NeoCortec Arduino Library

Generated by Doxygen 1.9.7

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 NeoMesh

Classes

- struct NcSetting
- class NeoMesh

  Object that handles connection to NeoCortec module.
- struct tNcSapiMessage
- class SAPIParser

  Class to parse messages from system interface.

Macros

- #define **DEFAULT_NEOCORTEC_BAUDRATE** 115200
- #define **SAPI_COMMAND_HEAD** 0x3E
- #define **SAPI_COMMAND_TAIL** 0x21
- #define **SAPI_COMMAND_LOGIN1** 0x01
- #define **SAPI_COMMAND_LOGIN2** 0x03
- #define **SAPI_COMMAND_START_BOOTLOADER1** 0x01
- #define **SAPI_COMMAND_START_BOOTLOADER2** 0x13
- #define **SAPI_COMMAND_GET_SETTING_FLASH1** 0x01
- #define **SAPI_COMMAND_GET_SETTING_FLASH2** 0x06
- #define **SAPI_COMMAND_SET_SETTING1** 0x01
- #define **SAPI_COMMAND_SET_SETTING2** 0x0A
- #define **SAPI_COMMAND_COMMIT_SETTINGS1** 0x01
- #define **SAPI_COMMAND_COMMIT_SETTINGS2** 0x08
- #define **SAPI_COMMAND_START_PROTOCOL1** 0x01
- #define **SAPI_COMMAND_START_PROTOCOL2** 0x12
- #define **NODE_ID_SETTING** 0xA
- #define **NETWORK_ID_SETTING** 0x2A
- #define **TRACE_OUTPUT_SETTING** 0x2C
- #define **GENERIC_APPLICATION_NORM_SETTING** 0x19
- #define **GENERIC_APPLICATION_ALT_SETTING** 0x3A
- #define **DEFAULT_PASSWORD_LVL10** {0x4c, 0x76, 0x6c, 0x31, 0x30}
- #define **SAPI_COMMAND_HEADER** 0x3E
- #define **SAPI_COMMAND_TAIL** 0x21
- #define **MINIMUM_MESSAGE_LENGTH** 5

Typedefs

- typedef void(∗ NeoMeshReadCallback) (uint8_t ∗msg, uint8_t msgLength)

  Application provided function that NcApi calls whenever any valid NeocCortec messages has been received.
- typedef void(∗ NeoMeshHostAckCallback) (tNcApiHostAckNack ∗m)

  Application provided functions that NcApi calls when a message type ¨0x50: Acknowledge for previously sent packet¨ is received, or a message type ¨0x51: Non-Acknowledge for previously sent packet¨ is received.
- typedef void(∗ NeoMeshHostUappStatusCallback) (tNcApiHostUappStatus ∗m)

  *Application provided functions that NcApi calls when a message type ¨0x56: Uapp packet send.
 message type ¨0x57: Uapp packet was droped.*
- typedef void(∗ NeoMeshHostDataCallback) (tNcApiHostData ∗m)

  Application provided function that NcApi calls when a message type ¨0x52: Host Data¨ is received.
- typedef void(∗ NeoMeshHostDataHapaCallback) (tNcApiHostDataHapa ∗m)

  Application provided function that NcApi calls when a message type ¨0x53: Host Data HAPA¨ is received.
- typedef void(∗ NeoMeshHostUappDataCallback) (tNcApiHostUappData ∗m)

  Application provided function that NcApi calls when a message type ¨0x54: Host Data Unacknowledged¨ is received.
- typedef void(∗ NeoMeshHostUappDataHapaCallback) (tNcApiHostUappDataHapa ∗m)

  Application provided function that NcApi calls when a message type ¨0x55: Host Data HAPA Unacknowledged¨ is received.
- typedef void(∗ NeoMeshNodeInfoReplyCallback) (tNcApiNodeInfoReply ∗m)

  Application provided function that NcApi calls when a message type ¨0x58: Node Info Reply¨ is received.
- typedef void(∗ NeoMeshNeighborListReplyCallback) (tNcApiNeighborListReply ∗m)

  Application provided function that NcApi calls when a message type ¨0x59: Neighbor List Reply¨ is received.
- typedef void(∗ NeoMeshRouteInfoRequestReplyCallback) (tNcApiRouteInfoRequestReply ∗m)

  Application provided function that NcApi calls when a message type ¨0x5c: Route Info Request Reply¨ is received.
- typedef void(∗ NeoMeshNetCmdResponseCallback) (tNcApiNetCmdReply ∗m)

  Application provided function that NcApi calls when a message type ¨0x5a: Network Command Reply¨ is received.
- typedef void(∗ NeoMeshWesStatusCallback) (tNcApiWesStatus ∗m)

  Application provided function that NcApi calls when a message type ¨0x60: WES Status¨ is received.
- typedef void(∗ NeoMeshWesSetupRequestCallback) (tNcApiWesSetupRequest ∗m)

  Application provided function that NcApi calls when a message type ¨0x61: WES Setup Request¨ is received.

Enumerations

- enum tNcModuleMode { SAPI_LOGGED_OUT , SAPI , AAPI }

  Enum to keep track of module modes.
- enum **tNcApiSapiMessageType** {
  LoginOK = 0x80 , **LoginError** = 0x81 , **BootloaderStarted** = 0x82 , **ProtocolStarted** = 0x83 ,
  ProtocolError = 0x84 , **ProtocolListOutput** = 0x85 , **SettingValue** = 0x86 }

Functions

- NcApiErrorCodes NcApiSupportTxData (uint8_t n, uint8_t *finalMsg, uint8_t finalMsgLength)

  Application provied function that NcApi calls if there is any pending data to be written to the UART.
- void NcApiSupportMessageReceived (uint8_t n, void *callbackToken, uint8_t *msg, uint8_t msgLength)

  Application provided function that NcApi calls after it has succesfully received a full message.
- void NcApiSupportMessageWritten (uint8_t n, void *callbackToken, uint8_t *finalMsg, uint8_t finalMsg↩ Length)

  Application provided function that NcApi calls after it has succesfully written the message.
- NeoMesh::NeoMesh (Stream *serial, uint8_t cts_pin)

  Construct new NeoMesh object.
- void **NeoMesh::start** ()

  Starts the NeoMesh API.
- void **NeoMesh::update** ()

  Handles all housekeeping. Should be called from main loop.
- void NeoMesh::set_password (uint8_t new_password[5])

  Change the password the API should use to log into the NC module.
- void **NeoMesh::write** (uint8_t *finalMsg, uint8_t finalMsgLength)
- void NeoMesh::change_node_id (uint16_t node_id)

  Change the id of the node in the NeoMesh network When the ID of a node is changed, it will not revert on reboot. The ID is saved safely within the NeoCortec module. This function reboots the NeoCortec module, so it will not be possible to send data from this node for a period of time after calling this function.
- void NeoMesh::change_network_id (uint8_t network_id[16])

  Change the network id Change the network ID setting within the NeoCortec module. As goes for the node id, the network id is not reverted on reboot. Alle nodes in a network must have the same network id in order to communicate.
- void NeoMesh::change_trace_output_setting (bool mode)

  Change trace output setting Tracing output enables the user to see neighbors connected to the node in realtime. Of course, this comes with the cost of higher power consumption, which is why you usually only want this setting to be turned on, on gateway nodes with main power. Batterypowered sensor nodes should have this setting turned off.
- void **NeoMesh::set_baudrate** (uint32_t baudrate)

  Change baudrate (Must be called before start) If the module is configured to use a dfferent baudrate than 115200, this function must be called with the custom baudrate before the start function is called.
- void NeoMesh::send_unacknowledged (uint16_t destNodeId, uint8_t port, uint16_t appSeqNo, uint8_↩ t *payload, uint8_t payloadLen)

  send an unacknowledged message to a node in the network
- void NeoMesh::send_acknowledged (uint16_t destNodeId, uint8_t port, uint8_t *payload, uint8_t payload↩ Len)

  send an acknowledged message to a node in the network If the host_ack_callback is set it will be called when the message recepient has acknowledged
- void NeoMesh::send_wes_command (NcApiWesCmdValues cmd)

  Send a WES command to the node.
- void NeoMesh::send_wes_respond (uint64_t uid, uint16_t nodeId)

  Send a wes response.
- bool NeoMesh::change_setting (uint8_t setting, uint8_t *value, uint8_t length)

  Change a setting in the NC module.
- bool NeoMesh::switch_sapi_aapi ()

  Switch NeoCortec module to SAPI mode.
- bool NeoMesh::login_sapi ()

  Log in.
- void NeoMesh::start_bootloader ()

  Send command to start bootloader.
- void NeoMesh::start_protocol_stack ()

  Starts the protocol.

- bool NeoMesh::get_setting (uint8_t setting, NcSetting *setting_ret)

  Gets a setting from the NC modules flash.
- void NeoMesh::set_setting (uint8_t setting, uint8_t *setting_value, uint8_t setting_value_length)

  Changes a setting in the NC modules RAM.
- void **NeoMesh::commit_settings** ()

  Move all settings in RAM to FLASH.
- void NeoMesh::write_sapi_command (uint8_t cmd1, uint8_t cmd2, uint8_t *data, uint8_t data_length)

  Write a system interface command.
- void NeoMesh::write_raw (uint8_t *data, uint8_t length)

  Writes raw bytes to protocol uart.
- bool NeoMesh::wait_for_sapi_response (tNcSapiMessage *message, uint32_t timeout_ms)

  Wait for system interface to send response.
- tNcModuleMode NeoMesh::get_module_mode ()

  get module mode
- static void **NeoMesh::pass_through_cts** ()
- void SAPIParser::push_char (uint8_t c)

  Push new character to buffer.
- bool SAPIParser::message_available ()

  See if a message is received but not yet read.
- tNcSapiMessage **SAPIParser::get_pending_message** ()

  Get latest message message_available() should be called before this If no new message is received, the same message will be returned as last time this function was called.

