an interrupt line, then upon receiving the GPIO interrupt, the host may assert NSS and start clock to initiate data transfer from the module. In the case when the GPIO interrupt pin is not available on the host, it may assert NSS and send clock periodically to poll data from the module.

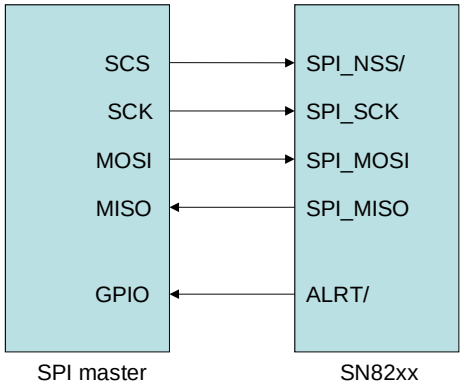

Figure 6 SN8200-host SPI interface diagram

By default, ALRT/ is P43. The SPI interface is configured with the following format: 8-bit data latched at the rising edge of the clock.

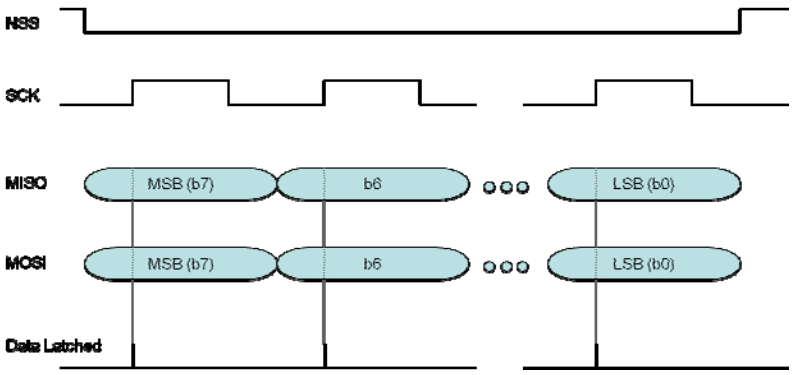

Figure 7 Default SN8200 SPI data clock timing diagram

But the interface configuration may be different. In that case, use the SNIC Monitor to modify the FW parameters to match that configuration. See the "SN8200 SNIC EVK User Guide" for details on the different SPI configurations supported by SN8200. Some of the configurable parameters are shown below.

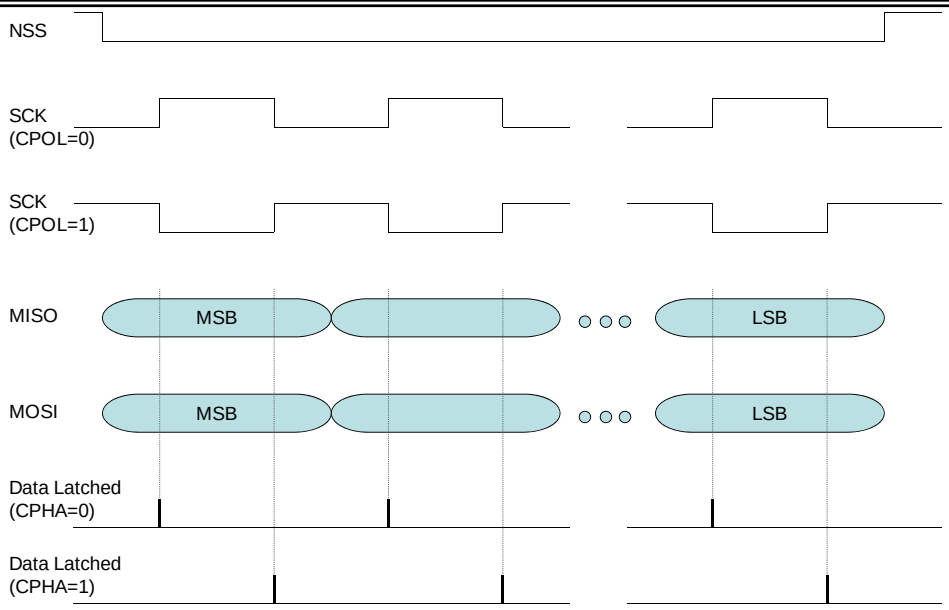

Figure 8 SPI data clock timing diagram with MSB send first

3.1 JTAG downloading and SPI

The SPI shares the 3 pins with the JTAG (see Table 94), so it is necessary to place SN8200 into BOOT mode to disable the SPI interface before using the JTAG to download a new FW into SN8200. The following reference connections may be used to enable the FW loader to control that process:

1. Drive BOOT high
2. Toggle nRESET
3. Download FW using JTAG

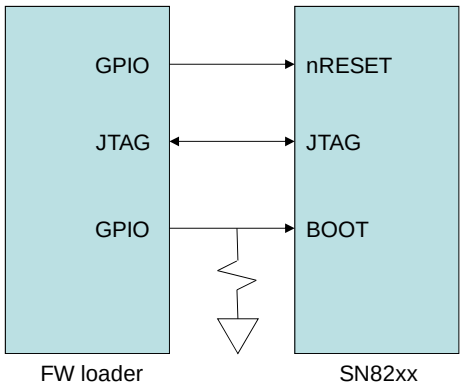

Figure 9 FW upload control

3.2 SPI frame structure

Message exchange between the customer SPI master and SN8200 is accomplished through a simple frame structures. The frame structure is as shown in Figure 10.

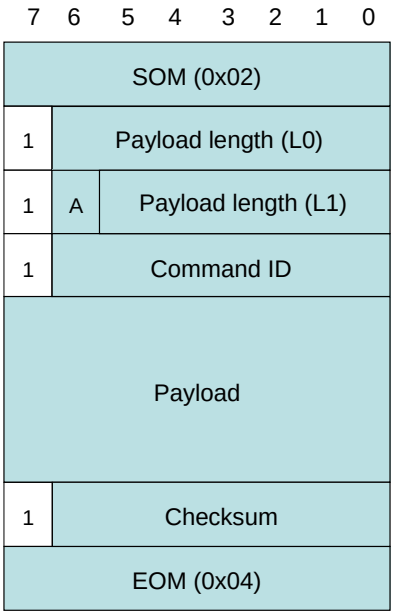

Figure 10 SPI frame format

Each frame is delineated by a Start of Message (SOM) and End of Message (EOM) byte. The header of the SPI frame is the same as that of the UART frame, but the Payload field and Checksum are treated differently, as detailed below. The rest of the fields are as follows:

- The frame starts with a SOM (0x02) and ends with an EOM (0x04). The SOM and EOM values may also appear in application payload, since the SPI interface is assumed to be stable so the payload octets are not escaped to maximum throughput. Note that this is different from the UART frame where the payload octets are escaped with an ESC.
- Payload length (L1:L0): octet length of application payload including any escape characters. L0 stands for bit0 to bit6, and L1 stands for bit7 to bit12 of the payload length.
- Command ID: specifies types of payload
- A: 1 if ACK required, 0 if no ACK is required. If A=1, then the receiver must send ACK upon a successful validation of the frame. A frame requiring acknowledgement must be acknowledged before any other non-response structured frame may be transmitted, i.e., a command response is always permitted.
- Checksum: sum of L0, A | L1 and command ID. Note that this is different from the UART frame where the checksum calculation includes the payload field.

3.3 SPI message acknowledgement

A transmitter sets A=1 when sending a frame requiring an acknowledgement. Once such a message is sent, the transmitter starts a timer Ttx (500ms). A transmitter must not send any SPI frames (even that with A=0) until it has either received an ACK, NAK or Ttx has timedout.

Only frames received with A=1 is acknowledged. In that case, if a valid frame is reveived, the receiver composes the ACK frame (Figure 11) and sends it to the transmitter. When the transmitter recives an ACK, it stops Ttx and may send another SPI frame.

The receiver validates the message after receiving the EOM. If the checksum does not match, the receiver composes the NAK frame (Figure 12) and sends it to the transmitter.

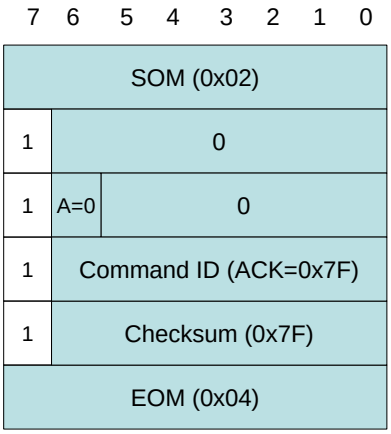

Figure 11 SPI ACK frame format

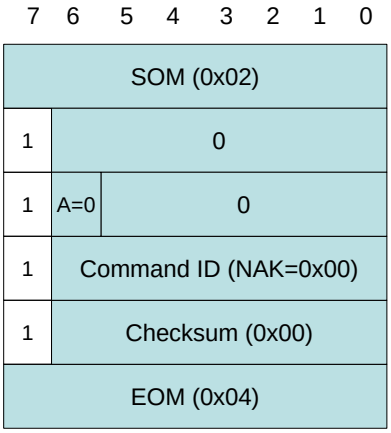

Figure 12 SPI NAK frame format

3.4 Command frames

The command frames are defined in the payload as specified in the ensuing sections. The following are command IDs reserved by the serial protocol.

Command ID	Description
0x00	NAK
0x7F	ACK

Table 2 Reserved command IDs

3.5 Compatibility with future versions

We will be expanding the capabilities of the API. One such possible extension is to append new fields to the end of an existing message. For example, we may add DNS server list at the end of the SNIC_GET_DHCP_INFO_RSP message. Host application development should keep this in mind in general.

4. General Management (CMD_ID_GEN)

The Command ID for general management is CMD_ID_GEN (0x01). The specifics of the command are defined in the payload field of the frame. All general management commands have the following format:

UINT8	SN8200 general management operation (RFSCID)
…	Rest of payload depends on RFSCID

The first byte of the payload of this command describes the general management operation, and it contains the Response Flag and Sub-Command ID (RFSCID). The MSB of RFSCID is the response flag, with values '0' and '1' indicating command request and its response, respectively. The remaining 7 LSBs of RFSCID is the Sub-Command ID (SCID) specifying the general management operation to perform. The following SCIDs are defined for general management.

SCID	Value	Description
	0x01 – 0x07	Reserved
GEN_FW_VER_GET_REQ	0x08	Get firmware version string
GEN_RESTORE_REQ	0x09	Restore to factory default

Table 3 SCIDs for general management

4.1 Get firmware version info (GEN_FW_VER_GET_REQ)

SNIC firmware has a built in version string. Use this command to retrieve the version info.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	GEN_FW_VER_GET_REQ	
1	Request Sequence	

Table 4 Get FW version info

Request Sequence is copied from the received message. The format for the SN8200 response is as follows:

Byte	Descriptions
0	GEN_FW_VER_GET_RSP (0x88)
1	Request Sequence
2	Status of operation: GEN_SUCCESS or GEN_FAILED
3	Version string length
4…	Version string []

Table 5 Response: Get FW version info

4.2 Restore to factory default (GEN_RESTORE_REQ)

This command resets the module to factory default state by restoring the NVM contents to factory default values. A soft reset will be performed automatically after the NVM has been restored. Application needs to send WIFI_GET_STATUS_REQ or SNIC_GET_DHCP_INFO_REQ commands to determine the new state of the SN82xx module.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	GEN_RESTORE_REQ	
1	Request Sequence	

Table 6 Restore to factory default

Request Sequence is copied from the received message. The format for the SN8200 response is as follows:

Byte	Descriptions
0	GEN_RESTORE_RSP (0x89)
1	Request Sequence
2	Status of operation: GEN_SUCCESS or GEN_FAILED

Table 7 Restore to factory default

5. SNIC API (CMD_ID_SNIC)

The Command ID for SNIC is CMD_ID_SNIC (0x70). The specifics of the command are defined in the payload field of the frame.

The first byte of the payload of this command describes the SNIC operation, and it contains the Response Flag and Sub-Command ID (RFSCID). The MSB of RFSCID is the response flag, with values '0' and '1' indicating command request and its response, respectively. The remaining 7 LSBs of RFSCID is the Sub-Command ID (SCID) specifying the general management operation to perform. The following SCIDs are defined for SNIC.