Variables

- NeoMesh * **instances** [1]
- tNcApi **g_ncApi** [1]

  Application defined array of NcApi instances in use.
- uint8_t **g_numberOfNcApis** = 1

  Application defined number of elements in the g_ncApi array.
- tNcApiRxHandlers **ncRx**

4.1.1 Detailed Description

4.1.2 Typedef Documentation

4.1.2.1 NeoMeshHostAckCallback

```
typedef void(* NeoMeshHostAckCallback) (tNcApiHostAckNack *m)
```

Application provided functions that NcApi calls when a
message type ¨0x50: Acknowledge for previously sent packet¨ is received, or a
message type ¨0x51: Non-Acknowledge for previously sent packet¨ is received.

The appropriate function is called when a HostAck or HostNAck message has been received for a previously sent payload package. The callback function delivers a pointer to a struct containing the relevant information.

Parameters

m	Strongly typed message

4.1.2.2 NeoMeshHostDataCallback

```
typedef void(* NeoMeshHostDataCallback) (tNcApiHostData *m)
```

Application provided function that NcApi calls when a
message type ¨0x52: Host Data¨ is received.

The callback is issued when the modules receive payload data, that requires acknowledge, from another module in the NEOCORTEC mesh network. The callback function delivers a pointer to a struct containing the relevant information.

Parameters

m	Strongly typed message

4.1.2.3 NeoMeshHostDataHapaCallback

```
typedef void(* NeoMeshHostDataHapaCallback) (tNcApiHostDataHapa *m)
```

Application provided function that NcApi calls when a
message type ¨0x53: Host Data HAPA¨ is received.

The callback is issued when the modules receive payload data, that requires acknowledge, from another module in the NEOCORTEC mesh network which has been configured to use the High Precision Packet Age feature (HAPA). The callback function delivers a pointer to a struct containing the relevant information.

Parameters

m	Strongly typed message

4.1.2.4 NeoMeshHostUappDataCallback

```
typedef void(* NeoMeshHostUappDataCallback) (tNcApiHostUappData *m)
```

Application provided function that NcApi calls when a
message type ¨0x54: Host Data Unacknowledged¨ is received.

The callback is issued when the modules receive payload data, that NOT requires acknowledge, from another module in the NEOCORTEC mesh network. The callback function delivers a pointer to a struct containing the relevant information.

Parameters

m	Strongly typed message

4.1.2.5 NeoMeshHostUappDataHapaCallback

```
typedef void(* NeoMeshHostUappDataHapaCallback) (tNcApiHostUappDataHapa *m)
```

Application provided function that NcApi calls when a
message type ¨0x55: Host Data HAPA Unacknowledged¨ is received.

The callback is issued when the modules receive payload data, that requires acknowledge, from another module in the NEOCORTEC mesh network which has been configured to use the High Precision Packet Age feature (HAPA). The callback function delivers a pointer to a struct containing the relevant information.

Parameters

m	Strongly typed message

4.1.2.6 NeoMeshHostUappStatusCallback

```
typedef void(* NeoMeshHostUappStatusCallback) (tNcApiHostUappStatus *m)
```

Application provided functions that NcApi calls when a
message type ¨0x56: Uapp packet send.
 message type ¨0x57: Uapp packet was droped.

The appropriate function is called when a Uapp send or dropped message has been received for a previously sent payload package. The callback function delivers a pointer to a struct containing the relevant information.

Parameters

m	Strongly typed message

4.1.2.7 NeoMeshNeighborListReplyCallback

```
typedef void(* NeoMeshNeighborListReplyCallback) (tNcApiNeighborListReply *m)
```

Application provided function that NcApi calls when a
message type ¨0x59: Neighbor List Reply¨ is received.

Parameters

m	Strongly typed message

4.1.2.8 NeoMeshNetCmdResponseCallback

```
typedef void(* NeoMeshNetCmdResponseCallback) (tNcApiNetCmdReply *m)
```

Application provided function that NcApi calls when a
message type ¨0x5a: Network Command Reply¨ is received.

Parameters

m	Strongly typed message

4.1.2.9 NeoMeshNodeInfoReplyCallback

```
typedef void(* NeoMeshNodeInfoReplyCallback) (tNcApiNodeInfoReply *m)
```

Application provided function that NcApi calls when a
message type ¨0x58: Node Info Reply¨ is received.

Parameters

m	Strongly typed message

4.1.2.10 NeoMeshReadCallback

```
typedef void(* NeoMeshReadCallback) (uint8_t *msg, uint8_t msgLength)
```

Application provided function that NcApi calls whenever any valid NeocCortec messages has been received.

This function will deliver a byte array containing the received raw UART frame.
It is normally not necessary to register for this callback, as there are other callbacks which are specific to the various types of application data.

Parameters

msg	Pointer to the message
msgLength	Message length in bytes

4.1.2.11 NeoMeshRouteInfoRequestReplyCallback

```
typedef void(* NeoMeshRouteInfoRequestReplyCallback) (tNcApiRouteInfoRequestReply *m)
```

Application provided function that NcApi calls when a
message type ¨0x5c: Route Info Request Reply¨ is received.

Parameters

m	Strongly typed message

4.1.2.12 NeoMeshWesSetupRequestCallback

```
typedef void(* NeoMeshWesSetupRequestCallback) (tNcApiWesSetupRequest *m)
```

Application provided function that NcApi calls when a
message type ¨0x61: WES Setup Request¨ is received.

Parameters

m	Strongly typed message

4.1.2.13 NeoMeshWesStatusCallback

```
typedef void(* NeoMeshWesStatusCallback) (tNcApiWesStatus *m)
```

Application provided function that NcApi calls when a message type ¨0x60: WES Status¨ is received.

Parameters

m	Strongly typed message

4.1.3 Enumeration Type Documentation

4.1.3.1 tNcModuleMode

```
enum tNcModuleMode
```

Enum to keep track of module modes.

Enumerator

SAPI_LOGGED_OUT	Module is in system interface mode, but not yet logged in.
SAPI	Module is in system interface mode and ready to receive system commands.
AAPI	Module is in application mode and ready to send and receive data.

4.1.4 Function Documentation

4.1.4.1 change_network_id()

```
void NeoMesh::change_network_id (
            uint8_t network_id[16] )
```

Change the network id Change the network ID setting within the NeoCortec module. As goes for the node id, the network id is not reverted on reboot. Alle nodes in a network must have the same network id in order to communicate.

Parameters

network↩ _id	The new network id as 16 bytes

4.1.4.2 change_node_id()

```
void NeoMesh::change_node_id (
            uint16_t node_id )
```

Change the id of the node in the NeoMesh network When the ID of a node is changed, it will not revert on reboot. The ID is saved safely within the NeoCortec module. This function reboots the NeoCortec module, so it will not be possible to send data from this node for a period of time after calling this function.

Parameters

node_id	The new nodeid. NOTE: Can not be 0

4.1.4.3 change_setting()

```
bool NeoMesh::change_setting (
            uint8_t setting,
            uint8_t * value,
            uint8_t length )
```

Change a setting in the NC module.

This is safe to call at all times. Bootloader mode will be entered and exited

4.1.4.4 change_trace_output_setting()

```
void NeoMesh::change_trace_output_setting (
            bool mode )
```

Change trace output setting Tracing output enables the user to see neighbors connected to the node in realtime. Of course, this comes with the cost of higher power consumption, which is why you usually only want this setting to be turned on, on gateway nodes with main power. Batterypowered sensor nodes should have this setting turned off.

Parameters

mode	True if trace output should be turned on

4.1.4.5 get_module_mode()

```
tNcModuleMode NeoMesh::get_module_mode ( )
```

get module mode

Returns

tNcModuleMode choices: AAPI, SAPI, SAPI_LOGGED_OUT

4.1.4.6 get_setting()

```
bool NeoMesh::get_setting (
            uint8_t setting,
            NcSetting * setting_ret )
```

Gets a setting from the NC modules flash.

This setting is safe to call at all times. It will automatically enter bootloader mode if it is not done before the function is called, in which case it will also start the protocol stack when the settings is retrieved

Parameters

setting	The id of the setting to be retrieved
setting_ret	A NcSetting object in which to put the info of the retrieved setting

Returns

true if setting is rerieved. False otherwise

4.1.4.7 login_sapi()

```
bool NeoMesh::login_sapi ( )
```

Log in.

Log in to bootloader. This must be done before changing any settings

Returns

true if logged in. False otherwise. If module is not in SAPI mode or already logged in, this function will return false;

4.1.4.8 message_available()

```
bool SAPIParser::message_available ( )
```

See if a message is received but not yet read.

Returns

True if a message is pending. False otherwise

4.1.4.9 NcApiSupportMessageReceived()

```
void NcApiSupportMessageReceived (
            uint8_t n,
            void * callbackToken,
            uint8_t * msg,
            uint8_t msgLength )
```

Application provided function that NcApi calls after it has succesfully received a full message.

The API will call this function once a message has successfully been received in full from the module. This can be used by the application layer to initiate invocation of the relevant callback function via the function NcApiExecute↩ Callbacks().

Parameters

n	Index of tNcApi instance that message was written to
callbackToken	Application provided context / token / tag
msg	Pointer to the message
msgLength	Message length in bytes

4.1.4.10 NcApiSupportMessageWritten()

```
void NcApiSupportMessageWritten (
            uint8_t n,
            void * callbackToken,
            uint8_t * finalMsg,
            uint8_t finalMsgLength )
```

Application provided function that NcApi calls after it has succesfully written the message.

The API will call this function once a message has successfully been written in full to the module. This can be used by the application layer to check that a previous request to send a message was completed successfully.

Parameters

n	Index of tNcApi instance that message was written to
callbackToken	Application provided context / token / tag
finalMsg	Pointer to the message
finalMsgLength	Message length in bytes

4.1.4.11 NcApiSupportTxData()

```
NcApiErrorCodes NcApiSupportTxData (
            uint8_t n,
            uint8_t * finalMsg,
            uint8_t finalMsgLength )
```

Application provied function that NcApi calls if there is any pending data to be written to the UART.

The API will call this function to send data to the UART. The function is called with a pointer to the actual data to be written. The function shall implement the necessary code required to output the data to the UART. If not all data is send whem returning from the function, NCAPI_DATA_PENDING must be returned, and NcApiTxDataDone must be called when last data is send.

Parameters

n	Index of tNcApi instance that the data should be written to, ie. which UART
finalMsg	Pointer to the buffer
finalMsgLength	Number of bytes to be written

Returns

NCAPI_OK if all data is send, if any pending data NCAPI_DATA_PENDING is returned.

4.1.4.12 NeoMesh()

```
NeoMesh::NeoMesh (
            Stream * serial,
            uint8_t cts_pin )
```

Construct new NeoMesh object.

Parameters

uart_num	Which UART is connected to the AAPI UART of the NeoCortec module
serial	Pointer to the Stream object attached to UART

4.1.4.13 push_char()

```
void SAPIParser::push_char (
            uint8_t c )
```

Push new character to buffer.

Parameters

c	New character

4.1.4.14 send_acknowledged()

```
void NeoMesh::send_acknowledged (
            uint16_t destNodeId,
            uint8_t port,
            uint8_t * payload,
            uint8_t payloadLen )
```

send an acknowledged message to a node in the network If the host_ack_callback is set it will be called when the message recepient has acknowledged

Parameters

destNodeId	The node id of the recepient
port	Which port to send to. Allows recepient to filter messages. If not used, write 0
payload	The payload data to send
payloadLen	The length of the payload array

4.1.4.15 send_unacknowledged()

```
void NeoMesh::send_unacknowledged (
            uint16_t destNodeId,
            uint8_t port,
            uint16_t appSeqNo,
            uint8_t * payload,
            uint8_t payloadLen )
```

send an unacknowledged message to a node in the network

Parameters

destNodeId	The node id of the recepient
port	Which port to send to. Allows recepient to filter messages. If not used, write 0
appSeqNo	message sequence number. If more messages are sent after each other, the sequence number must be different each time
payload	The payload data to send
payloadLen	The length of the payload array

4.1.4.16 send_wes_command()

```
void NeoMesh::send_wes_command (
            NcApiWesCmdValues cmd )
```

Send a WES command to the node.

Parameters

cmd	The command

4.1.4.17 send_wes_respond()

```
void NeoMesh::send_wes_respond (
            uint64_t uid,
            uint16_t nodeId )
```

Send a wes response.

Parameters

uid	
nodeId	

4.1.4.18 set_password()

```
void NeoMesh::set_password (
```

```
            uint8_t new_password[5] )
```

Change the password the API should use to log into the NC module.

In order to change settings on the module, it needs to be in bootloader mode and logged in. The standard login password is ¨Lvl10¨. This function only needs to be called if the password on the NC module is different from ¨Lvl10¨

Parameters

new_password	An array of 5 bytes containing the password

4.1.4.19 set_setting()

```
void NeoMesh::set_setting (
            uint8_t setting,
            uint8_t * setting_value,
            uint8_t setting_value_length )
```

Changes a setting in the NC modules RAM.

Before calling this function the NC module must be in SAPI mode and logged in. The setting being set is only saved to ram, so when finishd commit_settings¨()¨ must be called. If you wish to change a setting and for the API to automatically enter and exit bootloader mode, call the function change_setting¨()¨ instead

Parameters

setting	The id of the setting to change
setting_value	Pointer to the setting value
setting_value_length	Length of setting value

4.1.4.20 start_bootloader()

```
void NeoMesh::start_bootloader ( )
```

Send command to start bootloader.

This function is useless. When entering system uart mode on AAPI uart the bootloader will automatically be started.

4.1.4.21 start_protocol_stack()

```
void NeoMesh::start_protocol_stack ( )
```

Starts the protocol.

If bootloader mode is entered and settings are changed, this function should be called when done, so that the NC module can once again join a mesh network and send and receive messages

4.1.4.22 switch_sapi_aapi()

```
bool NeoMesh::switch_sapi_aapi ( )
```

Switch NeoCortec module to SAPI mode.

The NeoCortec module can be in two different modes: AAPI and SAAPI. When in AAPI mode it is able to send messages to and receive messages from a NeoMesh network. When in SAPI mode it is able to change module settings. Next step after switching to SAPI mode will often be to log in with password

Returns

True if the module successfully switched to SAPI mode. False otherwise

4.1.4.23 wait_for_sapi_response()

```
bool NeoMesh::wait_for_sapi_response (
            tNcSapiMessage * message,
            uint32_t timeout_ms )
```

Wait for system interface to send response.

Waits for system interface to return a response. If it takes longer than the given timeout the function will return before a response is received

Parameters

message	Pointer to a message object in which to put the response data
timeout_ms	Maximum amount of time this should wait. In milliseconds

Returns

True if message was received. False if timed out

4.1.4.24 write_raw()

```
void NeoMesh::write_raw (
            uint8_t * data,
            uint8_t length )
```

Writes raw bytes to protocol uart.

Parameters

data	Data to write
length	Length of data array

4.1.4.25 write_sapi_command()

```
void NeoMesh::write_sapi_command (
            uint8_t cmd1,
            uint8_t cmd2,
            uint8_t * data,
            uint8_t data_length )
```

Write a system interface command.

Before calling this function the NC module must be in bootloader mode and logged in

Parameters

cmd1	Command one
cmd2	Command two
data	Data to pass if command requires
data_length	Length of passed data

Chapter 5

Class Documentation

5.1 NcApi Struct Reference

This is the definition of a global structure holding various information, in particular the RX and TX buffers for a specific UART, and tha set of application callbacks to handle any received messages. These data are managed by the NcApi module, and as such the fields are considered internal to NcApi.

`#include <NcApi.h>`

Public Attributes

- uint8_t **rxBuffer** [NCAPI_RXBUFFER_SIZE]

  Internal UART receive buffer.
- uint16_t **rxPosition**

  Internal position in UART receive buffer.
- volatile uint8_t **txMsgLen**

  Internal UART transmit buffer length.
- uint8_t **txBuffer** [NCAPI_TXBUFFER_SIZE]

  Internal UART transmit buffer.
- void * **writeCallbackToken**

  Internal UART transmit callback token.
- volatile uint8_t **recvBufIsSynced**

  Internal UART receive buffer in sync.
- tNcApiRxHandlers * **NcApiRxHandlers**

  Set of application callbacks to handle any received messages.

5.1.1 Detailed Description

This is the definition of a global structure holding various information, in particular the RX and TX buffers for a specific UART, and tha set of application callbacks to handle any received messages. These data are managed by the NcApi module, and as such the fields are considered internal to NcApi.

The documentation for this struct was generated from the following file:

- NcApi.h

5.2 NcApiAltCmdMessage Struct Reference

Definition of message type ¨0x20: ALT command¨.

`#include <NcApi.h>`

Public Attributes

- NcApiAltCmdValues **cmd**
  ALT Command.

5.2.1 Detailed Description

Definition of message type ¨0x20: ALT command¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.3 NcApiAltCmdParams Struct Reference

Parameters for the function handling message type ¨0x20: ALT command¨.

`#include <NcApi.h>`

Public Attributes

- tNcApiAltCmdMessage **msg**
  The actual corresponding message.
- void * **callbackToken**
  Application provided token / context / tag that it wants to called back with. NcApi does not inspect this parameter, it merely passes it along.

5.3.1 Detailed Description

Parameters for the function handling message type ¨0x20: ALT command¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.4 NcApiHostAckNack Struct Reference

Parameters for the function handling message type ¨0x50: Acknowledge for previously sent packet¨
Parameters for the function handling message type ¨0x51: Non-Acknowledge for previously sent packet¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **originId**

5.4.1 Detailed Description

Parameters for the function handling message type ¨0x50: Acknowledge for previously sent packet¨
Parameters for the function handling message type ¨0x51: Non-Acknowledge for previously sent packet¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.5 NcApiHostData Struct Reference

Parameters for the function handling message type ¨0x52: Host Data¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **originId**
- uint16_t **packageAge**
- uint8_t **port**
- uint8_t **payloadLength**
- uint8_t * **payload**

5.5.1 Detailed Description

Parameters for the function handling message type ¨0x52: Host Data¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.6 NcApiHostDataHapa Struct Reference

Parameters for the function handling message type ¨0x53: Host Data HAPA¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **originId**
- uint32_t **packageAge**
- uint8_t **port**
- uint8_t **payloadLength**
- uint8_t * **payload**

5.6.1 Detailed Description

Parameters for the function handling message type ¨0x53: Host Data HAPA¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.7 NcApiHostUappData Struct Reference

Parameters for the function handling message type ¨0x54: Host Data Unacknowledged¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **originId**
- uint16_t **packageAge**
- uint8_t **port**
- uint16_t **appSeqNo**
- uint8_t **payloadLength**
- uint8_t ∗ **payload**

5.7.1 Detailed Description

Parameters for the function handling message type ¨0x54: Host Data Unacknowledged¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.8 NcApiHostUappDataHapa Struct Reference

Parameters for the function handling message type ¨0x55: Host Data HAPA Unacknowledged¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **originId**
- uint32_t **packageAge**
- uint8_t **port**
- uint16_t **appSeqNo**
- uint8_t **payloadLength**
- uint8_t ∗ **payload**

5.8.1 Detailed Description

Parameters for the function handling message type ¨0x55: Host Data HAPA Unacknowledged¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.9 NcApiHostUappStatus Struct Reference

Parameters for the function handling message type ¨0x56: Uapp packet send
 Parameters for the function handling message type ¨0x57: Uapp packet droped.

```
#include <NcApi.h>
```

Public Attributes

- uint16_t **originId**
- uint16_t **appSeqNo**

5.9.1 Detailed Description

Parameters for the function handling message type ¨0x56: Uapp packet send
 Parameters for the function handling message type ¨0x57: Uapp packet droped.