SCID	Value	Description
SNIC_INIT_REQ	0x00	SNIC API initialization
SNIC_CLEANUP_REQ	0x01	SNIC API cleanup
SNIC_SEND_FROM_SOCKET_REQ	0x02	Send from socket
SNIC_CLOSE_SOCKET_REQ	0x03	Close socket
SNIC_ SOCKET _PARTIAL_CLOSE_ REQ	0x04	Socket partial close
SNIC_GETSOCKOPT_REQ	0x05	Get socket option
SNIC_SETSOCKOPT_REQ	0x06	Set socket option
SNIC_SOCKET_GETNAME_REQ	0x07	Get name or peer name
SNIC_SEND_ARP_REQ	0x08	Send ARP request
SNIC_GET_DHCP_INFO_REQ	0x09	Get DHCP info
SNIC_RESOLVE_NAME_REQ	0x0A	Resolve a host name to IP address
SNIC_IP_CONFIG_REQ	0x0B	Configure DHCP or static IP
SNIC_DATA_IND_ACK_CONFIG_REQ	0x0C	ACK configuration for data indications
SNIC_TCP_CREATE_SOCKET_REQ	0x10	Create TCP socket
SNIC_TCP_CREATE_CONNECTION_REQ	0x11	Create TCP connection server
SNIC_TCP_CONNECT_TO_SERVER_REQ	0x12	Connect to TCP server
SNIC_UDP_CREATE_SOCKET_REQ	0x13	Create UDP socket
SNIC_UDP_START_RECV_REQ	0x14	Start UDP receive on socket
SNIC_UDP_SIMPLE_SEND_REQ	0x15	Send UDP packet
SNIC_UDP_SEND_ FROM_SOCKET_REQ	0x16	Send UDP packet from socket
SNIC_TCP_CONNECTION_STATUS_IND	0x20	Connection status indication
SNIC_TCP_CLIENT_SOCKET_IND	0x21	TCP client socket indication
SNIC_CONNECTION_RECV_IND	0x22	TCP or connected UDP packet

		received indication
SNIC_UDP_RECV_IND	0x23	UCP packet received indication
SNIC_ARP_REPLY_IND	0x24	ARP reply indication

Table 8 SCIDs for SNIC API

All command responses have a status octet. It can be either 0 (SNIC_SUCCESS) or an error code listed in the following table:

Return code	Value
SNIC_SUCCESS	0x00
SNIC_FAIL	0x01
SNIC_INIT_FAIL	0x02
SNIC_CLEANUP_FAIL	0x03
SNIC_GETADDRINFO_FAIL	0x04
SNIC_CREATE_SOCKET_FAIL	0x05
SNIC_BIND_SOCKET_FAIL	0x06
SNIC_LISTEN_SOCKET_FAIL	0x07
SNIC_ACCEPT_SOCKET_FAIL	0x08
SNIC_PARTIAL_CLOSE_FAIL	0X09
SNIC_SOCKET_PARTIALLY_CLOSED	0x0A
SNIC_SOCKET_CLOSED	0x0B
SNIC_CLOSE_SOCKET_FAIL	0x0C
SNIC_PACKET_TOO_LARGE	0x0D
SNIC_SEND_FAIL	0x0E
SNIC_CONNECT_TO_SERVER_FAIL	0x0F
SNIC_NOT_ENOUGH_MEMORY	0x10
SNIC_TIMEOUT	0x11
SNIC_CONNECTION_UP	0x12
SNIC_GETSOCKOPT_FAIL	0x13
SNIC_SETSOCKOPT_FAIL	0x14
SNIC_INVALID_ARGUMENT	0x15
SNIC_SEND_ARP_FAIL	0x16
SNIC_INVALID_SOCKET	0x17
SNIC_COMMAND_PENDING	0x18
SNIC_SOCKET_NOT_BOUND	0x19
SNIC_SOCKET_NOT_CONNECTED	0x1A
SNIC_NO_NETWORK	0x20
SNIC_INIT_NOT_DONE	0x21
SNIC_NET_IF_FAIL	0x22
SNIC_NET_IF_NOT_UP	0x23
SNIC_DHCP_START_FAIL	0x24

Table 9 Return code for SNIC API

NOTES:

- SCID for any response equals to the corresponding request's SCID | 0x80. E.g., SCID of SNIC_CLEANUP_RSP should be SNIC_CLEANUP_REQ | 0x80 = 0x81.
- All indication confirmations are optional, including SNIC_, WIFI_, and other prefixed indications. SCID for any indication confirmation equals to the corresponding indication's SCID | 0x80. E.g.,

SCID of SNIC_TCP_CONNECTION_STATUS_CFM should be SNIC_TCP_CONNECTION_STATUS_IND | 0x80 = 0xA0.

- If Socket is a parameter of a request, even if the request fails, the Socket will not be closed and the application needs to close it unless there is an indication of SNIC_TCP_CONNECTION_STATUS_IND with SNIC_SOCKET_CLOSED status for the Socket.

5.1 SNIC API initialization (SNIC_INIT_REQ)

This command initializes the SNIC framework on SN82xx. TCP/UDP socket communication is ready to be performed only after this command is called. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT16 Default receive buffer size

The Default receive buffer size is the default maximum size of receive buffer in the SN82xx. If 0 is specified, a system defined value (2048) will be used. If there is a Receive buffer size field in other commands, then it must be less than or equal to the Default receive buffer size. If the Receive buffer size in any of those commands is 0, the Default receive buffer size will be used. The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_INIT_REQ	
1	Request Sequence	
2	MSB Default receive buffer size	
3	LSB Default receive buffer size	

Table 10 SNIC API initialization

Upon a successful reception of the command, the SN82xx sends to the host the following response. If user specified Default receive buffer size is bigger than what the SN82xx can handle, the system defined value will be returned in the response; otherwise, user specified Default receive buffer size will be retuned. Maximum number of UDP and TCP sockets supported by SN82xx will also be returned.

Note:

1. If DHCP client has not been started and is to be started after this command, one less UDP socket will be available for application, e.g., if the response returns 3 maximum UDP sockets and DHCP client is started later on, then the available UDP socket number would be 2.
2. The total number (S) of sockets available for application usage equals the sum of UDP and TCP sockets. Number of UDP sockets returned is the maximum value. Any of the unused UDP sockets can be used for TCP in addition to those specified by Number of TCP sockets returned; meaning the actual Number of TCP sockets available for use equals S minus the actual UDP sockets being used by application. E.g., if command response shows 2 for UDP, and 2 for TCP, S = 2+2 = 4. Application can create up to 2 UDP sockets. But if application does not use UDP at all, then the actual number of TCP sockets available for application usage is S = 4. If soft AP is not started, the returned numbers should be larger, but the number of TCP socket is limited by a system preset value of 5.

Byte	Descriptions	Remark

0	SNIC_INIT_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or SNIC_INIT_FAIL	
3	MSB Default receive buffer size	Present only if Status is SNIC_SUCCESS.
4	LSB Default receive buffer size	
5	Maximum number of UDP sockets supported	
6	Maximum number of TCP sockets supported	

Table 11 Response: SNIC API initialization

5.2 SNIC API cleanup (SNIC_CLEANUP_REQ)

This command closes the SNIC framework on SN82xx. It should cleanup resources for socket communication on SN82xx. If some sockets are not closed, this command will close all of them. No more network communication can be performed until SNIC_INIT_REQ is called. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_CLEANUP_REQ	
1	Request Sequence	

Table 12 SNIC API cleanup

Upon a successful reception of the report, the SN82xx sends to the host the following response.

Byte	Descriptions
0	SNIC_CLEANUP_RSP
1	Request Sequence
2	Status of operation: SNIC_SUCCESS or SNIC_CLEANUP_FAIL

Table 13 Confirm: SNIC API cleanup

5.3 Send from socket (SNIC_SEND_FROM_SOCKET_REQ)

This command instructs the SN82xx to send a packet to the remote peer via a connected socket.

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Socket
UINT8 Option
UINT16 Payload Length
… Payload

In TCP server case, Socket is the socket number returned by SNIC_TCP_CLIENT_SOCKET_IND. In TCP client case, Socket can be either from SNIC_CONNECT_TO_TCP_SERVER_RSP, or from the SNIC_TCP_CONNECTION_STATUS_IND with SNIC_CONNECTION_UP status. In UDP case, Socket is the socket number returned by SNIC_UDP_CREATE_SOCKET_REQ and it must be in connected mode.

A success response of this command does not guarantee the receiver receives the packet. If error occurs, a SNIC_TCP_CONNECTION_STATUS_IND with SNIC_SOCKET_CLOSED will be sent to the application in TCP case. No indication will be sent in UDP case.

Option is the action SN8200 will perform to the socket after the send operation. Use it when application is sure to close or shutdown the connection after sending. The effect is the same as using SNIC_CLOSE_SOCKET_REQ or SNIC_SOCKET_PARTIAL_CLOSE_REQ, but round-trip UART traffic is reduced. The value for Option is:

0 = no action,

1 = shutdown socket in both directions, and

2 = close socket.

The payload format is as follows:

Byte	Descriptions	Get to ignore field
0	SNIC_TCP_SEND_FROM_SOCKET_REQ	
1	Request Sequence	
2	Socket	
3	Option	
4	MSB Payload Length	
5	LSB Payload Length	
	… Payload	

Table 14 TCP Send from socket

Request Sequence is copied from the received message. The format for the response is as follows:

Byte	Descriptions	Remark
0	SNIC_SEND_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or Error code	
3	MSB Number of bytes sent	Present only if Status is SNIC_SUCCESS.
4	LSB Number of bytes sent	

Table 15 Response: TCP Send from socket

Applicable error codes are:
- SNIC_PACKET_TOO_LARGE
- SNIC_SEND_FAIL

Note: if there is no error in sending, the action of closing or shuting down the socket should be assumed successful.

5.4 Close Socket (SNIC_CLOSE_SOCKET_REQ)

This command instructs the SN82xx to close a socket.

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Socket

Socket is any valid socket number in both TCP and UDP cases.

The payload format is as follows:

Byte	Descriptions	Get to ignore field
0	SNIC_CLOSE_SOCKET_REQ	
1	Request Sequence	
2	Socket	

Table 16 Close Socket Connection

Request Sequence is copied from the received message. The format for the response is as follows:

Byte	Descriptions
0	SNIC_CLOSE_SOCKET_RSP
1	Request Sequence
2	Status of operation: SNIC_SUCCESS or SNIC_CLOSE_SOCKET_FAIL

Table 17 Response: Close Socket Connection

5.5 Socket Partial Close (SNIC_SOCKET_PARTIAL_CLOSE_REQ)

This command instructs the SN82xx to partially close a connection oriented socket. It is usually used for gracefully closing a connection. This command is applicable for TCP socket and connect mode UDP socket.

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Socket
UINT8 Direction

Values for Direction are: 0-receive, 1-send, and 2-both directions. Currently only 2 (both directions) is supported. Note: even if both directions are shutdown, the socket is not automatically closed. The payload format is as follows:

Byte	Descriptions	Get to ignore field
0	SNIC_SOCKET_PARTIAL_CLOSE_REQ	
1	Request Sequence	
2	Socket	
3	Direction	

Table 18 Socket Partial Close

Request Sequence is copied from the received message. The format for the response is as follows:

Byte	Descriptions
0	SNIC_SOCKET_PARTIAL_CLOSE__RSP
1	Request Sequence
2	Status of operation: SNIC_SUCCESS or SNIC_PARTIAL_CLOSE_FAIL

Table 19 Response: Socket Partial Close

5.6 Get socket option (SNIC_GETSOCKOPT_REQ)

This command gets the socket option from the SN82xx. Parameters are:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Socket
UINT16 Level
UINT16 Option name

Level can be either SOL_SOCKET (0x0fff), IPPROTO_IP (0x0000), or IPPROTO_TCP (0x0006). Option names are defined in the following tables:

NOTE: For BOOL type, size is 4 bytes, and non-zero=true, 0=false.

Option Name	Type	Meaning
SO_BROADCAST (0x0020)	BOOL	Specify whether the socket can send broadcast message
SO_DONTLINGER (0xFF7F)	BOOL	Specify whether the socket should not block the socket close operation while waiting for data to be send.
SO_LINGER (0x0080)	UINT16	Set socket close linger time in seconds.
SO_RCVBUF (0x1002)	UINT16	The total per-socket buffer space reserved for receives.
SO_REUSEADDR (0x0004)	BOOL	The socket can be bound to an address which is already in use.
SO_SNDBUF (0x1001)	UINT16	The total per-socket buffer space reserved for sends.
SO_TYPE (0x1008)	UINT32	The type of the socket (SOCK_STREAM 1 SOCK_DGRAM 2 SOCK_RAW 3).