The documentation for this struct was generated from the following file:

- NcApi.h

5.10 NcApiNeighbor Struct Reference

Public Attributes

- uint16_t **nodeId**
- uint8_t **RSSI**

The documentation for this struct was generated from the following file:

- NcApi.h

5.11 NcApiNeighborListReply Struct Reference

Parameters for the function handling message type ¨0x59: Neighbor List Reply¨.

```
#include <NcApi.h>
```

Public Attributes

- uint8_t **NeighborsCount**

  Numbers of neighbors.
- tNcApiNeighbor **Neighbor** [12]

  Array of neighbors.

5.11.1 Detailed Description

Parameters for the function handling message type ¨0x59: Neighbor List Reply¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.12 NcApiNeighborListRequestMessage Struct Reference

Definition of message type ¨0x09: Neighbor List Request¨.

`#include <NcApi.h>`

Public Attributes

- void * **dummy**

  (No parameters)

5.12.1 Detailed Description

Definition of message type ¨0x09: Neighbor List Request¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.13 NcApiNeighborListRequestParams Struct Reference

Parameters for the function handling message type ¨0x09: Neighbor List Request¨.

`#include <NcApi.h>`

Public Attributes

- tNcApiNeighborListRequestMessage **msg**

  The actual corresponding message.
- void * **callbackToken**

  Application provided token / context / tag that it wants to called back with. NcApi does not inspect this parameter, it merely passes it along.

5.13.1 Detailed Description

Parameters for the function handling message type ¨0x09: Neighbor List Request¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.14 NcApiNetCmdMessage Struct Reference

Definition of message type ¨0x0a: Network Command¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **destNodeId**

  Destination node ID.
- uint8_t **cmd**

  Network Command.
- uint8_t * **payload**

  Pointer to payload, if any.
- uint8_t **payloadLength**

  PayloadLength Length of payload.

5.14.1 Detailed Description

Definition of message type ¨0x0a: Network Command¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.15 NcApiNetCmdParams Struct Reference

Parameters for the function handling message type ¨0x0a: Network Command¨.

`#include <NcApi.h>`

Public Attributes

- tNcApiNetCmdMessage **msg**

  The actual corresponding message.
- void * **callbackToken**

  Application provided token / context / tag that it wants to called back with. NcApi does not inspect this parameter, it merely passes it along.

5.15.1 Detailed Description

Parameters for the function handling message type ¨0x0a: Network Command¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.16 NcApiNetCmdReply Struct Reference

Parameters for the function handling message type ¨0x5a: Network command response¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **originId**
- NcApiNetCmdValues **cmd**
- uint8_t **payloadLength**

  PayloadLength Length of payload.
- uint8_t * **payload**

  Pointer to payload, if any.

5.16.1 Detailed Description

Parameters for the function handling message type ¨0x5a: Network command response¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.17 NcApiNodeInfoParams Struct Reference

Parameters for the function handling message type ¨0x08: Node Info Request¨.

`#include <NcApi.h>`

Public Attributes

- tNcApiNodeInfoRequestMessage **msg**

  The actual corresponding message.
- void * **callbackToken**

  Application provided token / context / tag that it wants to called back with. NcApi does not inspect this parameter, it merely passes it along.

5.17.1 Detailed Description

Parameters for the function handling message type ¨0x08: Node Info Request¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.18 NcApiNodeInfoReply Struct Reference

Parameters for the function handling message type ¨0x58: Node Info Reply¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **nodeId**
  Node ID.
- uint8_t **uid** [5]
  Node uid.
- NcApiNodeType **Type**
  Node Hardware Type.

5.18.1 Detailed Description

Parameters for the function handling message type ¨0x58: Node Info Reply¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.19 NcApiNodeInfoRequestMessage Struct Reference

Definition of message type ¨0x08: Node Info Request¨.

`#include <NcApi.h>`

Public Attributes

- void * **dummy**
  (No parameters)

5.19.1 Detailed Description

Definition of message type ¨0x08: Node Info Request¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.20 NcApiRouteInfoRequestReply Struct Reference

Parameters for the function handling message type ¨0x5c: Route Info Request Reply¨.

`#include <NcApi.h>`

Public Attributes

- uint8_t **Bitmap** [16]

5.20.1 Detailed Description

Parameters for the function handling message type ¨0x5c: Route Info Request Reply¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.21 NcApiRxHandlers Struct Reference

Set of application callbacks to handle any received messages. Each callback is optional allowing the application to register specific callbacks only for the message types of particular interest.

`#include <NcApi.h>`

Public Attributes

- pfnNcApiReadCallback **pfnReadCallback**

  Optional callback for all received messages as a byte array.
- pfnNcApiHostAckCallback **pfnHostAckCallback**

  Optional callback for all received HostAck messages.
- pfnNcApiHostAckCallback **pfnHostNAckCallback**

  Optional callback for all received HostNAck messages.
- pfnNcApiHostUappStatusCallback **pfnHostUappSendCallback**

  Optional callback for all received UappSend messages.
- pfnNcApiHostUappStatusCallback **pfnHostUappDropedCallback**

  Optional callback for all received UappDropped messages.
- pfnNcApiHostDataCallback **pfnHostDataCallback**

  Optional callback for all received HostData messages.
- pfnNcApiHostDataHapaCallback **pfnHostDataHapaCallback**

  Optional callback for all received HostDataHapa messages.
- pfnNcApiHostUappDataCallback **pfnHostUappDataCallback**

  Optional callback for all received HostUappData messages.
- pfnNcApiHostUappDataHapaCallback **pfnHostUappDataHapaCallback**

  Optional callback for all received HostUappDataHapa messages.
- pfnNcApiNodeInfoReplyCallback **pfnNodeInfoReplyCallback**

  Optional callback for all received NodeInfoReply messages.
- pfnNcApiNeighborListReplyCallback **pfnNeighborListReplyCallback**

  Optional callback for all received NeighborListReply messages.
- pfnNcApiRouteInfoRequestReplyCallback **pfnRouteInfoRequestReplyCallback**

  Optional callback for all received RouteInfoRequestReply messages.
- pfnNcApiNetCmdResponseCallback **pfnNetCmdResponseCallback**

  Optional callback for all received NetCmdResponse messages.
- pfnNcApiWesSetupRequestCallback **pfnWesSetupRequestCallback**

  Optional callback for all received WesSetupRequest messages.
- pfnNcApiWesStatusCallback **pfnWesStatusCallback**

  Optional callback for all received WesStatus messages.

5.21.1 Detailed Description

Set of application callbacks to handle any received messages. Each callback is optional allowing the application to register specific callbacks only for the message types of particular interest.

The documentation for this struct was generated from the following file:

- NcApi.h

5.22 NcApiSendAckMessage Struct Reference

Definition of message type ¨0x03: Acknowledged Packet¨.

```
#include <NcApi.h>
```

Public Attributes

- uint16_t **destNodeId**

  Destination node ID.
- uint8_t **destPort**

  Destination port.
- uint8_t **payloadLength**

  PayloadLength Length of payload.
- uint8_t * **payload**

  Pointer to payload, if any.

5.22.1 Detailed Description

Definition of message type ¨0x03: Acknowledged Packet¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.23 NcApiSendAckParams Struct Reference

Parameters for the function handling message type ¨0x03: Acknowledged Packet¨.

`#include <NcApi.h>`

Public Attributes

- tNcApiSendAckMessage **msg**

  The actual corresponding message.
- void * **callbackToken**

  Application provided token / context / tag that it wants to called back with. NcApi does not inspect this parameter, it merely passes it along.

5.23.1 Detailed Description

Parameters for the function handling message type ¨0x03: Acknowledged Packet¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.24 NcApiSendUnackMessage Struct Reference

Definition of message type ¨0x02: Unacknowledged Packet¨.

`#include <NcApi.h>`

Public Attributes

- uint16_t **destNodeId**

  Destination node ID.
- uint8_t **destPort**

  Destination port.
- uint16_t **appSeqNo**

  Application sequence number.
- uint8_t * **payload**

  Pointer to payload, if any.
- uint8_t **payloadLength**

  PayloadLength Length of payload.

5.24.1 Detailed Description

Definition of message type ¨0x02: Unacknowledged Packet¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.25 NcApiSendUnackParams Struct Reference

Parameters for the function handling message type ¨0x02: Unacknowledged Packet¨.

`#include <NcApi.h>`

Public Attributes

- tNcApiSendUnackMessage **msg**

  The actual corresponding message.
- void * **callbackToken**

  Application provided token / context / tag that it wants to called back with. NcApi does not inspect this parameter, it merely passes it along.

5.25.1 Detailed Description

Parameters for the function handling message type ¨0x02: Unacknowledged Packet¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.26 NcApiWesCmdMessage Struct Reference

Definition of message type ¨0x10: WES Command¨.

`#include <NcApi.h>`

Public Attributes

- NcApiWesCmdValues **cmd**
  WES Command.

5.26.1 Detailed Description

Definition of message type ¨0x10: WES Command¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.27 NcApiWesCmdParams Struct Reference

Parameters for the function handling message type ¨0x10: WES Command¨.

`#include <NcApi.h>`

Public Attributes

- tNcApiWesCmdMessage **msg**
  The actual corresponding message.
- void * **callbackToken**
  Application provided token / context / tag that it wants to called back with. NcApi does not inspect this parameter, it merely passes it along.

5.27.1 Detailed Description

Parameters for the function handling message type ¨0x10: WES Command¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.28 NcApiWesResponseMessage Struct Reference

Public Attributes

- uint8_t **uid** [5]
  UID.
- uint16_t **nodeId**
  NodeId.
- uint8_t **appSettings** [WES_APPSETTINGS_LENGTH]
  appSettings

The documentation for this struct was generated from the following file:

- NcApi.h

5.29 NcApiWesResponseParams Struct Reference

Parameters for the function handling message type ¨0x11: WES Setup Response¨.

```
#include <NcApi.h>
```

Public Attributes

- tNcApiWesResponseMessage **msg**
  The actual corresponding message.
- void * **callbackToken**
  Application provided token / context / tag that it wants to called back with. NcApi does not inspect this parameter, it merely passes it along.