Table 20 SOL_SOCKET option names

Option Name	Type	Meaning
IP_ADD_MEMBERSHIP (0x0003)	IP_MREQ	Request to receive from a multicast group
IP_DROP_MEMBERSHIP (0x0004)	IP_MREQ	Request to stop receiving from a multicast group
IP_MULTICAST_TTL (0x0005)	UINT8	Specify the time-to-live for a multicast packet
IP_MULTICAST_IF (0x0006)	UINT32	Specify the address of the multicast interface
IP_TOS (0x0001)	UINT32	Specify the type of service parameter
IP_TTL (0x0002)	UINT32	Specify the time-to-live for a non-multicast packet

Table 21 IPPROTO_IP option names

Note: IP_MREQ contains 2 4-byte IP addresses:
UINT32 IP multicast address of group
UINT32 local IP address of interface

Option Name	Type	Meaning
TCP_NODELAY (0x0001)	BOOL	Disables the Nagle algorithm for

		send coalescing.

Table 22 IPPROTO_TCP option names

The payload format is as follows:

Byte	Descriptions	Get to ignore field
0	SNIC_GETSOCKOPT_REQ	
1	Request Sequence	
2	Socket	
3	MSB Level	
4	LSB Level	
5	MSB Option name	
6	LSB Option name	

Table 23 Get socket option

For response, Request Sequence is copied from the received message. All multi-byte values are transferred in big endian except the Return value which is transferred in little endian. The format for the response is as follows:

Byte	Descriptions	Remark
0	SNIC_GETSOCKOPT_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or SNIC_GETSOCKOPT_FAIL	If command fails, it means the option is not supported.
3	MSB Return value length	Present only if status is SNIC_SUCCESS.
4	LSB Return value length	
	… Return value	

Table 24 Response: Get socket option

5.7 Set socket option (SNIC_SETSOCKOPT_REQ)

This command sets the socket option from the SN82xx. Parameters are:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Socket
UINT16 Level
UINT16 Option name
UINT16 Option value length

… Option value

Option names see Table 20, Table 21, and Table 22. All multi-byte values are transferred in big endian except the Option value which is transferred in little endian. The payload format is as follows:

Byte	Descriptions	Get to ignore field
0	SNIC_SETSOCKOPT_REQ	
1	Request Sequence	
2	Socket	
3	MSB Level	
4	LSB Level	
5	MSB Option name	
6	LSB Option name	
7	MSB Option value length	
8	LSB Option value length	
	… Option value	

Table 25 Set socket option

Request Sequence is copied from the received message. The format for the response is as follows:

Byte	Descriptions	Remark
0	SNIC_SETSOCKOPT_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or SNIC_SETSOCKOPT_FAIL	If command fails, it means the option is not supported, value, or length not valid.

Table 26 Response: Set socket option

Note: if the socket is a UDP socket, but the level to set is IPPROTO_TCP, the command will be ignored and status of operation will be SNIC_SUCCESS.

5.8 Send ARP request (SNIC_SEND_ARP_REQ)

This command queries the physical (MAC) address for a particular IP address. The parameters are:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Interface number
UINT32 Destination IP address

UINT32 Source IP address
UINT8 Timeout

Interface number is either 0 or 1. 0 indicates STA interface. 1 indicates soft AP interface. Currently only STA interface is supported.

The ARP request attempts to obtain the physical address that corresponds to the Destination IP address. The Source IP address is the IP address of the sender. It is used to select the interface to send the request on for the ARP entry in a multi-homed system. The application may specify 0x0 corresponding to the INADDR_ANY IPv4 address for this parameter.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_SEND_ARP_REQ	
1	Request Sequence	
2	Interface Number	
3	MSB Destination IP address	
4	…	
5	…	
6	LSB Destination IP address	
7	MSB Source IP address	
8	…	
9	…	
10	LSB Source IP address	
11	Timeout (in seconds)	

Table 27 Send ARP

The response does not contain the MAC address requested. If no command level error, status SNIC_COMMAND_PENDING will be in the response. Otherwise, if there is conflict in the source and destination IP address, status SNIC_SEND_ARP_FAIL will be in the response. If SN82xx receives an ARP reply, the MAC address will be reported by SNIC_ARP_REPLY_IND indication. If timeout, the indication will have SNIC_TIMEOUT status.

Note: If multiple ARP requests need to be sent, it is required that the commands be sent sequentially. Wait for either a timeout indication or a success indication before sending another ARP request. This is due to system resource limitation. While waiting for the indication, host application can send other commands (except for SNIC_SEND_ARP_REQ and SNIC_RESOLVE_NAME_REQ).

Request Sequence is copied from the received request message. The format from the SN82xx response is as follows:

Byte	Descriptions	Remark
0	SNIC_SEND_ARP_RSP	
1	Request Sequence	
2	Status of operation: SNIC_COMMAND_PENDING or SNIC_SEND_ARP_FAIL	

Table 28 Response: Send ARP

5.9 Get DHCP info (SNIC_GET_DHCP_INFO_REQ)

This command queries the DHCP information for a particular interface. The parameters are:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Interface number

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_GET_DHCP_INFO_REQ	
1	Request Sequence	
2	Interface number	

Table 29 Get DHCP info

Interface number is either 0 or 1. 0 indicates STA interface. 1 indicates soft AP interface.

Request Sequence is copied from the received request message. The format from the SN82xx response is as follows:

Byte	Descriptions	Remark
0	SNIC_GET_DHCP_INFO_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or SNIC_FAIL	
3	Local MAC[0]	Present only if Status is SNIC_SUCCESS.
4-7	…	
8	Local MAC[5]	
9	MSB Local IP	
10-11	…	
12	LSB Local IP	
13	MSB Gateway IP	
14-15	…	
16	LSB Gateway IP	

17	MSB Subnet mask	
18-19	…	
20	LSB Subnet mask	

Table 30 Response: Get DHCP info

5.10 Get socket name or peer name (SNIC_SOCKET_GETNAME_REQ)

This command gets the socket's local name or peer name if connected. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Which name
UINT8 Socket

If Which name is 0, the response contains the local IP and port if the socket is bound. If the socket is not bound, SNIC_SOCKET_NOT_BOUND will be returned.

If Which name is 1, the reponse contains the peer's IP and port if the socket is connected. If the socket is not connected, SNIC_SOCKET_NOT_CONNECTED will be returned. In UDP case, if the Socket is used in SNIC_UDP_SEND_FROM_SOCKET_REQ with Connection mode set to 1, then the socket is considered connected, even though the peer has not done the same.

For other failure, SNIC_FAIL will be returned.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_SOCKET_GETNAME_REQ	
1	Request Sequence	
2	Which name	
3	Socket	

Table 31 Get socket name or peer name

Request Sequence is copied from the received request message. The format from the SN82xx response is as follows:

Byte	Descriptions	Remark
0	SNIC_SOCKET_GETNAME_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or Error code	
3	MSB IP address	Present only if status is SNIC_SUCCESS.
4	…	
5	…	

6	LSB IP address	
7	MSB Port	
8	LSB Port	

Table 32 Response: Get socket name or peer name

Applicable Error codes are:
- SNIC_INVALID_SOCKET
- SNIC_SOCKET_NOT_BOUND
- SNIC_SOCKET_NOT_CONNECTED
- SNIC_FAIL

5.11 Resolve host name to IP address (SNIC_RESOLVE_NAME_REQ)

This command converts a remote host name to IP address. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Interface number
UINT8 Name Length
UINT8 Name [ ]

Interface number is either 0 or 1. 0 indicates STA interface. 1 indicates soft AP interface. Currently only STA interface is supported.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_RESOLVE_NAME_REQ	
1	Request Sequence	
2	Name length (n)	
3	Name [0]	
…	…	
	Name [n-1]	

Table 33 Get host by name

If multiple SNIC_RESOLVE_NAME_REQ's need to be sent, it is required they be sent sequentially due to resource limitation. If the name is not resolved, it takes up to 15 seconds for the failure response to come back. While waiting for the response, host application can send other commands (except for SNIC_RESOLVE_NAME_REQ and SNIC_SEND_ARP_REQ).

Request Sequence is copied from the received request message. The format from the SN82xx response is as follows:

Byte	Descriptions	Remark
0	SNIC_RESOLVE_NAME_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or SNIC_FAIL.	
3	MSB IP address	Present only if status is SNIC_SUCCESS.
4	…	
5	…	
6	LSB IP address	

Table 34 Response: Get host by name

5.12 Configure DHCP or static IP (SNIC_IP_CONFIG_REQ)

This command instructs SN82xx configure the mechanism for obtaining the IP address. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Interface number
UINT8 DHCP mode
UINT32 IP
UINT32 Netmask
UINT32 Gateway IP

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_IP_CONFIG_REQ	
1	Request Sequence	
2	Interface Number (0=STA, 1=AP)	
3	DHCP mode (0=static IP, 1=DHCP)	
4	MSB IP	Present only if DHCP mode = 0.
…	…	
7	LSB IP	
8	MSB Netmask	
…	…	
11	LSB Netmask	
12	MSB Gateway IP	
…	…	
15	LSB Gateway IP	

Table 35 IP Config

Upon a successful reception of the command, the SN82xx tries to configure IP. If DHCP is enabled, DHCP client will start. Configuration parameters are saved in NVM for later use.

Command response is as follows:

Byte	Descriptions	Remark
0	SNIC_IP_CONFIG_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or error codes.	

Table 36 Response: IP Config

Error codes:

- SNIC_NET_IF_FAIL
- SNIC_NET_IF_NOT_UP
- SNIC_NO_NETWORK
- SNIC_DHCP_START_FAIL

5.13 ACK Configuration for data indications (SNIC_DATA_IND_ACK_CONFIG_REQ)

This command configures the ack parameters if host application needs to acknowledge data indications. By default, all serial messages have the ACK bit set to 0. See Section 2.1. That mean all UART messages are not acknowledged in the serial protocol level. For slower host, if UART flow control cannot be enabled and the host cannot read from the serial port fast enough, this command provides a way to let SN8200 to set the ACK bit to 1 for data indications (i.e., SNIC_CONNECTION_RECV_IND or SNIC_UDP_RECV_IND). So for each data indication sent from SN8200 to the host, SN8200 will wait for the serial protocol ACK before sending the next data indication. If within “timeout”, the SN8200 has not received the ACK, it will resend the indication. User can configure the “timeout” value, and the retry number.

Note: If “Retry times” is set to 0 or 1, the data indication will be sent once.

Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Protocol
UINT8 Ack Enable
UINT16 Timeout
UINT8 Retry times

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_DATA_IND_ACK_CONFIG_REQ	

1	Request Sequence	
2	Protocol (1=TCP, 2=UDP, 3=both)	
3	Ack Enable (1=enabled, 0=disabled)	
4	MSB Timeout (in milliseconds)	
5	LSB Timeout	
6	Retry times	

Table 37 Ack config for data indications

Upon a successful reception of the command, the SN82xx sets the parameters for later use. Command response is as follows:

Byte	Descriptions	Remark
0	SNIC_DATA_IND_ACK_CONFIG_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or SNIC_FAIL	

Table 38 Response: Ack config for data indications

5.14 Create TCP socket (SNIC_TCP_CREATE_SOCKET_REQ)

This command creates a TCP socket with bind option. The parameters are:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Bind option
UINT32 Local IP address
UINT16 Local port

If Bind option is 0, the socket will not be bound, and Local IP address and Local port should not be present. Otherwise, it will be bound to Local IP address and Local port specified. 0x0 for IP or port are valid, which means system assigned.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_TCP_CREATE_SOCKET_REQ	
1	Request Sequence	
2	Bind option	
3	MSB Local IP address	Present only if Bind Option is 1.
4	…	
5	…	

6	LSB Local IP address	
7	MSB Local port	
8	LSB Local port	

Table 39 Create TCP socket

The Socket created will be returned in the response from SN82xx, which can be used to create a TCP connection server or connect to a remote server. Request Sequence is copied from the received request message. The format from the SN82xx response is as follows:

Byte	Descriptions	Remark
0	SNIC_TCP_CREATE_SOCKET_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or Error code	
3	Socket	Present only if status is SNIC_SUCCESS.

Table 40 Response: Create TCP Socket

Applicable Error codes are:
- SNIC_CREATE_SOCKET_FAIL
- SNIC_BIND_SOCKET_FAIL

5.15 Create TCP connection server (SNIC_TCP_CREATE_CONNECTION_REQ)

This command instructs the SN82xx to use the TCP server socket to listen for incoming connection. The socket for a successfully established connection is reported via SNIC_TCP_CLIENT_SOCKET_IND. The actual incoming data packets will be delivered to host application via the SNIC_CONNECTION_RECV_IND indication after client connections.

The command has the following parameters:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Socket
UINT16 Receive buffer size
UINT8 Maximum client connections

Socket must have been created by SNIC_TCP_CREATE_SOCKET_REQ command (with Bind option equals 1). Application can use SNIC_SETSOCKOPT_REQ command to configure the socket before creating the connection.

Receive buffer size is the maximum packet size the application wants to receive per transmission. It should be less than or equal to the Default receive buffer size specified in SNIC_INIT_REQ. If it is 0 or exceeds the system capability, the Default receive buffer size is returned.

Maximum client connections is the maximum number of client connections this connection server can serve simutanously. If the number is 0 or exceeds the system capability, the actual value is returned in the response.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_TCP_CREATE_CONNECTION_REQ	
1	Request Sequence	
2	Socket	
3	MSB Receive buffer size	
4	LSB Receive buffer size	
5	Maximum client connections	

Table 41 Create TCP connection server

If no error, a response with the actual Receive buffer size and Maximum client connections should be returned. The socket specified in the request will be listening for incoming connection request from client. If there is an error, application is responsible to close the socket or use it for other purposes.

Request Sequence is copied from the received request message. The format from the SN82xx response is as follows:

Byte	Descriptions	Remark
0	SNIC_CREATE_TCP_CONNECTION_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or SNIC_LISTEN_SOCKET_FAIL	
3	MSB Receive buffer size	Present only if Status is SNIC_SUCCESS.
4	LSB Receive buffer size	
5	Maximum client connections	

Table 42 Response: Create TCP connection server

5.16 Connect to TCP Server (SNIC_TCP_CONNECT_TO_SERVER_REQ)

This command instructs the SN82xx to connect to a remote TCP server.

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Socket
UINT32 Server IP address
UINT16 Server Port

UINT16 Receive buffer size
UINT8 Timeout

Socket must have been created by SNIC_TCP_CREATE_SOCKET_REQ command. Application can use SNIC_SETSOCKOPT_REQ command to configure the socket before making the connect request.

Receive buffer size is the maximum packet size the application wants to receive per transmission. It must be less than or equal to the Default receive buffer size from SNIC_INIT_REQ in the SN82xx. If it is 0 or exceeds the system capability, the Default receive buffer size is returned.

If the connect attempt is immediately completed, the response will contain SNIC_SUCCESS status, with the actual Receive buffer size.

If the connect attempt is not immediately completed, the response will have the SNIC_COMMAND_PENDING status. The Timeout value is the time (in seconds) the SN82xx will wait before aborting the connection attempt. If timeout occurs, the SNIC_TCP_CONNECTION_STATUS_IND indication with SNIC_TIMEOUT status will be sent to the application. If connection is successful before timeout, the SNIC_TCP_CONNECTION_STATUS_IND with SNIC_CONNECTION_UP status will be sent to the application. Timeout value should be non-zero.

Note: If multiple connect requests need to be sent, it is required that the commands be sent sequentially. Wait for either a timeout indication or a success indication before sending another connect request. This is due to system resource limitation.

The payload format is as follows:

Byte	Descriptions	Get to ignore field
0	SNIC_TCP_CONNECT_TO_SERVER_REQ	
1	Request Sequence	
2	Socket	
3	MSB Server IP address	
4	…	
5	…	
6	LSB Server IP address	
7	MSB Server Port	
8	LSB Server Port	
9	MSB Receive buffer size	
10	LSB Receive buffer size	
11	Timeout	

Table 43Connect to TCP Server

If no error, a response with the actual Receive buffer size should be returned. Send and receive can be done through the socket specified. Incoming data will be sent to application by SNIC_CONNECTION_RECV_IND. If SNIC_CONNECT_TO_SERVER_FAIL is returned, the socket is still open and can be used later.

Request Sequence is copied from the received message. The format for the response is as follows:

Byte	Descriptions	Remark
0	SNIC_CONNECT_TO_TCP_SERVER_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS, SNIC_COMMAND_PENDING or Error code	
3	MSB Receive buffer size	Present only if status is SNIC_SUCCESS.
4	LSB Receive buffer size	

Table 44 Response: Connect to TCP Server

Applicable Error codes are:
- SNIC_CONNECT_TO_SERVER_FAIL (
- SNIC_INVALID_SOCKET

5.17 Create UDP Socket (SNIC_UDP_CREATE_SOCKET_REQ)

This command instructs the SN82xx to create a UDP socket. The socket returned by this command may be used in subsequent SNIC_UDP_SEND_FROM_SOCKET_REQ or SNIC_UDP_START_RECV_REQ command.

The command has the following parameters:

UINT8 Request Sequence (0x00 – 0x7F)
UINT8 Bind option
UINT32 Local IP address
UINT16 Local port

If Bind option is 0, the socket will not be bound, and Local IP address and Local port should not be present. Otherwise, it will be bound to Local IP address and Local port specified. 0x0 for IP or port are valid, which means system assigned.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_UDP_CREATE_SOCKET_REQ	
1	Request Sequence	
2	Bind option	
2	MSB Local IP address	Present only if Bind Option is 1.
3	…	
4	…	
5	LSB Local IP address	
6	MSB Local port	
7	LSB Local port	

Table 45 Create UDP socket

Request Sequence is copied from the received request message. The format from the SN82xx response is as follows:

Byte	Descriptions	Remark
0	SNIC_CREATE_UDP_SOCKET_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or Error code	
3	Socket	Present only if status is SNIC_SUCCESS.

Table 46 Response: Create UDP Socket

Applicable Error codes are:
- SNIC_CREATE_SOCKET_FAIL
- SNIC_BIND_SOCKET_FAIL

5.18 Start UDP receive on socket (SNIC_UDP_START_RECV_REQ)

This command instructs the SN82xx to start receiving incoming data packets. The actual data packets will be delivered to host application via the SNIC_UDP_RECV_IND or SNIC_CONNECTION_RECV_IND indication.

The command has the following parameters:

UINT8 Request Sequence (0x00 – 0x7F)
UINT8 Socket
UINT16 Receive buffer size

The Socket should have been created by command SNIC_UDP_CREATE_SOCKET_REQ. The same socket can be used in SNIC_UDP_SEND_FROM_SOCKET_REQ command, so that send and receive can be done via the same socket (port). The application is responsible to close the socket using SNIC_CLOSE_SOCKET_REQ.

Receive buffer size is the maximum packet size the application wants to receive per transmission. It must be less than or equal to the Default receive buffer size from SNIC_INIT_REQ in the SN82xx. If 0 or exceeds the system capability, the Default receive buffer size will be used and returned in the response.

After this command, the Socket can receive any UDP sender with connected mode or non-connected mode. The SN82xx will generate SNIC_UDP_RECV_IND indication for incoming data, which includes sender's IP and port info.

But if this Socket is later connected to a peer UDP server by SNIC_UDP_SEND_FROM_SOCKET_REQ with Connection mode set to1, the SN82xx will generate SNIC_CONNECTION_RECV_IND indication without the sender's IP and port info. See Section 5.20. After that, this Socket will only be able to receive from the one sender it connects to.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_UDP_START_RECV_REQ	
1	Request Sequence	
2	Socket	
3	MSB Receive buffer size	
4	LSB Receive buffer size	

Table 47 Start UDP Receive

Request Sequence is copied from the received request message. The format from the SN82xx response is as follows:

Byte	Descriptions	Remark
0	SNIC_UDP_START_RECV_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or SNIC_NOT_ENOUGH_MEMORY	
3	MSB Receive buffer size	Present only if Status is SNIC_SUCCESS.
4	LSB Receive buffer size	

Table 48 Response: Start UDP Receive

5.19 Send UDP packet (SNIC_UDP_SIMPLE_SEND_REQ)

This command instructs the SN82xx to send a packet to the remote peer.

UINT8 Request Sequence (0x00 - 0x7F)
UINT32 Remote IP address
UINT16 Remote port
UINT16 Payload length
… Payload

A socket will be created for sending the packet out through the default network connection, but will be closed after the transmission. This command can be used when the application just wants to send out one packet to peer, and it also does not expect to receive any packets from peer.

The payload format is as follows:

Byte	Descriptions	Get to ignore field
0	SNIC_UDP_SIMPLE_SEND_REQ	
1	Request Sequence	

2	MSB Remote IP address	
3	…	
4	…	
5	LSB Remote IP address	
6	MSB Remote Port	
7	LSB Remote Port	
8	MSB Payload Length	
9	LSB Payload Length	
	… Payload	

Table 49 Send UDP packet

Request Sequence is copied from the received message. The format for the response is as follows:

Byte	Descriptions	Remark
0	SNIC_UDP_SIMPLE_SEND_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or Error code	
3	MSB Number of bytes sent	Present only if Status is SNIC_SUCCESS.
4	LSB Number of bytes sent	

Table 50 Response: Send UDP packet

Applicable Error codes are:

- SNIC_CREATE_SOCKET_FAIL
- SNIC_SEND_FAIL

5.20 Send UDP packet from socket (SNIC_UDP_SEND_FROM_SOCKET_REQ)

This command instructs the SN82xx to send a packet to the remote peer.

UINT8 Request Sequence (0x00 - 0x7F)
UINT32 Remote IP address
UINT16 Remote port
UINT8 Socket
UINT8 Connection mode
UINT16 Payload length
… Payload

The Socket should have been created by command SNIC_UDP_CREATE_SOCKET_REQ. If SNIC_UDP_START_RECV_REQ is not called on the socket, the application can only send out UDP packet from this socket. If SNIC_UDP_START_RECV_REQ has been called for this socket, the application can send and receive UDP packets from the socket. This implies the application can send and receive packets from the same local port. The application is responsible to close the socket using SNIC_CLOSE_SOCKET_REQ.

If Connection mode is 1, the SN82xx will first connect to the UDP server then send data. Since the socket is still connected after the call, application can send subsequent data using another command SNIC_SEND_FROM_SOCKET_REQ.

The benefit of the connected mode is that subsequent send can use SNIC_SEND_FROM_SOCKET_REQ, which does not require the receiver's IP and port every time, and thus reduces overhead. If this socket is also used to receive by calling SNIC_UDP_START_RECV_REQ, the receive indication to the host will also omits the sender IP and port info, further reducing overhead.

The payload format is as follows:

Byte	Descriptions	Get to ignore field
0	SNIC_UDP_SEND_FROM_SOCKET_REQ	
1	Request Sequence	
2	MSB Remote IP address	
3	…	
4	…	
5	LSB Remote IP address	
6	MSB Remote Port	
7	LSB Remote Port	
8	Socket	
9	Connection mode	
10	MSB Payload Length	
11	LSB Payload Length	
	… Payload	

Table 51 Send UDP packet from socket

Request Sequence is copied from the received message. The format for the response is as follows:

Byte	Descriptions	Remark
0	SNIC_UDP_SEND_FROM_SOCKET_RSP	
1	Request Sequence	
2	Status of operation: SNIC_SUCCESS or Error code	
3	MSB Number of bytes sent	Present only if Status is

4	LSB Number of bytes sent	SNIC_SUCCESS.

Table 52 Response: Send UDP packet from socket

Applicable Error codes are:

- SNIC_CONNECT_TO_SERVER_FAIL
- SNIC_SEND_FAIL

5.21 Connection status indication (SNIC_TCP_CONNECTION_STATUS_IND)

Indication is originated from the SN82xx and sent to the host application.

This event describes the status of a network connection (identified by a socket) using one of the codes defined in Table 9. The indication has the following parameters:

UINT8 Indication Sequence (0x00 - 0x7F)
UINT8 Status code
UINT8 Socket

All multi-byte values are transferred in big endian. The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_TCP_CONNECTION_STATUS_IND	
1	Indication Sequence	
2	Status code	
3	Socket	

Table 53 Connection status indication

Upon a successful reception of the report, the host application sends to the SN82xx the following reception confirmation. Indication Sequence is copied from the received indication.

Byte	Descriptions
0	SNIC_CONNECTION_STATUS_CFM
1	Indication Sequence

Table 54 Confirm: Connection status indication

Applicable Status codes are:

- SNIC_CONNECTION_UP
- SNIC_ACCEPT_SOCKET_FAIL
- SNIC_SOCKET_PARTIALLY_CLOSED

- SNIC_SOCKET_CLOSED
- SNIC_SEND_FAIL
- SNIC_CONNECT_TO_SERVER_FAIL
- SNIC_TIMEOUT

5.22 TCP Client Socket Indication (SNIC_TCP_CLIENT_SOCKET_IND)

This event is generated when a TCP client has connected to the server running in the SN82xx. It contains the listenning socket (which should have been specified in SNIC_TCP_CREATE_CONNECTION_REQ before) and the client socket. The client socket is then used by the host application in subsequent communications. The inidcation has the following parameters:

UINT8 Indication Sequence (0x00 - 0x7F)
UINT8 Listenning socket
UINT8 Client socket
UINT32 Client IP address
UINT16 Client Port

All multi-byte values are transferred in big endian. The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_TCP_CLIENT_SOCKET_IND	
1	Indication Sequence	
2	Listenning socket	
3	Client socket	
4	MSB Client IP address	
5	…	
6	…	
7	LSB Client IP address	
8	MSB Client Port	
9	LSB Client Port	

Table 55 TCP Client Socket Indication

Upon a successful reception of the report, the host application sends to the SN82xx the following reception confirmation. Application can deny further access (after examine the client IP and port) by closing the Client Socket.