5.29.1 Detailed Description

Parameters for the function handling message type ¨0x11: WES Setup Response¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.30 NcApiWesSetupRequest Struct Reference

Parameters for the function handling message type ¨0x61: WES Setup Request¨.

```
#include <NcApi.h>
```

Public Attributes

- uint8_t **uid** [5]
- uint8_t **appFuncType**

5.30.1 Detailed Description

Parameters for the function handling message type ¨0x61: WES Setup Request¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.31 NcApiWesStatus Struct Reference

Parameters for the function handling message type ¨0x60: WES Status¨.

`#include <NcApi.h>`

Public Attributes

- uint8_t **Status**
  See NcApiWesStatusValues.

5.31.1 Detailed Description

Parameters for the function handling message type ¨0x60: WES Status¨.

The documentation for this struct was generated from the following file:

- NcApi.h

5.32 NcSetting Struct Reference

Public Attributes

- uint8_t **value** [32]
- uint8_t **length**

The documentation for this struct was generated from the following file:

- NeoMesh.h

5.33 NeoMesh Class Reference

Object that handles connection to NeoCortec module.

`#include <NeoMesh.h>`

Public Member Functions

- NeoMesh (Stream *serial, uint8_t cts_pin)

  Construct new NeoMesh object.
- void **start** ()

  Starts the NeoMesh API.
- void **write** (uint8_t *finalMsg, uint8_t finalMsgLength)
- void **update** ()

  Handles all housekeeping. Should be called from main loop.
- void change_node_id (uint16_t node_id)

  Change the id of the node in the NeoMesh network When the ID of a node is changed, it will not revert on reboot. The ID is saved safely within the NeoCortec module. This function reboots the NeoCortec module, so it will not be possible to send data from this node for a period of time after calling this function.
- void change_network_id (uint8_t network_id[16])

  Change the network id Change the network ID setting within the NeoCortec module. As goes for the node id, the network id is not reverted on reboot. Alle nodes in a network must have the same network id in order to communicate.
- void change_trace_output_setting (bool mode)

  Change trace output setting Tracing output enables the user to see neighbors connected to the node in realtime. Of course, this comes with the cost of higher power consumption, which is why you usually only want this setting to be turned on, on gateway nodes with main power. Batterypowered sensor nodes should have this setting turned off.
- void **set_baudrate** (uint32_t baudrate)

  Change baudrate (Must be called before start) If the module is configured to use a dfferent baudrate than 115200, this function must be called with the custom baudrate before the start function is called.
- void send_unacknowledged (uint16_t destNodeId, uint8_t port, uint16_t appSeqNo, uint8_t *payload, uint8_t payloadLen)

  send an unacknowledged message to a node in the network
- void send_acknowledged (uint16_t destNodeId, uint8_t port, uint8_t *payload, uint8_t payloadLen)

  send an acknowledged message to a node in the network If the host_ack_callback is set it will be called when the message recepient has acknowledged
- void send_wes_command (NcApiWesCmdValues cmd)

  Send a WES command to the node.
- void send_wes_respond (uint64_t uid, uint16_t nodeId)

  Send a wes response.
- void set_password (uint8_t new_password[5])

  Change the password the API should use to log into the NC module.
- bool switch_sapi_aapi ()

  Switch NeoCortec module to SAPI mode.
- bool login_sapi ()

  Log in.
- void change_node_id_sapi (uint16_t nodeid)

  Change modules node id.
- void write_raw (uint8_t *data, uint8_t length)

  Writes raw bytes to protocol uart.
- bool wait_for_sapi_response (tNcSapiMessage *message, uint32_t timeout_ms)

  Wait for system interface to send response.
- void start_bootloader ()

Send command to start bootloader.

- void start_protocol_stack ()

  Starts the protocol.

- bool get_setting (uint8_t setting, NcSetting *setting_ret)

  Gets a setting from the NC modules flash.

- void set_setting (uint8_t setting, uint8_t *setting_value, uint8_t setting_value_length)

  Changes a setting in the NC modules RAM.

- void **commit_settings** ()

  Move all settings in RAM to FLASH.

- void write_sapi_command (uint8_t cmd1, uint8_t cmd2, uint8_t *data, uint8_t data_length)

  Write a system interface command.

- bool change_setting (uint8_t setting, uint8_t *value, uint8_t length)

  Change a setting in the NC module.

- tNcModuleMode get_module_mode ()

  get module mode

Static Public Member Functions

- static void **pass_through_cts** ()

Public Attributes

- NeoMeshReadCallback **read_callback** = 0
- NeoMeshHostAckCallback **host_ack_callback** = 0
- NeoMeshHostAckCallback **host_nack_callback** = 0
- NeoMeshHostDataCallback **host_data_callback** = 0
- NeoMeshHostDataHapaCallback **host_data_hapa_callback** = 0
- NeoMeshWesSetupRequestCallback **wes_setup_request_callback** = 0
- NeoMeshWesStatusCallback **wes_status_callback** = 0

5.33.1 Detailed Description

Object that handles connection to NeoCortec module.

5.33.2 Member Function Documentation

5.33.2.1 change_node_id_sapi()

```
void NeoMesh::change_node_id_sapi (
            uint16_t nodeid )
```

Change modules node id.

Change ID of node. Important: This function will automatically enter SAPI mode and log in

Parameters

nodeid	The new node id

The documentation for this class was generated from the following files:

- NeoMesh.h
- NeoMesh.cpp

5.34 SAPIParser Class Reference

Class to parse messages from system interface.

`#include <SAPIParser.h>`

Public Member Functions

- void push_char (uint8_t c)

  Push new character to buffer.
- bool message_available ()

  See if a message is received but not yet read.
- tNcSapiMessage **get_pending_message** ()

  Get latest message message_available() should be called before this If no new message is received, the same message will be returned as last time this function was called.

5.34.1 Detailed Description

Class to parse messages from system interface.

The documentation for this class was generated from the following files:

- SAPIParser.h
- SAPIParser.cpp

5.35 tNcSapiMessage Struct Reference

Public Attributes

- uint8_t **command**
- uint8_t **data** [32]
- uint8_t **data_length**

The documentation for this struct was generated from the following file:

- SAPIParser.h

Chapter 6

File Documentation

6.1 NcApi.h

```
00001 /*
00002  Copyright (c) 2015, NeoCortec A/S
00003  All rights reserved.
00004
00005  Redistribution and use in source and binary forms, with or without
00006  modification, are permitted provided that the following conditions are met:
00007
00008  1. Redistributions of source code must retain the above copyright notice,
00009  this list of conditions and the following disclaimer.
00010
00011  2. Redistributions in binary form must reproduce the above copyright notice,
00012  this list of conditions and the following disclaimer in the documentation
00013  and/or other materials provided with the distribution.
00014
00015  3. Neither the name of the copyright holder nor the names of its contributors
00016  may be used to endorse or promote products derived from this software
00017  without specific prior written permission.
00018
00019  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00022  DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
00023  FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
00024  DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
00025  SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
00026  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
00027  OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00028  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00029  */
00030
00031 #ifndef NCAPI_H_
00032 #define NCAPI_H_
00033
00034 #include <stdint.h>
00035
00036 #ifdef __cplusplus
00037 extern "C" {
00038 #endif
00039
00040 #ifndef NCAPI_TXBUFFER_SIZE
00041 #define NCAPI_TXBUFFER_SIZE 32
00042 #define NCAPI_MAX_PAYLOAD_LENGTH (NCAPI_TXBUFFER_SIZE-5)
00043 #endif
00044
00045 #ifndef NCAPI_RXBUFFER_SIZE
00046 #define NCAPI_RXBUFFER_SIZE 255
00047 #endif
00048
00049 #define NCAPI_UNUSED(X) (void)X
00050
00051 typedef enum {
00052     NCAPI_OK = 0,
00053     NCAPI_ERR_NODEID = 1,
00054     NCAPI_ERR_DESTPORT = 2,
00055     NCAPI_ERR_PAYLOAD = 3,
00056     NCAPI_ERR_ENQUEUED = 4,
00057     NCAPI_ERR_NULLPAYLOAD = 5,
00058     NCAPI_ERR_NOARGS = 6,
```

```
00059     NCAPI_BUSY = 7,
00060     NCAPI_DATA_PENDING = 8
00061 } NcApiErrorCodes;
00062
00063 typedef enum {
00064     NCAPI_WES_STOP = 0,
00065     NCAPI_WES_STARTSERVER = 1,
00066     NCAPI_WES_REQUESTSTATUS = 2,
00067     NCAPI_WES_STARTCLIENT = 3,
00068 } NcApiWesCmdValues;
00069
00070 typedef enum {
00071     NCAPI_WES_STOPPED = 0,
00072     NCAPI_WES_SERVERRUNNING = 1,
00073     NCAPI_WES_CLIENTRUNNING = 2,
00074 } NcApiWesStatusValues;
00075
00076 typedef enum {
00077     NCAPI_NODE_TYPE_NC2400 = 1,
00078     NCAPI_NODE_TYPE_NC1000 = 2,
00079     NCAPI_NODE_TYPE_NC0400 = 3,
00080
00081 } NcApiNodeType;
00082
00083 typedef enum {
00084     NCAPI_ALT_STOP = 0,
00085     NCAPI_ALT_START = 1,
00086 } NcApiAltCmdValues;
00087
00088 typedef enum {
00089     NCAPI_NetCmd_ACK = 0,
00090     NCAPI_NetCmd_NACK = 1,
00091     NCAPI_NetCmd_Hibernate = 2,
00092     NCAPI_NetCmd_Wake = 3,
00093     NCAPI_NetCmd_Wes = 5,
00094 } NcApiNetCmdValues;
00095
00096
00097 typedef struct NcApiNeighbor {
00098     uint16_t nodeId;
00099     uint8_t RSSI;
00100 } tNcApiNeighbor;
00101
00102
00103 // ------------------------------------------------
00104 // Definitions of messages to be sent
00105 // and parameters for the related functions
00106 // ------------------------------------------------
00107
00108
00113 typedef struct NcApiSendUnackMessage {
00114     uint16_t destNodeId;
00115     uint8_t destPort;
00116     uint16_t appSeqNo;
00117     uint8_t * payload;
00118     uint8_t payloadLength;
00119 } tNcApiSendUnackMessage;
00120
00125 typedef struct NcApiSendUnackParams {
00126     tNcApiSendUnackMessage msg;
00127     void * callbackToken;
00128 } tNcApiSendUnackParams;
00129
00134 typedef struct NcApiSendAckMessage {
00135     uint16_t destNodeId;
00136     uint8_t destPort;
00137     uint8_t payloadLength;
00138     uint8_t * payload;
00139 } tNcApiSendAckMessage;
00140
00145 typedef struct NcApiSendAckParams {
00146     tNcApiSendAckMessage msg;
00147     void * callbackToken;
00148 } tNcApiSendAckParams;
00149
00154 typedef struct NcApiNodeInfoRequestMessage {
00155     void * dummy;
00156 } tNcApiNodeInfoRequestMessage;
00157
00162 typedef struct NcApiNodeInfoParams {
00163     tNcApiNodeInfoRequestMessage msg;
00164     void * callbackToken;
00165 } tNcApiNodeInfoParams;
00166
00171 typedef struct NcApiNeighborListRequestMessage {
00172     void * dummy;
00173 } tNcApiNeighborListRequestMessage;
```

```
00174
00179 typedef struct NcApiNeighborListRequestParams {
00180     tNcApiNeighborListRequestMessage msg;
00181     void * callbackToken;
00182 } tNcApiNeighborListRequestParams;
00183
00188 typedef struct NcApiNetCmdMessage {
00189     uint16_t destNodeId;
00190     uint8_t cmd;
00191     uint8_t * payload;
00192     uint8_t payloadLength;
00193 } tNcApiNetCmdMessage;
00194
00199 typedef struct NcApiNetCmdParams {
00200     tNcApiNetCmdMessage msg;
00201     void * callbackToken;
00202 } tNcApiNetCmdParams;
00203
00208 typedef struct NcApiWesCmdMessage {
00209     NcApiWesCmdValues cmd;
00210 } tNcApiWesCmdMessage;
00211
00216 typedef struct NcApiWesCmdParams {
00217     tNcApiWesCmdMessage msg;
00218     void * callbackToken;
00219 } tNcApiWesCmdParams;
00220
00225 #define WES_APPSETTINGS_LENGTH 24
00226 typedef struct NcApiWesResponseMessage {
00227     uint8_t uid[5];
00228     uint16_t nodeId;
00229     uint8_t appSettings[WES_APPSETTINGS_LENGTH];
00230 } tNcApiWesResponseMessage;
00231
00236 typedef struct NcApiWesResponseParams {
00237     tNcApiWesResponseMessage msg;
00238     void * callbackToken;
00239 } tNcApiWesResponseParams;
00240
00245 typedef struct NcApiAltCmdMessage {
00246     NcApiAltCmdValues cmd;
00247 } tNcApiAltCmdMessage;
00248
00253 typedef struct NcApiAltCmdParams {
00254     tNcApiAltCmdMessage msg;
00255     void * callbackToken;
00256 } tNcApiAltCmdParams;
00257
00258
00259 // ------------------------------------------------
00260 // Definitions of messages to be received
00261 // ------------------------------------------------
00262
00268 typedef struct NcApiHostAckNack {
00269     // Message
00270     uint16_t originId;
00271 } tNcApiHostAckNack;
00272
00273
00274
00275
00281 typedef struct NcApiHostUappStatus {
00282     // Message
00283     uint16_t originId;
00284     uint16_t appSeqNo;
00285 } tNcApiHostUappStatus;
00286
00287
00288
00289
00294 typedef struct NcApiHostData {
00295     uint16_t originId;
00296     uint16_t packageAge;
00297     uint8_t port;
00298     uint8_t payloadLength;
00299     uint8_t * payload;
00300 } tNcApiHostData;
00301
00306 typedef struct NcApiHostDataHapa {
00307     uint16_t originId;
00308     uint32_t packageAge;
00309     uint8_t port;
00310     uint8_t payloadLength;
00311     uint8_t * payload;
00312 } tNcApiHostDataHapa;
00313
00318 typedef struct NcApiHostUappData {
```

```
00319     uint16_t originId;
00320     uint16_t packageAge;
00321     uint8_t port;
00322     uint16_t appSeqNo;
00323     uint8_t payloadLength;
00324     uint8_t * payload;
00325 } tNcApiHostUappData;
00326
00331 typedef struct NcApiHostUappDataHapa {
00332     uint16_t originId;
00333     uint32_t packageAge;
00334     uint8_t port;
00335     uint16_t appSeqNo;
00336     uint8_t payloadLength;
00337     uint8_t * payload;
00338 } tNcApiHostUappDataHapa;
00339
00344 typedef struct NcApiNodeInfoReply {
00345     uint16_t nodeId;
00346     uint8_t uid[5];
00347     NcApiNodeType Type;
00348 } tNcApiNodeInfoReply;
00349
00354 typedef struct NcApiNeighborListReply {
00355     uint8_t NeighborsCount;
00356     tNcApiNeighbor Neighbor[12];
00357 } tNcApiNeighborListReply;
00358
00359
00364 typedef struct NcApiRouteInfoRequestReply {
00365     uint8_t Bitmap[16];
00366 } tNcApiRouteInfoRequestReply;
00367
00372 typedef struct NcApiNetCmdReply {
00373     uint16_t originId;
00374     NcApiNetCmdValues cmd;
00375     uint8_t payloadLength;
00376     uint8_t * payload;
00377 } tNcApiNetCmdReply;
00378
00383 typedef struct NcApiWesStatus {
00384     uint8_t Status;
00385 } tNcApiWesStatus;
00386
00391 typedef struct NcApiWesSetupRequest {
00392     uint8_t uid[5];
00393     uint8_t appFuncType;
00394 } tNcApiWesSetupRequest;
00395
00396
00397 // ------------------------------------------------
00398 // Definitions of API functions
00399 // ------------------------------------------------
00400
00409 NcApiErrorCodes NcApiStatus
00410 (
00411     uint8_t n
00412 );
00413
00426 void NcApiSupportMessageReceived(uint8_t n,void * callbackToken, uint8_t * msg, uint8_t msgLength);
00427
00432 void NcApiCallbackNwuActive(uint8_t n);
00433
00444 void NcApiCtsActive(uint8_t n);
00445
00461  NcApiErrorCodes NcApiSupportTxData(uint8_t n, uint8_t * finalMsg, uint8_t finalMsgLength);
00462
00471 void NcApiTxDataDone(uint8_t n);
00472
00485 void NcApiSupportMessageWritten(uint8_t n, void * callbackToken, uint8_t * finalMsg, uint8_t
      finalMsgLength);
00486
00496 void NcApiRxData(uint8_t n, uint8_t byte);
00497
00510 typedef void (*pfnNcApiReadCallback)(uint8_t n, uint8_t * msg, uint8_t msgLength);
00511
00524 typedef void (*pfnNcApiHostAckCallback)(uint8_t n, tNcApiHostAckNack * m);
00525
00526
00527
00540 typedef void (*pfnNcApiHostUappStatusCallback)(uint8_t n, tNcApiHostUappStatus * m);
00541
00553 typedef void (*pfnNcApiHostDataCallback)(uint8_t n, tNcApiHostData * m);
00554
00567 typedef void (*pfnNcApiHostDataHapaCallback)(uint8_t n, tNcApiHostDataHapa * m);
00568
00580 typedef void(*pfnNcApiHostUappDataCallback)(uint8_t n, tNcApiHostUappData * m);
```

```
00581
00594 typedef void(*pfnNcApiHostUappDataHapaCallback)(uint8_t n, tNcApiHostUappDataHapa * m);
00595
00603 typedef void (*pfnNcApiNodeInfoReplyCallback)(uint8_t n, tNcApiNodeInfoReply * m);
00604
00612 typedef void (*pfnNcApiNeighborListReplyCallback)(uint8_t n, tNcApiNeighborListReply * m);
00613
00621 typedef void (*pfnNcApiRouteInfoRequestReplyCallback)(uint8_t n, tNcApiRouteInfoRequestReply * m);
00622
00630 typedef void(*pfnNcApiNetCmdResponseCallback)(uint8_t n, tNcApiNetCmdReply * m);
00631
00639 typedef void (*pfnNcApiWesStatusCallback)(uint8_t n, tNcApiWesStatus * m);
00640
00648 typedef void (*pfnNcApiWesSetupRequestCallback)(uint8_t n, tNcApiWesSetupRequest * m);
00649
00650
00656 typedef struct NcApiRxHandlers {
00657     pfnNcApiReadCallback pfnReadCallback;
00658     pfnNcApiHostAckCallback pfnHostAckCallback;
00659     pfnNcApiHostAckCallback pfnHostNAckCallback;
00660     pfnNcApiHostUappStatusCallback pfnHostUappSendCallback;
00661     pfnNcApiHostUappStatusCallback pfnHostUappDropedCallback;
00662     pfnNcApiHostDataCallback pfnHostDataCallback;
00663     pfnNcApiHostDataHapaCallback pfnHostDataHapaCallback;
00664     pfnNcApiHostUappDataCallback pfnHostUappDataCallback;
00665     pfnNcApiHostUappDataHapaCallback pfnHostUappDataHapaCallback;
00666     pfnNcApiNodeInfoReplyCallback pfnNodeInfoReplyCallback;
00667     pfnNcApiNeighborListReplyCallback pfnNeighborListReplyCallback;
00668     pfnNcApiRouteInfoRequestReplyCallback pfnRouteInfoRequestReplyCallback;
00669     pfnNcApiNetCmdResponseCallback pfnNetCmdResponseCallback;
00670     pfnNcApiWesSetupRequestCallback pfnWesSetupRequestCallback;
00671     pfnNcApiWesStatusCallback pfnWesStatusCallback;
00672 } tNcApiRxHandlers;
00673
00674
00680 void NcApiInit(void);
00681
00693 NcApiErrorCodes NcApiSendUnacknowledged(uint8_t n, tNcApiSendUnackParams * args);
00694
00706 NcApiErrorCodes NcApiSendAcknowledged(uint8_t n, tNcApiSendAckParams * args);
00707
00716 NcApiErrorCodes NcApiSendNodeInfoRequest(uint8_t n, tNcApiNodeInfoParams * args);
00717
00726 NcApiErrorCodes NcApiSendNeighborListRequest(uint8_t n,void * callbackToken);
00727
00736 NcApiErrorCodes NcApiSendRouteInfoRequest(  uint8_t n,void * callbackToken);
00737
00746 NcApiErrorCodes NcApiSendNetCmd(uint8_t n, tNcApiNetCmdParams * args);
00747
00756 NcApiErrorCodes NcApiSendWesCmd(uint8_t n, tNcApiWesCmdParams * args);
00757
00766 NcApiErrorCodes NcApiSendWesResponse(uint8_t n, tNcApiWesResponseParams * args);
00767
00774 NcApiErrorCodes NcApiSendAltCmd(uint8_t n, tNcApiAltCmdParams * args);
00775
00788 void NcApiExecuteCallbacks(uint8_t n, uint8_t * msg, uint8_t msgLength);
00789
00794 void NcApiCancelEnqueuedMessage(uint8_t n);
00795
00796
00800 /* (Since the function is not supported, the description is excluded from Doxygen)
00801  *
00802  * \brief Sends one unknown message
00803  * @param n Index of tNcApi instance that the message should be sent via
00804  * @param args Pointer to instance of the argument structure that holds the parameters
00805  * @return 0 upon success. Anything else is an error
00806  */
00807 NcApiErrorCodes NcApiSendRaw(uint8_t n, tNcApiSendAckParams * args);
00808
00809
00816 typedef struct NcApi {
00817     uint8_t rxBuffer[ NCAPI_RXBUFFER_SIZE];
00818     uint16_t rxPosition;
00819     volatile uint8_t txMsgLen;
00820     uint8_t txBuffer[ NCAPI_TXBUFFER_SIZE];
00821     void * writeCallbackToken;
00822     volatile uint8_t recvBufIsSynced;
00823     tNcApiRxHandlers * NcApiRxHandlers;
00824 } tNcApi;
00825
00827 extern tNcApi g_ncApi[];
00828
00830 extern uint8_t g_numberOfNcApis;
00831
00832
00833 #ifdef __cplusplus
00834 }
```

```
00835 #endif
00836
00837 #endif /* NCAPI_H_ */
```