Indication Sequence is copied from the received indication.

Byte	Descriptions
0	SNIC_TCP_CLIENT_SOCKET_CFM

1	Indication Sequence

Table 56 Confirm: TCP Client Socket Indication

5.23 TCP or connected UDP packet received indication (SNIC_CONNECTION_RECV_IND)

This event is generated when a TCP server or a UDP server (in connected mode) receives a packet. Since there is no client address and port information, the application may need to call SNIC_SOCKET_GETNAME_REQ to get them. The inidcation has the following parameters:

UINT8 Indication Sequence (0x00 - 0x7F)
UINT8 Server socket
UINT16 Payload length
… Payload

All multi-byte values are transferred in big endian. The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_CONNECTION_RECV_IND	
1	Indication Sequence	
2	Server socket	
3	MSB Payload length	
4	LSB Payload length	
	… Payload	

Table 57 TCP packet received indication

Upon a successful reception of the report, the host application sends to the SN82xx the following reception confirmation. Indication Sequence is copied from the received indication.

Byte	Descriptions
0	SNIC_CONNECTION_RECV_CFM
1	Indication Sequence

Table 58 Confirm: TCP packet received indication

5.24 UDP packet received indication (SNIC_UDP_RECV_IND)

This event is generated when a UDP server (in unconnected mode) receives a packet. The indication has the following parameters:

UINT8 Indication Sequence (0x00 - 0x7F)
UINT8 Server socket
UINT32 Remote IP address
UINT16 Remote port
UINT16 Payload length
… Payload

All multi-byte values are transferred in big endian. The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_UDP_RECV_IND	
1	Indication Sequence	
2	Server socket	
3	MSP Remote IP address	
4	…	
5	…	
6	LSB Remote IP address	
7	MSB Remote port	
8	LSB Remote port	
9	MSB Payload length	
10	LSB Payload length	
	… Payload	

Table 59 UDP packet received indication

Upon a successful reception of the report, the host application sends to the SN82xx the following reception confirmation. Indication Sequence is copied from the received indication.

Byte	Descriptions
0	SNIC_UDP_RECV_CFM
1	Indication Sequence

Table 60 Confirm: UDP packet received indication

5.25 ARP reply indication (SNIC_ARP_REPLY_IND)

This event is generated when SN82xx receives an ARP reply or if a pending ARP reply timesout. The indication has the following parameters:

UINT8 Indication Sequence (0x00 - 0x7F)
UINT32 Destination IP address
UINT8 Status
UINT8 MAC address [6]

All multi-byte values are transferred in big endian. The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	SNIC_ARP_REPLY_IND	
1	Indication Sequence	
2	MSP Destination IP address	
3	…	
4	…	
5	LSB Destination IP address	
6	Status: SNIC_SUCCESS or SNIC_TIMEOUT	
7	MAC address [0]	Present only if Status is SNIC_SUCCESS.
8-11	…	
12	MAC address[5]	

Table 61 ARP reply indication

Upon a successful reception of the report, the host application sends to the SN82xx the following reception confirmation. Indication Sequence is copied from the received indication.

Byte	Descriptions
0	SNIC_ARP_REPLY_CFM
1	Indication Sequence

Table 62 Confirm: ARP reply indication

6. WIFI API (CMD_ID_WIFI)

The Command ID for WIFI is CMD_ID_WIFI (0x50). The specifics of the command are defined in the payload field of the frame.

The first byte of the payload of this command describes the WIFI operation, and it contains the Response Flag and Sub-Command ID (RFSCID). The MSB of RFSCID is the response flag, with values '0' and '1' indicating command request and its response, respectively. The remaining 7 LSBs of RFSCID is the Sub-Command ID (SCID) specifying the general management operation to perform. The following SCIDs are defined for SNIC.

SCID	Value	Description
WIFI_ON_REQ	0x00	Turn on Wifi
WIFI_OFF_REQ	0x01	Turn off Wifi
WIFI_JOIN_REQ	0x02	Associate to a network
WIFI_DISCONNECT_REQ	0x03	Disconnect from a network
WIFI_GET_STATUS_REQ	0x04	Get WiFi status
WIFI_SCAN_REQ,	0x05	Scan WiFi networks
WIFI_GET_STA_RSSI_REQ	0x06	Get STA signal strength (RSSI)
WIFI_AP_CTRL_REQ	0x07	Soft AP on-off control
WIFI_NETWORK_STATUS_IND	0x10	Network status indication
WIFI_SCAN_RESULT_IND	0x11	Scan result indication

Table 63 SCIDs for WIFI API

Return codes for WIFI API are listed in the following table.

WIFI_SUCCESS	0x00
WIFI_ERR_UNKNOWN_COUNTRY	0x01
WIFI_ERR_INIT_FAIL	0x02
WIFI_ERR_ALREADY_JOINED	0x03
WIFI_ERR_AUTH_TYPE	0x04
WIFI_ERR_JOIN_FAIL	0x05
WIFI_ERR_NOT_JOINED	0x06
WIFI_ERR_LEAVE_FAILED	0x07
WIFI_COMMAND_PENDING	0x08
WIFI_NETWORK_UP	0x10
WIFI_NETWORK_DOWN	0x11
WIFI_FAIL	0xFF

Table 64 Return code for WIFI API

6.1 Turn on WiFi (WIFI_ON_REQ)

This command turns on Wifi on SN82xx. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Country code [2]

Country code is a 2-character ASCII string. Eg, “US” = the Uninited States. For the complete list, see Appendix A.

SN8200 supports both soft AP mode and STA mode at the same time. SN8200 has reserved flash space (NVM) to store startup parameters for both the soft AP and the STA. Only STA’s parameters can be dynamically changed at run time.

Turning on WiFi would cause the following to happen:

- The following operations occur using the parameters specified in the NVM if the AP mode is enabled.
  - Turn on the soft AP
  - Starts DNS server, DHCP server and HTTP server. The HTTP server provides a means for configuring the WLAN access parameters for the STA.
- Turn on the STA. If the NVM has valid startup parameters, the STA will try to join the saved SSID with saved authentication information. The NVM also stores whether DHCP or static IP is used for STA. If DHCP is used, DHCP client will be started. After a successful join, STA’s IP will be configured according to the NVM.

By default, the soft AP is turned on to allow user to use a WiFi enabled computer to connect to the soft AP, and instructs the STA to join one of the surrounding APs. So WiFi is turned on by default and this command is not required at startup.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	WIFI_ON_REQ	
1	Request Sequence	
2	Country code [0]	
3	Country code [1]	

Table 65 Turn on Wifi

Upon a successful reception of the command, the SN82xx turns on Wifi hardware. Command response is as follows:

Byte	Descriptions	Remark
0	WIFI_ON_RSP	
1	Request Sequence	
2	Status of operation: WIFI_SUCCESS or error code.	

Table 66 Response: Turn on Wifi

ERROR codes:
- WIFI_ERR_INIT_FAIL
- WIFI_ERR_UNKNOWN_COUNTRY
- WIFI_FAIL

6.2 Turn off WiFi (WIFI_OFF_REQ)

This command turns off Wifi on SN82xx. Turning off WiFi causes the following to happen:

- Turn off the soft AP, including shutting down DNS server, DHCP server and HTTP server.
- Disconnect STA from any joined network, and close all sockets opened by application.

Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	WIFI_OFF_REQ	
1	Request Sequence	

Table 67 Turn off Wifi

Upon a successful reception of the command, the SN82xx turns off Wifi hardware. Command response is as follows:

Byte	Descriptions	Remark
0	WIFI_OFF_RSP	
1	Request Sequence	
2	Status of operation: WIFI_SUCCESS or WIFI_FAIL	

Table 68 Response: Turn off Wifi

6.3 Soft AP on-off control (WIFI_AP_CTRL_REQ)

This command turns on or off the soft AP. The WIFI_ON(OFF)_REQ controls both the soft AP and STA at the same time, while this command only controls the soft AP. An example use case is, the soft AP (and its web server) is turned on at startup to configure STA to join a network and is no longer needed after the STA is connected. WIFI_AP_CTRL_REQ can be used to turn the soft AP off.

Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Onoff
UINT8 Persistency

OnOff =1 indicates turning on soft AP. Persistency=1 indicates the soft AP's on/off state will be saved in NVM. For example, if OnOff =0 and Persistency=1, the soft AP will not be turned on after a reset.

The SSID, security mode, DHCP, and country code for the soft AP use the values configured in NVM, so they are not in this command.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	WIFI_AP_CTRL_REQ	
1	Request Sequence	
2	Onoff (On=1, off=0)	
3	Persistency (Persistent=1, not persistent = 0)	

Table 69 AP on-off control

Upon a successful reception of the command, the SN82xx turns on/off Wifi soft AP. Command response is as follows:

Byte	Descriptions	Remark
0	WIFI_AP_CTRL_RSP	
1	Request Sequence	
2	Status of operation: WIFI_SUCCESS or WIFI_FAIL.	

Table 70 Response: AP on-off control

6.4 Associate to a network (WIFI_JOIN_REQ)

This command instructs SN82xx to associate to a network. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 SSID[up to 33]
UINT8 Security mode
UINT8 Security key length (0-64)
… Security key[ ]

Security modes are defined in the following table:

WIFI_SECURITY_OPEN	0x00
WEP (not supported in join request)	0x01
WIFI_SECURITY_WPA_TKIP_PSK	0x02
WIFI_SECURITY_WPA2_AES_PSK	0x04
WIFI_SECURITY_WPA2_MIXED_PSK	0x06

Table 71 Security mode for WIFI association

SSID is a character string up to 32 bytes in length with 1 additional byte for NUL-termination. The payload format is as follows:

Byte	Descriptions	Remark
0	WIFI_JOIN_REQ	
1	Request Sequence	
2…N	SSID[0] SSID[1]…NUL, N ≤ 35	Up to 33 octets including NUL termination
N+1	Security mode	
N+2	Security key length (0-64)	
…	Security key	Present only if key length > 0.

Table 72 Join network

Upon a successful reception of the command, the SN82xx tries to associate to a network. SSID and authentication parameters will be saved in NVM which will be used in subsequent power up (see 6.1).

Command response is as follows:

Byte	Descriptions	Remark
0	WIFI_JOIN_RSP	
1	Request Sequence	
2	Status of operation: WIFI_SUCCESS or error code.	

Table 73 Response: Join network

Error codes:

- WIFI_ERR_ALREADY_JOINED
- WIFI_ERR_AUTH_TYPE
- WIFI_ERR_JOIN_FAIL
- WIFI_FAIL

6.5 Disconnect from a network (WIFI_DISCONNECT_REQ)

This command instructs the SN82xx to disconnect from a network. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	WIFI_DISCONNECT_REQ	
1	Request Sequence	

Table 74 Disconnect from network

Upon a successful reception of the command, the SN82xx disconnects from associated network. Sockets opened by application are not closed.

Command response is as follows:

Byte	Descriptions	Remark
0	WIFI_DISCONNECT_RSP	
1	Request Sequence	
2	Status of operation: WIFI_SUCCESS or error code	

Table 75 Response: Disconnect from network

Error codes:

- WIFI_ERR_NOT_JOINED
- WIFI_ERR_LEAVE_FAILED

6.6 Get WiFi status (WIFI_GET_STATUS_REQ)

This command queries the WiFi status from SN82xx. This command should be called by application after startup to determine the WiFi status since the SN8200 may have joined an AP automatically based on NVM parameters (see 6.1).

Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Interface

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	WIFI_GET_STATUS_REQ	
1	Request Sequence	
2	Interface (0=STA 1=Soft AP)	

Table 76 Get WiFi status

Upon a successful reception of the command, the SN82xx returns the WiFi network status. Command response is as follows:

Byte	Descriptions	Remark
0	WIFI_GET_STATUS_RSP	

1	Request Sequence	
2	WiFi Status code	
3…8	MAC address	Present only if WiFi Status code is not WIFI_OFF.
9…	SSID [] (NULL terminated string, maximum 33bytes)	Present only if WiFi Status code is 2 or 3.

Table 77 Response: Get WiFi status

WiFi Status codes are listed below. Note: 0, 1 and 2 are applicable to STA. 0 and 3 are applicable to Soft AP.

- WIFI_OFF (0)
- NO_NETWORK (1)
- STA_JOINED (2)
- AP_STARTED (3)

6.7 Scan WiFi networks (WIFI_SCAN_REQ)

This command instructs the SN82xx to scan available networks. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT8 Scan Type
UINT8 BSS Type
UINT8 BSSID [6]
UINT8 Channel list []
UINT8 SSID[]

BSSID, Channel List, and SSID are optional fields. All 0's for BSSID, Channel list or SSID indicates it is not present.

- Scan Type: 0 = Active scan, 1= Passive scan
- BSS Type: 0 = Infrastructure, 1 = ad hoc, 2 = any
- BSSID: 6 bytes MAC address of the AP or STA. 6 bytes of 0's indicates it is not present.
- Channel list: 0 terminated array, up to 10 array elements.
- SSID: 0 terminated string for the AP or STA SSID, up to 33 bytes including NUL-termination.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	WIFI_SCAN_REQ	
1	Request Sequence	

2	Scan Type	
3	BSS Type	
4..9	BSSID	0's
10…	Channel List []	0
…	SSID[0] SSID[1] … NUL	0

Table 78 Scan networks

Upon a successful reception of the command, the SN82xx starts to scan. The response will indicate only WIFI_SUCCESS if no error. Actual scan result shall be sent from SN82xx as multiple indications defined in WIFI_SCAN_RESULT_IND (6.9). Command response is as follows:

Byte	Descriptions	Remark
0	WIFI_SCAN_RSP	
1	Request Sequence	
2	Status of operation: WIFI_SUCCESS or WIFI_FAIL.	

Table 79 Response: Scan networks

6.8 Get RSSI (WIFI_GET_STA_RSSI_REQ)

This command requests the reporting of the current RSSI from SN82xx's STA interface. Parameters are as follows:

UINT8 Request Sequence (0x00 – 0x7F)

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	WIFI_GET_STA_RSSI_REQ	
1	Request Sequence	

Table 80 Get RSSI

Upon a successful reception of the command, the SN82xx returns the current RSSI. If the returning RSSI is 0, it means STA is not in joined state or STA interface is not up. Command response is as follows:

Byte	Descriptions	Remark
0	WIFI_GET_STA_RSSI_RSP	
1	Request Sequence	
2	RSSI (signed char)	RSSI in dBm

Table 81 Response: Get RSSI

6.9 Scan result indication (WIFI_SCAN_RESULT_IND)

Scan result is sent from SN82xx to host application using multiple WIFI_SCAN_RESULT_IND indications. Parameters are as follows:

UINT8 Indication Sequence
UINT8 Number of SSID records
SSID records

Host application shall continue to receive scan results as long as the Number of SSID record is non-zero. The payload of the indication is as follows:

Byte	Descriptions	Remark
0	WIFI_SCAN_RESULT_IND	
1	Indication Sequence	
2	Number of SSID records	
…	SSID records	Present only if Number of SSID records is non-zero.

SSID record is defined as follows. If the SSID is hidden, the SSID field shall be all 0's, and the number of 0's equals the length of the actual SSID plus 1 for the terminating 0.

Byte	Descriptions	Remark
0	Channel	
1	RSSI (signed char)	
2	Security mode (Table 71)	
3…8	BSSID [6]	
9	Network Type (0=Infrastructure, 1=ad hoc)	
10	Max Data Rate (Mbps)	
11	Reserved	
12 …	SSID [ ] (NULL terminated string, maximum 33 bytes)	

Table 82 Indication: SSID record

Upon a successful reception of the indication, the host application sends to the SN82xx the following reception confirmation. Indication Sequence is copied from the received indication.

Byte	Descriptions

0	WIFI_SCAN_RESULT_CFM
1	Indication Sequence

Table 83 Confirm: Scan result indication

6.10 WIFI network status indication (WIFI_NETWORK_STATUS_IND)

Indication is originated from the SN82xx and sent to the host application.

This event describes the status of WIFI network connection for an interface. Currently only the STA status is reported. Soft AP is always UP. The indication has the following parameters:

UINT8 Indication Sequence (0x00 - 0x7F)
UINT8 Interface
UINT8 Status code
UINT8 SSID[up to 33 including NUL-termination]

Status codes are:

- WIFI_ NETWORK _UP
- WIFI_ NETWORK _DOWN

All multi-byte values are transferred in big endian. The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	WIFI_NETWORK_STATUS_IND	
1	Indication Sequence	
2	Interface (0=STA 1=soft AP)	
3	Status code	
4…	SSID[0] SSID[1] … NUL	0

Table 84 WIFI connection status indication

SSID is a character string up to 32 bytes in length with 1 additional byte for NUL-termination. Upon a successful reception of the report, the host application sends to the SN82xx the following reception confirmation. Indication Sequence is copied from the received indication.

Byte	Descriptions
0	WIFI_NETWORK_STATUS_CFM
1	Indication Sequence

Table 85 Confirm: WIFI connection status indication

7. IO and Peripheral API (CMD_ID_IO)

The Command ID for controlling and accessing IOs and peripherals is CMD_ID_IO (0x03). The specifics of the command are defined in the payload field of the frame. All IO and peripheral commands have the following format:

UINT8	SN8200 IO and peripheral operation (RFSCID)
…	Rest of payload depends on RFSCID

The first byte of the payload of this command describes the IO and peripheral operation, and it contains the Response Flag and Sub-Command ID (RFSCID). The MSB of RFSCID is the response flag, with values '0' and '1' indicating command request and its response, respectively. The remaining 7 LSBs of RFSCID is the Sub-Command ID (SCID) specifying the IO and peripheral operation to perform. The following SCIDs are defined.

SCID	Value	Description
IO_I2C_INIT_REQ	0x00	Initialize I^2C interface
IO_I2C_READ_REQ	0x01	Read data from I^2C device
IO_I2C_WRITE_REQ	0x02	Write data to I^2C device
IO_I2C_WRITE_TO_READ_REQ	0x03	Write data to I^2C device followed by read command
IO_GPIO_CONFIG_REQ	0x04	Configure GPIO pins
IO_GPIO_WRITE_REQ	0x05	Set GPIO pin outputs
IO_GPIO_READ_REQ	0x06	Read GPIO pin inputs
IO_GPIO_INT_IND	0x07	GPIO interrupt indication

Table 86 SCIDs for IO and peripheral API

Value	Status	Description
0	IO_I2C_SUCCESS IO_GPIO_SUCCESS	operation success
1	IO_I2C_FAILED IO_GPIO_FAILED	operation failure not specified below
2	IO_ERR_I2C_IB	insufficient buffer to handle request
3	IO_ERR_I2C_AL	arbitration lost
4	IO_ERR_I2C_AF	acknowledgement failure
5	IO_ERR_I2C_FE	bad start/stop
6	IO_ERR_I2C_OR	overrun/underrun error

Table 87 I2C status code

7.1 Initialize I²C interface (IO_I2C_INIT_REQ)

This command initializes and configures I²C interface for master mode operation. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT32 I2C clock rate (Hz)
UINT16 Own Address
UINT16 Control

Clock is I²C clock rate in Hz. Setting Clock = 0 disables the interface, in which case Own Address and Control fields are ignored by SN8200.

Own address is I²C master's own address. It is defined as follows.

b_{15}: 0: 7-bit address mode; 1: 10-bit address mode

- b_6-b_0: 7-bit address
- b_9-b_0: 10-bit address

others: reserved

Control is defined as follows.

b_{14}: Duty Cycle, only for fast mode (Clock > 100 kHz). It is ignored in standard mode. Set it to 0 in I²C for $t_{low}/t_{high} = 2$, 1 for $t_{low}/t_{high} = 16/9$. Default is 0.

Others: 0 (reserved)

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	IO_I2C_INIT_REQ (0x00)	
1	Request Sequence	
2	MSB Clock	
3	…	
4	…	
5	LSB Clock	
6	MSB Own Address	
7	LSB Own Address	
8	MSB Control	
9	LSB Control	

Table 88 Initialize I²C interface

The format for the SN8200 response is as follows:

Byte	Descriptions

0	IO_I2C_INIT_RSP (0x80)
1	Request Sequence
2	Status of operation: IO_I2C_SUCCESS IO_I2C_FAILED
3	MSB Max Write Payload length
4	LSB Max Write Payload length
5	MSB Max Read Payload length
6	LSB Max Read Payload length

Table 89 Response: Initialize I^2C interface

Max Write Buffer size defines the maximum buffer space available for the I2C write operation.

Max Read Buffer size defines the maximum buffer space available for the I2C read operation.

7.2 Read Data from I^2C Device (IO_I2C_READ_REQ)

This command reads data from an I^2C device. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT16 Payload Length
UINT16 Slave Address

Payload Length is the total number of bytes to be received following the command.

The Slave Address field is defined as follows.

- For 7-bit slave address:

  b_{15}-b_8: 0
  b_7-b_1: 7-bit address
  b_0: 0 (reserved)

- For 10-bit slave address:

  b_{15}-b_{11} 11110
  b_{10}: a_9
  b_9: a_8
  b_8: 0 (reserved)
  b_7-b_0: a_7-a_0

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	IO_I2C_READ_REQ (0x01)	
1	Request Sequence	
2	MSB Payload Length	
3	LSB Payload Length	

4	MSB Slave Address	
5	LSB Slave Address	

Table 90 Read data from I²C device

The format for the SN8200 response is as follows:

Byte	Descriptions
0	IO_I2C_READ_RSP (0x81)
1	Request Sequence
2	Status of operation: IO_I2C_SUCCESS IO_I2C_FAILED IO_ERR_I2C_IB IO_ERR_I2C_AL IO_ERR_I2C_AF IO_ERR_I2C_FE IO_ERR_I2C_OR
3	MSB Payload length
4	LSB Payload length
…	Payload of length Payload Length

Table 91 Response: Read data from I²C device

7.3 Write Data to I²C Device (IO_I2C_WRITE_REQ)

This command writes data to an I²C device. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT16 Payload Length
UINT16 Slave Address
UINT8[]Payload, octets to write as specified by Payload Length

Payload Length is the total number of bytes to send to the Slave Address.

The Slave Address follows the same definition in the previous section.

Payload specifies the Payload Length octets to write.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	IO_I2C_WRITE_REQ (0x02)	
1	Request Sequence	
2	MSB Payload Length	

3	LSB Payload Length	
4	MSB Slave Address	
5	LSB Slave Address	
…	Payload of length Payload Length	

Table 92 Write data to I²C device

The format for the SN8200 response is as follows:

Byte	Descriptions
0	IO_I2C_WRITE_RSP (0x82)
1	Request Sequence
2	Status of operation: IO_I2C_SUCCESS IO_I2C_FAILED IO_ERR_I2C_IB IO_ERR_I2C_AL IO_ERR_I2C_AF IO_ERR_I2C_FE IO_ERR_I2C_OR

Table 93 Response: Write data to I²C device

7.4 Write Data to I²C Device followed by read (IO_I2C_WRITE_TO_READ_REQ)

This command writes data to an I²C device then read a specified amount of data back. The read operation is only performed if the write is successful. Parameters are as follows:

UINT8 Request Sequence (0x00 - 0x7F)
UINT16 Payload Length (n)
UINT16 Slave Address
UINT8[] Payload octets to write as specified by Payload Length
UINT16 Read Payload Length

Payload Length is the total number of bytes to send to the Slave Address.

The Slave Address follows the same definition in the previous section.

Payload specifies the Payload Length octets to write.

Read Payload Length is the total number of bytes to be received following the write command.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	IO_I2C_WRITE_TO_READ_REQ (0x03)	
1	Request Sequence	
2	MSB Payload Length	
3	LSB Payload Length	
4	MSB Slave Address	
5	LSB Slave Address	
…	Payload of length Payload Length (n)	
n+6	MSB Read Payload Length	
n+7	LSB Read Payload Length	

Table 8 Write data to I^2C device followed by read command

The format for the SN8200 response is as follows:

Byte	Descriptions
0	IO_I2C_WRITE_TO_READ_RSP (0x83)
1	Request Sequence
2	Status of write operation: IO_I2C_SUCCESS IO_I2C_FAILED IO_ERR_I2C_IB IO_ERR_I2C_AL IO_ERR_I2C_AF IO_ERR_I2C_FE IO_ERR_I2C_OR
3	Status of read operation: IO_I2C_SUCCESS IO_I2C_FAILED IO_ERR_I2C_IB IO_ERR_I2C_AL IO_ERR_I2C_AF IO_ERR_I2C_FE IO_ERR_I2C_OR
4	MSB Read Payload length
5	LSB Read Payload length
…	Payload read

Table 9 Response: Write data to I^2C device followed by read command

7.5 Configure GPIO pins (IO_GPIO_CONFIG_REQ)

This command configures GPIO pins. Parameters are as follows:

UINT32 GPIO ID Mask
UINT8 Mode
UINT8 Interrupt Trigger

GPIO ID Mask specifies the GPIOs to configure. IO configuration as well as read/write/trigger uses this 32-bit mask so multiple IOs can be accessed sequentially from IO0 to IO19.