6.2 NeoMesh.h

```
00001 /*******************************************************************************
00002  * @file NeoMesh.h
00003  * @date 2023-08-03
00004  * @author Markus Rytter (markus.r@live.dk)
00005  *
00006  * @copyright Copyright (c) 2023
00007  *
00008  ******************************************************************************/
00009
00015 #ifndef NEOMESH_H
00016 #define NEOMESH_H
00017
00018 /*******************************************************************************
00019  *    Includes
00020  ******************************************************************************/
00021 #include <Stream.h>
00022 #include <Arduino.h>
00023 #include "NcApi.h"
00024 #include "SAPIParser.h"
00025
00026 /*******************************************************************************
00027  *    Defines
00028  ******************************************************************************/
00029
00030 #define DEFAULT_NEOCORTEC_BAUDRATE 115200
00031
00032 #define SAPI_COMMAND_HEAD 0x3E
00033 #define SAPI_COMMAND_TAIL 0x21
00034 #define SAPI_COMMAND_LOGIN1 0x01
00035 #define SAPI_COMMAND_LOGIN2 0x03
00036 #define SAPI_COMMAND_START_BOOTLOADER1 0x01
00037 #define SAPI_COMMAND_START_BOOTLOADER2 0x13
00038 #define SAPI_COMMAND_GET_SETTING_FLASH1 0x01
00039 #define SAPI_COMMAND_GET_SETTING_FLASH2 0x06
00040 #define SAPI_COMMAND_SET_SETTING1 0x01
00041 #define SAPI_COMMAND_SET_SETTING2 0x0A
00042 #define SAPI_COMMAND_COMMIT_SETTINGS1 0x01
00043 #define SAPI_COMMAND_COMMIT_SETTINGS2 0x08
00044 #define SAPI_COMMAND_START_PROTOCOL1 0x01
00045 #define SAPI_COMMAND_START_PROTOCOL2 0x12
00046
00047 #define NODE_ID_SETTING 0xA
00048 #define NETWORK_ID_SETTING 0x2A
00049 #define TRACE_OUTPUT_SETTING 0x2C
00050 #define GENERIC_APPLICATION_NORM_SETTING 0x19
00051 #define GENERIC_APPLICATION_ALT_SETTING 0x3A
00052
00053 #define DEFAULT_PASSWORD_LVL10 {0x4c, 0x76, 0x6c, 0x31, 0x30}
00054
00055 /*******************************************************************************
00056  *    Type defines
00057  ******************************************************************************/
00058
00059 typedef struct {
00060     uint8_t value[32];
00061     uint8_t length;
00062 } NcSetting;
00063
00067 typedef enum {
00068
00072     SAPI_LOGGED_OUT,
00073
00077     SAPI,
00078
00082     AAPI
00083 } tNcModuleMode;
00084
00096 typedef void (*NeoMeshReadCallback)(uint8_t * msg, uint8_t msgLength);
00097
00109 typedef void (*NeoMeshHostAckCallback)(tNcApiHostAckNack * m);
00110
00111
00112
00124 typedef void (*NeoMeshHostUappStatusCallback)(tNcApiHostUappStatus * m);
00125
00136 typedef void (*NeoMeshHostDataCallback)(tNcApiHostData * m);
00137
```

```
00149 typedef void (*NeoMeshHostDataHapaCallback)(tNcApiHostDataHapa * m);
00150
00161 typedef void(*NeoMeshHostUappDataCallback)(tNcApiHostUappData * m);
00162
00174 typedef void(*NeoMeshHostUappDataHapaCallback)(tNcApiHostUappDataHapa * m);
00175
00182 typedef void (*NeoMeshNodeInfoReplyCallback)(tNcApiNodeInfoReply * m);
00183
00190 typedef void (*NeoMeshNeighborListReplyCallback)(tNcApiNeighborListReply * m);
00191
00198 typedef void (*NeoMeshRouteInfoRequestReplyCallback)(tNcApiRouteInfoRequestReply * m);
00199
00206 typedef void(*NeoMeshNetCmdResponseCallback)(tNcApiNetCmdReply * m);
00207
00214 typedef void (*NeoMeshWesStatusCallback)(tNcApiWesStatus * m);
00215
00222 typedef void (*NeoMeshWesSetupRequestCallback)(tNcApiWesSetupRequest * m);
00223
00224
00225
00226
00227 /*****************************************************************************
00228  *    Class prototypes
00229  *****************************************************************************/
00230
00234 class NeoMesh
00235 {
00236 public:
00242     NeoMesh(Stream * serial, uint8_t cts_pin);
00243
00247     void start();
00248
00249     // IGNORE:
00250     void write(uint8_t *finalMsg, uint8_t finalMsgLength);
00251
00255     void update();
00256
00265     void change_node_id(uint16_t node_id);
00266
00274     void change_network_id(uint8_t network_id[16]);
00275
00284     void change_trace_output_setting(bool mode);
00285
00291     void set_baudrate(uint32_t baudrate);
00292
00301     void send_unacknowledged(uint16_t destNodeId, uint8_t port, uint16_t appSeqNo, uint8_t *payload,
      uint8_t payloadLen);
00302
00311     void send_acknowledged(uint16_t destNodeId, uint8_t port, uint8_t *payload, uint8_t payloadLen);
00312
00317     void send_wes_command(NcApiWesCmdValues cmd);
00318
00324     void send_wes_respond(uint64_t uid, uint16_t nodeId);
00325
00333     void set_password(uint8_t new_password[5]);
00334
00343     bool switch_sapi_aapi();
00344
00351     bool login_sapi();
00352
00359     void change_node_id_sapi(uint16_t nodeid);
00360
00366     void write_raw(uint8_t *data, uint8_t length);
00367
00376     bool wait_for_sapi_response(tNcSapiMessage * message, uint32_t timeout_ms);
00377
00383     void start_bootloader();
00384
00391     void start_protocol_stack();
00392
00402     bool get_setting(uint8_t setting, NcSetting * setting_ret);
00403
00414     void set_setting(uint8_t setting, uint8_t *setting_value, uint8_t setting_value_length);
00415
00419     void commit_settings();
00420
00429     void write_sapi_command(uint8_t cmd1, uint8_t cmd2, uint8_t * data, uint8_t data_length);
00430
00435     bool change_setting(uint8_t setting, uint8_t * value, uint8_t length);
00436
00441     tNcModuleMode get_module_mode();
00442
00443     NeoMeshReadCallback read_callback = 0;
00444     NeoMeshHostAckCallback host_ack_callback = 0;
00445     NeoMeshHostAckCallback host_nack_callback = 0;
00446     NeoMeshHostDataCallback host_data_callback = 0;
00447     NeoMeshHostDataHapaCallback host_data_hapa_callback = 0;
```

```
00448     NeoMeshWesSetupRequestCallback wes_setup_request_callback = 0;
00449     NeoMeshWesStatusCallback wes_status_callback = 0;
00450
00451     // IGNORE:
00452     static void pass_through_cts();
00453
00454 private:
00455     uint8_t uart_num;
00456     uint8_t cts_pin;
00457     uint32_t baudrate = DEFAULT_NEOCORTEC_BAUDRATE;
00458     Stream * serial;
00459     SAPIParser sapi_parser;
00460     tNcModuleMode module_mode = AAPI;
00461
00462     uint8_t password[5] = DEFAULT_PASSWORD_LVL10; // TODO: Create setter function
00463
00464     static void read_callback_(uint8_t n, uint8_t *msg, uint8_t msgLength);
00465     static void host_ack_callback_(uint8_t n, tNcApiHostAckNack *p);
00466     static void host_nack_callback_(uint8_t n, tNcApiHostAckNack *p);
00467     static void host_data_callback_(uint8_t n, tNcApiHostData *m);
00468     static void host_data_hapa_callback_(uint8_t n, tNcApiHostDataHapa *p);
00469     static void wes_setup_request_callback_(uint8_t n, tNcApiWesSetupRequest *p);
00470     static void wes_status_callback_(uint8_t n, tNcApiWesStatus *p);
00471 };
00472
00473 /*****************************************************************************/
00476 #endif // NEOMESH_H
```