Pin Number	I/O ID	I/O ID Mask	SN8200 Pin Description
5	0	0x00000001	Reserved
6	1	0x00000002	GPIO
7	2	0x00000004	GPIO
9	3	0x00000008	GPIO
10	4	0x00000010	GPIO
11	5	0x00000020	GPIO
12	6	0x00000040	GPIO
32	7	0x00000080	UART_TX
33	8	0x00000100	UART_RX
34	9	0x00000200	UART_CTS
35	10	0x00000400	UART_RTS
36	11	0x00000800	JTMS
37	12	0x00001000	SPI_NSS/JTDI
38	13	0x00002000	JTCK
40	14	0x00004000	SPI_SCK/JTDO
41	15	0x00008000	SPI_MISO/JTRST
42	16	0x00010000	SPI_MOSI
43	17	0x00020000	I2C_SCL
44	18	0x00040000	I2C_SDA
46	19	0x00080000	GPIO

Table 94 GPIO ID

Each of the GPIO pins can be configured by GPIO Mode as output (push-pull or open-drain) or as input (with or without pull-up or pull-down).

GPIO Mode is defined as follows:

GPIO_Mode_IN_FLOATING	0x04
GPIO_Mode_IN_PULL_DN	0x28
GPIO_Mode_IN_PULL_UP	0x48
GPIO_Mode_OUT_OPEN_DR	0x14
GPIO_Mode_OUT_PUSH_PULL	0x10

Table 95 GPIO Mode

Interrupt trigger is used to monitor and report asynchronous IO changes. SN8200 reports occurrences of the specified trigger by IO_GPIO_INT_IND (Section 7.8). It is defined as follows:

GPIO_Trigger_Rising	0x08
GPIO_Trigger_Falling	0x0C
GPIO_Trigger_Rising_Falling	0x10

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	IO_GPIO_CONFIG_REQ (0x04)	
1	Request Sequence	
2	MSB GPIO ID Mask	
3	…	
4	…	
5	LSB GPIO ID Mask	
6	Mode	
7	Interrupt	

Table 96 Configure GPIO pins

The format for the SN8200 response is as follows:

Byte	Descriptions
0	IO_GPIO_CONFIG_RSP (0x84)

1	Request Sequence
2	Status of operation: IO_GPIO_SUCCESS IO_GPIO_FAILED

Table 97 Response: Configure GPIO pins

7.6 Write GPIO output pins (IO_GPIO_WRITE_REQ)

This command writes GPIO output pins. Parameters are as follows:

UINT32 GPIO ID Mask
UINT32 Value

GPIO ID Mask specifies the GPIOs to write. Multiple IOs can be accessed sequentially from IO0 (b0) to IO19 (b19). Value specifies the state of the GPIO pins, so if bit (bi) is '1' in GPIO ID Mask, the bit (bi) of Value specifies the value to output for that GPIO. For example, ID mask of 0xA with Value of 0x8 sets GPIO1 and 3 to '0' and '1', respectively.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	IO_GPIO_WRITE_REQ (0x05)	
1	Request Sequence	
2	MSB GPIO ID Mask	
3	…	
4	…	
5	LSB GPIO ID Mask	
2	MSB Value	
3	…	
4	…	
5	LSB Value	

Table 98 Write GPIO output pins

The format for the SN8200 response is as follows:

Byte	Descriptions
0	IO_GPIO_WRITE_RSP (0x85)
1	Request Sequence
2	Status of operation: IO_GPIO_SUCCESS

	IO_GPIO_FAILED

Table 99 Response: Write GPIO output pins

7.7 Read GPIO pin inputs (IO_GPIO_READ_REQ)

This command reads values from either input or output GPIO pins. Its parameters are as follows:

UINT32 GPIO ID Mask

GPIO ID Mask specifies the GPIOs to read. Multiple IOs can be accessed sequentially from IO0 (b0) to IO19 (b19), e.g., GPIO ID mask of 0xA reads GPIO1 followed by GPIO3.

The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	IO_GPIO_ READ_REQ (0x06)	
1	Request Sequence	
2	MSB GPIO ID Mask	
3	…	
4	…	
5	LSB GPIO ID Mask	

Table 100 Read GPIO pins

The format for the SN8200 response is as follows. Value specifies the state of the GPIO pins, so if bit (bi) is '1' in GPIO ID Mask, the bit (bi) of Value specifies the value to output for that GPIO.

Byte	Descriptions
0	IO_GPIO_READ_RSP (0x86)
1	Request Sequence
2	Status of operation: IO_GPIO_SUCCESS IO_GPIO_FAILED
3	MSB Value
4	…
5	…
6	LSB Value

Table 101 Response: Read GPIO pin inputs

7.8 GPIO interrupt indication (IO_GPIO_INT_IND)

This indication is sent to host when a GPIO interrupt specified by the GPIO configuration has occurred.

UINT32 GPIO ID Mask
UINT32 Value

Multiple interrupts may be reported for the specified GPIOs. However, there is no implied relative timing between the reported pins. Value specifies the state of the GPIO pins, so if bit (bi) is '1' in GPIO ID Mask, the bit (bi) of Value specifies the value to output for that GPIO. The payload format is as follows:

Byte	Descriptions	Set to ignore field
0	IO_GPIO_INT_IND (0x07)	
1	Indication Sequence	
2	MSB GPIO ID Mask	
3	…	
4	…	
5	LSB GPIO ID Mask	
6	MSB Value	
7	…	
8	…	
9	LSB Value	

Table 102 GPIO interrupt indication

8. Use cases

This section describes command sequences for sample usage cases.

8.1 WiFi network station control

The following commands are used to turn on the WiFi station and then joining a specified AP.

- WIFI_GET_STATUS_REQ
- (Optional) WIFI_ON_REQ to turn on the WiFi station
- (Optional) WIFI_JOIN_REQ to join specified AP

SN8200 supports soft AP function, the default firmware turns on WiFi to start the soft AP at power up. So the WIFI_ON_REQ is not required at startup. See Section 6.1 for details. There are three ways for the SN8200 to join an AP.

1. Auto-join by startup parameters saved in NVM at startup.
2. Use soft AP provided web page to scan and join AP using an external web browser.
3. Use WIFI_JOIN_REQ command to join an AP if there is no valid NVM startup parameters or if the SSID needs to be changed.

So WIFI_GET_STATUS_REQ command is recommended to first determine the WiFi status. If WiFi is off, then WIFI_ON_REQ command is needed. See Section 6.1 for consequences of this command. After turning on WiFi, WIFI_GET_STATUS_REQ should be used again to get the status, and then decide whether one of the above three ways is needed to join a network.

Once SN8200 has successfully joined a WiFi network, the SNIC IP networking services may be invoked to perform TCP/IP communications. See Section 8.2 for details.

The station can leave the WiFi network anytime. The following sequence is used to leave a network and then shut down the WiFi station.

- WIFI_DISCONNECT_REQ to disassociate from the AP and leave the current network. WIFI_JOIN_REQ can then optionally be used to rejoin the same AP or join a different AP.
- WIFI_OFF_REQ to turn off the WiFi station after disassociating from the AP. The soft AP is also turned off after this command.

8.2 SNIC TCP/IP services

The SNIC IP networking services are supported by a socket-like interface over the serial port. The following network features are supported:

- ICMP (Ping response)
- DHCP (Client)
- ARP
- TCP
- UDP
- DNS resolver

When enabled, the soft AP supports the following features:

- DHCP (server)
- DNS (server)
- HTTP (server)

Before using any TCP/UDP services, the following commands must be invoked to turn on the API and configure the IP address for SN8200.

- SNIC_INIT_REQ to initialize SNIC IP networking API
- SNIC_GET_DHCP_INFO_REQ to retrieve the node's MAC address, IP address, gateway address and subnet mask. IP address may have been obtained automatically by the NVM startup parameters.
- (Optional) SNIC_IP_CONFIG_REQ to configure SN8200 to use static or dynamic IP addresses. This command is only needed when SNIC_GET_DHCP_INFO_REQ does not return with valid IP information, or if IP configuration change is required.

At any time after successful IP address configuration, the following commands may be used to query IP networking information:

- (Optional) SNIC_SEND_ARP_REQ to determine the link level address for a given IP address
- (Optional) SNIC_RESOLVE_NAME_REQ to resolve a host name to IP address

Once SN8200 has successfully obtained an IP address, IP networking through TCP and UDP can be supported. The following scenarios are described in this section:

- TCP server (Section 8.2.1)
- TCP client (Section8.2.2)
- UDP server (Section 8.2.3)
- UDP client (Section 8.2.4)

Upon termination of IP networking functions and after all opened sockets have been closed, the following is invoked to shutdown the IP networking API.

- SNIC_CLEANUP_REQ to terminate the SNIC IP networking API

8.2.1 TCP server

The following lists a typical command sequence for a TCP server session.

- Configure WiFi network access (Section 8.1) and obtain an IP address (Section 8.2)
  - WIFI_GET_STATUS_REQ
  - (Optional) WIFI_ON_REQ
  - (Optional) WIFI_JOIN_REQ
  - SNIC_INIT_REQ
  - SNIC_GET_DHCP_INFO_REQ
  - (Optional) SNIC_IP_CONFIG_REQ
- SNIC_TCP_CREATE_SOCKET_REQ to create a TCP socket
- SNIC_TCP_CREATE_CONNECTION_REQ to listen for incoming connections
  - Should receive SNIC_TCP_CLIENT_SOCKET_IND for any new client connections.

    - Should receive SNIC_CONNECTION_RECV_IND if any client sends a TCP packet.
- SNIC_SEND_FROM_SOCKET_REQ to send a TCP packet to a client
- SNIC_CLOSE_SOCKET_REQ to terminate a TCP connection and close the socket

If application exits, the follow commands should be used to stop the SNIC API, exit the WLAN and shutdown the WiFi interface (applicable to all use cases):

- SNIC_CLEANUP_REQ
- WIFI_DISCONNECT_REQ
- WIFI_OFF_REQ

8.2.2 TCP client

The following lists a typical command sequence for a TCP client session.

- Configure WiFi network access (Section 8.1) and obtain an IP address (Section 8.2)
    - WIFI_GET_STATUS_REQ
    - (Optional) WIFI_ON_REQ
    - (Optional) WIFI_JOIN_REQ
    - SNIC_INIT_REQ
    - SNIC_GET_DHCP_INFO_REQ
    - (Optional) SNIC_IP_CONFIG_REQ
- SNIC_TCP_CREATE_SOCKET_REQ to create a TCP socket
- (Optional) SNIC_RESOLVE_NAME_REQ to resolve host name to IP address
- SNIC_TCP_CONNECT_TO_SERVER_REQ to connect to the TCP server
    - Should receive SNIC_TCP_CONNECTION_STATUS_IND if connection to server changes status.
    - Should receive SNIC_CONNECTION_RECV_IND if server sends any TCP packet.
- SNIC_SEND_FROM_SOCKET_REQ to send a TCP packet to server
- SNIC_CLOSE_SOCKET_REQ to terminate a TCP connection and close the socket

8.2.3 UDP server

The following lists a typical command sequence for a UDP server session.

- Configure WiFi network access (Section 8.1) and obtain an IP address (Section 8.2)
    - WIFI_GET_STATUS_REQ
    - (Optional) WIFI_ON_REQ
    - (Optional) WIFI_JOIN_REQ
    - SNIC_INIT_REQ
    - SNIC_GET_DHCP_INFO_REQ
    - (Optional) SNIC_IP_CONFIG_REQ
- SNIC_UDP_CREATE_SOCKET_REQ to create an UDP socket

- SNIC_UDP_START_RECV_REQ to receive any incoming UDP packet
    - Should receive SNIC_UDP_RECV_IND if peer sends a packet but the peer is not the client of a UDP connection
    - Should receive SNIC_CONNECTION_RECV_IND if client of a UDP connection sends a packet
- SNIC_UDP_SEND_FROM_SOCKET_REQ to send UDP packet to peer and optionally setting up connection mode.
- SNIC_SEND_FROM_SOCKET_REQ (optional) to send UDP packet to UDP client operating in connection mode
- SNIC_CLOSE_SOCKET_REQ to terminate any UDP connection and close the socket

8.2.4 UDP client

The UDP interface supports transient UDP sockets and persistent UDP sockets using connected and non-connected mode. In all cases, the first steps are to configure WiFi network access (Section 8.1) and obtain an IP address (Section 8.2)

- WIFI_GET_STATUS_REQ
- (Optional) WIFI_ON_REQ
- (Optional) WIFI_JOIN_REQ
- SNIC_INIT_REQ
- SNIC_GET_DHCP_INFO_REQ
- (Optional) SNIC_IP_CONFIG_REQ

For a UDP client using persistent socket, the following sequence applies:

- SNIC_UDP_CREATE_SOCKET_REQ to create an UDP socket
- SNIC_RESOLVE_NAME_REQ (optional) to resolve host name to IP address
- SNIC_UDP_SEND_FROM_SOCKET_REQ to send UDP packet to peer and optionally setting up connection mode.
- SNIC_SEND_FROM_SOCKET_REQ (optional) to send UDP packet to UDP server operating in connection mode.
- SNIC_CLOSE_SOCKET_REQ to terminate any UDP connection and close the socket

For a UDP client using transient socket, the following sequence applies:

- SNIC_UDP_SIMPLE_SEND_REQ to create a transient socket, send an UDP packet and close the socket

9. Appendix A (Country code)

{"AFGHANISTAN", "AF"},
{"ALBANIA", "AL"},
{"ALGERIA", "DZ"},
{"AMERICAN SAMOA", "AS"},
{"ANDORRA", "AD"},
{"ANGOLA", "AO"},
{"ANGUILLA", "AI"},
{"ANTARCTICA", "AQ"},
{"ANTIGUA AND BARBUDA", "AG"},
{"ARGENTINA", "AR"},
{"ARMENIA", "AM"},
{"ARUBA", "AW"},
{"ASCENSION ISLAND", "AC"},
{"AUSTRALIA", "AU"},
{"AUSTRIA", "AT"},
{"AZERBAIJAN", "AZ"},
{"BAHAMAS", "BS"},
{"BAHRAIN", "BH"},
{"BANGLADESH", "BD"},
{"BARBADOS", "BB"},
{"BELARUS", "BY"},
{"BELGIUM", "BE"},
{"BELIZE", "BZ"},
{"BENIN", "BJ"},
{"BERMUDA", "BM"},
{"BHUTAN", "BT"},
{"BOLIVIA", "BO"},
{"BOSNIA AND HERZEGOVINA", "BA"},
{"BOTSWANA", "BW"},
{"BOUVET ISLAND", "BV"},
{"BRAZIL", "BR"},
{"BRITISH INDIAN OCEAN TERRITORY", "IO"},
{"BRUNEI DARUSSALAM", "BN"},

```
{"BULGARIA",       "BG"},
{"BURKINA FASO",   "BF"},
{"BURUNDI",    "BI"},
{"CAMBODIA",       "KH"},
{"CAMEROON",       "CM"},
{"CANADA",     "CA"},
{"CAPE VERDE",     "CV"},
{"CAYMAN ISLANDS",  "KY"},
{"CENTRAL AFRICAN REPUBLIC",       "CF"},
{"CHAD",       "TD"},
{"CHILE",      "CL"},
{"CHINA",      "CN"},
{"CHRISTMAS ISLAND",   "CX"},
{"CLIPPERTON ISLAND",   "CP"},
{"COCOS (KEELING) ISLANDS",    "CC"},
{"COLOMBIA",       "CO"},
{"COMOROS",    "KM"},
{"CONGO",      "CG"},
{"CONGO, THE DEMOCRATIC REPUBLIC OF THE",      "CD"},
{"COOK ISLANDS",   "CK"},
{"COSTA RICA",     "CR"},
{"COTE D'IVOIRE",   "CI"},
{"CROATIA",    "HR"},
{"CUBA",       "CU"},
{"CYPRUS",     "CY"},
{"CZECH REPUBLIC",  "CZ"},
{"DENMARK",    "DK"},
{"DJIBOUTI",       "DJ"},
{"DOMINICA",       "DM"},
{"DOMINICAN REPUBLIC",  "DO"},
{"ECUADOR",    "EC"},
{"EGYPT",      "EG"},
{"EL SALVADOR",    "SV"},
{"EQUATORIAL GUINEA",   "GQ"},
{"ERITREA",    "ER"},
{"ESTONIA",    "EE"},
```

```
{"ETHIOPIA",       "ET"},
{"FALKLAND ISLANDS (MALVINAS)",     "FK"},
{"FAROE ISLANDS",   "FO"},
{"FIJI",       "FJ"},
{"FINLAND",     "FI"},
{"FRANCE",      "FR"},
{"FRENCH GUIANA",   "GF"},
{"FRENCH POLYNESIA",    "PF"},
{"FRENCH SOUTHERN TERRITORIES",     "TF"},
{"GABON",       "GA"},
{"GAMBIA",      "GM"},
{"GEORGIA",     "GE"},
{"GERMANY",     "DE"},
{"GHANA",       "GH"},
{"GIBRALTAR",       "GI"},
{"GREECE",      "GR"},
{"GREENLAND",       "GL"},
{"GRENADA",     "GD"},
{"GUADELOUPE",      "GP"},
{"GUAM",        "GU"},
{"GUATEMALA",       "GT"},
{"GUERNSEY",        "GG"},
{"GUINEA",      "GN"},
{"GUINEA-BISSAU",   "GW"},
{"GUYANA",      "GY"},
{"HAITI",       "HT"},
{"HEARD ISLAND AND MCDONALD ISLANDS",       "HM"},
{"HOLY SEE (VATICAN CITY STATE)",       "VA"},
{"HONDURAS",        "HN"},
{"HONG KONG",       "HK"},
{"HUNGARY",     "HU"},
{"ICELAND",     "IS"},
{"INDIA",       "IN"},
{"INDONESIA",       "ID"},
{"IRAN, ISLAMIC REPUBLIC OF",       "IR"},
{"IRAQ",        "IQ"},
```

{"IRELAND", "IE"},
{"ISRAEL", "IL"},
{"ITALY", "IT"},
{"JAMAICA", "JM"},
{"JAPAN", "JP"},
{"JERSEY", "JE"},
{"JORDAN", "JO"},
{"KAZAKHSTAN", "KZ"},
{"KENYA", "KE"},
{"KIRIBATI", "KI"},
{"KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF", "KP"},
{"KOREA, REPUBLIC OF", "KR"},
{"KUWAIT", "KW"},
{"KYRGYZSTAN", "KG"},
{"LAO PEOPLE'S DEMOCRATIC REPUBLIC", "LA"},
{"LATVIA", "LV"},
{"LEBANON", "LB"},
{"LESOTHO", "LS"},
{"LIBERIA", "LR"},
{"LIBYAN ARAB JAMAHIRIYA", "LY"},
{"LIECHTENSTEIN", "LI"},
{"LITHUANIA", "LT"},
{"LUXEMBOURG", "LU"},
{"MACAO", "MO"},
{"MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF", "MK"},
{"MADAGASCAR", "MG"},
{"MALAWI", "MW"},
{"MALAYSIA", "MY"},
{"MALDIVES", "MV"},
{"MALI", "ML"},
{"MALTA", "MT"},
{"MAN, ISLE OF", "IM"},
{"MARSHALL ISLANDS", "MH"},
{"MARTINIQUE", "MQ"},
{"MAURITANIA", "MR"},
{"MAURITIUS", "MU"},

```
{"MAYOTTE",     "YT"},
{"MEXICO",      "MX"},
{"MICRONESIA, FEDERATED STATES OF",     "FM"},
{"MOLDOVA, REPUBLIC OF",        "MD"},
{"MONACO",      "MC"},
{"MONGOLIA",        "MN"},
{"MONTENEGRO",      "ME"},
{"MONTSERRAT",      "MS"},
{"MOROCCO",     "MA"},
{"MOZAMBIQUE",      "MZ"},
{"MYANMAR",     "MM"},
{"NAMIBIA",     "NA"},
{"NAURU",       "NR"},
{"NEPAL",       "NP"},
{"NETHERLANDS",     "NL"},
{"NETHERLANDS ANTILLES",        "AN"},
{"NEW CALEDONIA",   "NC"},
{"NEW ZEALAND",     "NZ"},
{"NICARAGUA",       "NI"},
{"NIGER",       "NE"},
{"NIGERIA",     "NG"},
{"NIUE",        "NU"},
{"NORFOLK ISLAND",      "NF"},
{"NORTHERN MARIANA ISLANDS",        "MP"},
{"NORWAY",      "NO"},
{"OMAN",        "OM"},
{"PAKISTAN",        "PK"},
{"PALAU",       "PW"},
{"PALESTINIAN TERRITORY, OCCUPIED",     "PS"},
{"PANAMA",      "PA"},
{"PAPUA NEW GUINEA",    "PG"},
{"PARAGUAY",        "PY"},
{"PERU",        "PE"},
{"PHILIPPINES",     "PH"},
{"PITCAIRN",        "PN"},
{"POLAND",      "PL"},
```

```
{"PORTUGAL",       "PT"},
{"PUERTO RICO",    "PR"},
{"QATAR",      "QA"},
{"REUNION",    "RE"},
{"ROMANIA",    "RO"},
{"RUSSIAN FEDERATION",  "RU"},
{"RWANDA",     "RW"},
{"SAINT HELENA",   "SH"},
{"SAINT KITTS AND NEVIS",      "KN"},
{"SAINT LUCIA",    "LC"},
{"SAINT PIERRE AND MIQUELON",      "PM"},
{"SAINT VINCENT AND THE GRENADINES",       "VC"},
{"SAMOA",      "WS"},
{"SAN MARINO",     "SM"},
{"SAO TOME AND PRINCIPE",      "ST"},
{"SAUDI ARABIA",   "SA"},
{"SENEGAL",    "SN"},
{"SERBIA",     "RS"},
{"SEYCHELLES",     "SC"},
{"SIERRA LEONE",   "SL"},
{"SINGAPORE",      "SG"},
{"SLOVAKIA",       "SK"},
{"SLOVENIA",       "SI"},
{"SOLOMON ISLANDS", "SB"},
{"SOMALIA",    "SO"},
{"SOUTH AFRICA",   "ZA"},
{"SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS",       "GS"},
{"SPAIN",      "ES"},
{"SRI LANKA",      "LK"},
{"SUDAN",      "SD"},
{"SURINAME",       "SR"},
{"SVALBARD AND JAN MAYEN",     "SJ"},
{"SWAZILAND",      "SZ"},
{"SWEDEN",     "SE"},
{"SWITZERLAND",    "CH"},
{"SYRIAN ARAB REPUBLIC",       "SY"},
```

```
{"TAIWAN, PROVINCE OF CHINA",      "TW"},
{"TAJIKISTAN",     "TJ"},
{"TANZANIA, UNITED REPUBLIC OF",       "TZ"},
{"THAILAND",       "TH"},
{"TIMOR-LESTE (EAST TIMOR)",       "TL"},
{"TOGO",       "TG"},
{"TOKELAU",    "TK"},
{"TONGA",      "TO"},
{"TRINIDAD AND TOBAGO", "TT"},
{"TRISTAN DA CUNHA",   "TA"},
{"TUNISIA",    "TN"},
{"TURKEY",     "TR"},
{"TURKMENISTAN",   "TM"},
{"TURKS AND CAICOS ISLANDS",       "TC"},
{"TUVALU",     "TV"},
{"UGANDA",     "UG"},
{"UKRAINE",    "UA"},
{"UNITED ARAB EMIRATES",       "AE"},
{"UNITED KINGDOM",  "GB"},
{"UNITED STATES",   "US"},
{"UNITED STATES MINOR OUTLYING ISLANDS",       "UM"},
{"URUGUAY",    "UY"},
{"UZBEKISTAN",     "UZ"},
{"VANUATU",    "VU"},
{"VENEZUELA",      "VE"},
{"VIET NAM",       "VN"},
{"VIRGIN ISLANDS, BRITISH",    "VG"},
{"VIRGIN ISLANDS, U.S.",       "VI"},
{"WALLIS AND FUTUNA",   "WF"},
{"WESTERN SAHARA",  "EH"},
{"YEMEN",      "YE"},
{"YUGOSLAVIA",     "YU"},
{"ZAMBIA",     "ZM"},
{"ZIMBABWE",       "ZW"},
```

(END)