6.3 NeoParser.h

```
00001 /*
00002 Copyright (c) 2015, NeoCortec A/S
00003 All rights reserved.
00004
00005 Redistribution and use in source and binary forms, with or without
00006 modification, are permitted provided that the following conditions are met:
00007
00008 1. Redistributions of source code must retain the above copyright notice,
00009    this list of conditions and the following disclaimer.
00010
00011 2. Redistributions in binary form must reproduce the above copyright notice,
00012    this list of conditions and the following disclaimer in the documentation
00013    and/or other materials provided with the distribution.
00014
00015 3. Neither the name of the copyright holder nor the names of its contributors
00016    may be used to endorse or promote products derived from this software
00017    without specific prior written permission.
00018
00019 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00020 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00021 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00022 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
00023 FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
00024 DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
00025 SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
00026 CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
00027 OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00028 OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00029 */
00030
00031 #ifndef NEOPARSER_H_
00032 #define NEOPARSER_H_
00033
00034 #include <stdint.h>
00035 #include "NcApi.h"
00036
00037 #ifdef __cplusplus
00038 extern "C"
00039 {
00040 #endif
00041
00042 typedef enum NcApiMessageType
00043 {
00044     // Sending
00045     CommandUnacknowledgedEnum = 0x02,
00046     CommandAcknowledgedEnum   = 0x03,
00047     NodeInfoRequestEnum       = 0x08,
00048     NeighborListRequestEnum   = 0x09,
00049     NetCmdEnum                = 0x0a,
00050     RouteInfoRequestEnum      = 0x0c,
00051     WesCmdEnum                = 0x10,
00052     WesResponseEnum           = 0x11,
00053     AltCmdEnum                = 0x20,
```

```
00054
00055     // Receiving
00056     HostAckEnum               = 0x50,
00057     HostNAckEnum              = 0x51,
00058     HostDataEnum              = 0x52,
00059     HostDataHapaEnum          = 0x53,
00060     HostUappDataEnum          = 0x54,
00061     HostUappDataHapaEnum      = 0x55,
00062     HostUappDataSend          = 0x56,
00063     HostUappDataDropped       = 0x57,
00064
00065     NodeInfoReplyEnum         = 0x58,
00066     NeighborListReplyEnum     = 0x59,
00067     NetCmdReplyEnum           = 0x5a,
00068     RouteInfoRequestReplyEnum = 0x5c,
00069     WesStatusEnum             = 0x60,
00070     WesSetupRequestEnum       = 0x61,
00071
00072     // Special
00073     CommandRawEnum            = 0xff,
00074     EnableSAPIOnAAPIUart      = 0x0B
00075 } NcApiMessageType;
00076
00077 #define NCAPI_HOST_PREFIX_SIZE 2
00078 #define NCAPI_HOSTACK_LENGTH 2
00079 #define NCAPI_HOSTDATA_HEADER_SIZE 5
00080 #define NCAPI_HOSTDATA_MIN_LENGTH (NCAPI_HOSTDATA_HEADER_SIZE+NCAPI_HOST_PREFIX_SIZE)
00081 #define NCAPI_HOSTDATAHAPA_HEADER_SIZE 7
00082 #define NCAPI_HOSTDATAHAPA_MIN_LENGTH (NCAPI_HOSTDATAHAPA_HEADER_SIZE+NCAPI_HOST_PREFIX_SIZE)
00083 #define NCAPI_HOSTUAPPDATA_HEADER_SIZE 7
00084 #define NCAPI_HOSTUAPPDATA_MIN_LENGTH (NCAPI_HOSTUAPPDATA_HEADER_SIZE+NCAPI_HOST_PREFIX_SIZE)
00085 #define NCAPI_HOSTUAPPDATAHAPA_HEADER_SIZE 9
00086 #define NCAPI_HOSTUAPPDATAHAPA_MIN_LENGTH (NCAPI_HOSTUAPPDATAHAPA_HEADER_SIZE+NCAPI_HOST_PREFIX_SIZE)
00087
00088 #define NCAPI_NODEINFOREQUEST_LENGTH 0
00089 #define NCAPI_NEIGHBORLISTREQUEST_LENGTH 0
00090 #define NCAPI_NETCMD_LENGTH unused
00091 #define NCAPI_WESCMD_LENGTH 1
00092 #define NCAPI_WESSETUPREQUEST_LENGTH 6
00093 #define NCAPI_ALTCMD_LENGTH 1
00094
00095 #define NCAPI_NODEINFOREPLY_LENGTH 8
00096 #define NCAPI_ROUTEINFOREQUESTREPLY_LENGTH 16
00097 #define NCAPI_WESRESPONSE_LENGTH (7 + WES_APPSETTINGS_LENGTH)
00098 #define NCAPI_WESSTATUS_LENGTH 1
00099 #define NCAPI_NEIGHBORLISTREPLY_LENGTH 36
00100 #define NCAPI_NEIGHBORLISTREPLY_EX_LENGTH 39
00101 #define NCAPI_NETCMDRESPONSE_HEADER_SIZE 3
00102 #define NCAPI_NETCMDRESPONSE_MIN_LENGTH 5
00103
00104
00113 int NcApiIsValidApiFrame(uint8_t * buffer, uint16_t bufLength, uint16_t * outStartAt, uint16_t *
     outLength);
00114
00120 void NcApiGetMsgAsHostAck(uint8_t * buffer, tNcApiHostAckNack * p);
00121
00122
00128 void NcApiGetMsgAsHostUappStatus(uint8_t * buffer, tNcApiHostUappStatus * p);
00129
00135 void NcApiGetMsgAsHostData(uint8_t * buffer, tNcApiHostData * p);
00136
00142 void NcApiGetMsgAsHostDataHapa(uint8_t * buffer, tNcApiHostDataHapa * p);
00143
00149 void NcApiGetMsgAsHostUappData(uint8_t * buffer, tNcApiHostUappData * p);
00150
00156 void NcApiGetMsgAsHostUappDataHapa(uint8_t * buffer, tNcApiHostUappDataHapa * p);
00157
00163 void NcApiGetMsgAsNodeInfoReply(uint8_t * buffer, tNcApiNodeInfoReply * p);
00164
00170 void NcApiGetMsgAsWesStatus(uint8_t * buffer, tNcApiWesStatus * p);
00171
00177 void NcApiGetMsgAsWesSetupRequest(uint8_t * buffer, tNcApiWesSetupRequest * p);
00178
00184 void NcApiGetMsgAsNodeInfo(uint8_t * buffer, tNcApiNodeInfoReply * p);
00185
00191 void NcApiGetMsgAsNeighborListReply(uint8_t * buffer, tNcApiNeighborListReply * p);
00192
00198 void NcApiGetMsgAsRouteInfoRequestReply(uint8_t * buffer, tNcApiRouteInfoRequestReply * p);
00199
00205 void NcApiGetMsgAsNetCmdResponse(uint8_t * buffer, tNcApiNetCmdReply * p);
00206
00207
00208 #ifdef __cplusplus
00209 }
00210 #endif
00211
00212
```

```
00213 #endif /* NEOPARSER_H_ */
```

6.4 SAPIParser.h

```
00001 /*******************************************************************************
00002  * @file SAPIParser.h
00003  * @date 2023-08-04
00004  * @author Markus Rytter (markus.r@live.dk)
00005  *
00006  * @copyright Copyright (c) 2023
00007  *
00008  ******************************************************************************/
00009
00015 #ifndef SAPI_PARSER_H
00016 #define SAPI_PARSER_H
00017
00018 /*******************************************************************************
00019  *    Includes
00020  ******************************************************************************/
00021
00022 #include <stdint.h>
00023
00024 /*******************************************************************************
00025  *    Defines
00026  ******************************************************************************/
00027
00028 #define SAPI_COMMAND_HEADER 0x3E
00029 #define SAPI_COMMAND_TAIL 0x21
00030 #define MINIMUM_MESSAGE_LENGTH 5
00031
00032 /*******************************************************************************
00033  *    Type defines
00034  ******************************************************************************/
00035
00036 typedef enum {
00037     LoginOK            = 0x80,
00038     LoginError         = 0x81,
00039     BootloaderStarted  = 0x82,
00040     ProtocolStarted    = 0x83,
00041     ProtocolError      = 0x84,
00042     ProtocolListOutput = 0x85,
00043     SettingValue       = 0x86
00044 } tNcApiSapiMessageType;
00045
00046 typedef struct {
00047     uint8_t command;
00048     uint8_t data[32];
00049     uint8_t data_length;
00050 } tNcSapiMessage;
00051
00052 /*******************************************************************************
00053  *    Class prototypes
00054  ******************************************************************************/
00058 class SAPIParser
00059 {
00060 public:
00065     void push_char(uint8_t c);
00066
00071     bool message_available();
00072
00079     tNcSapiMessage get_pending_message();
00080
00081 private:
00082     uint8_t buffer[64];
00083     tNcSapiMessage pending_message;
00084     bool is_message_pending = false;
00085     int cursor = 0;
00086
00087     void check_for_message();
00088     void parse_message();
00089 };
00090
00091 /*******************************************************************************/
00094 #endif  // SAPI_PARSER_H
```

Index