Instruction

Serial API Host Appl. Prg. Guide

Document No.:	INS12350
Version:	4
Description:	Guideline for developing serial API based host applications
Written By:	JFR;ABR;JSI;PSH;AES;ANI
Date:	
Reviewed By:	JFR;AES;NTJ;JSI;BBR
Restrictions:	Partners Only

Approved by:

This document is the property of Sigma Designs Inc. The data contained herein, in whole or in part, may not be duplicated, used or disclosed outside the recipient for any purpose. This restriction does not limit the recipient's right to use information contained in the data if it is obtained from another source without restriction.

CONFIDENTIAL

REVISION RECORD

Doc. Ver.	Date	By	Pages affected	Brief description of changes
1	20121026	ABR AES JSI JFR	ALL	Initial draft
2	20121109	AES	6.1	Initialization
3	20121130	ABR	6.4.2, 6.6.1 & 6.6.3	Added details to description of exception handling when receiving data frames.
4	20140602	JFR	7.1 5.4.5.1	Added application node information command Added funcID parameter description

CONFIDENTIAL

Table of Contents

CONFIDENTIAL

Table of Figures

Table of Tables

CONFIDENTIAL

1 ABBREVIATIONS

Abbreviation	Explanation
ACK	Acknowledgement
AES	The Advanced Encryption Standard is a symmetric block cipher algorithm. The AES is a NIST-standard cryptographic cipher that uses a block length of 128 bits and key lengths of 128, 192 or 256 bits. Officially replacing the Triple DES method in 2001, AES uses the Rijndael algorithm developed by Joan Daemen and Vincent Rijmen of Belgium.
ANZ	Australia/New Zealand
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CAN	Cancel
DLL	Dynamic Link Library
DUT	Device Under Test
EU	Europe
GNU	An organization devoted to the creation and support of Open Source software
HK	Hong Kong
HW	Hardware
IN	India
ISR	Interrupt Service Routines
JP	Japan
LRC	Longitudinal Redundancy Check
MY	Malaysia
NAK	Not Acknowledged
NWI	Network Wide Inclusion
PA	Power Amplifier
POR	Power On Reset
PRNG	Pseudo-Random Number Generator
PWM	Pulse Width Modulator
RF	Radio Frequency
RS-232	TIA-232-F Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange
RU	Russian Federation
SDK	Software Developer's Kit
SIS	SUC ID Server
SOF	Start Of Frame
SPI	Serial Peripheral Interface
SUC	Static Update Controller
US	United States
USB	Universal Serial Bus
USB CDC	Universal Serial Bus Communications Device Class
WUT	Wake Up Timer

2 INTRODUCTION

2.1 Purpose

The purpose of this document is to provide a user guide for host processor application development using the serial API interface. This guide covers the whole range of available Z-Wave chips/modules.

2.2 Audience and prerequisites

The audience of this document is Z-Wave partners and Sigma Designs.

2.3 Terms used in this document

This document describes mandatory and optional aspects of the required compliance of a Z-Wave product to the Z-Wave standard.

The guidelines outlined in IETF RFC 2119 [1] with respect to key words used to indicate requirement levels are followed. Essentially, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

3 OVERVIEW

The Serial Applications Programming Interface (Serial API) allows a host to communicate with a Z-Wave chip. The host may be PC or a less powerful embedded host CPU, e.g. in a remote control or in a gateway device. Depending on the actual chip family, the Serial API may be accessed via RS-232 or USB physical interfaces.

A number of sample applications demonstrate how to communicate with a Z-Wave chip via the Serial API.

The following host based sample applications are available on the SDK:

- PC Controller
  - Demonstrates Serial API features of the static controller API
- PC Installer Tool
  - Demonstrates Serial API features of the installer controller API
- PC Bridge
  - Demonstrates Serial API features of the bridge controller API
- ATMega Development Board Controller
  - Demonstrates Serial API features of the portable controller API

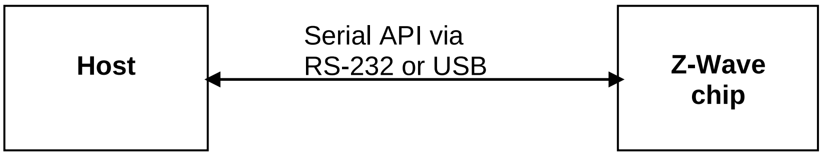


Figure 1. Communication via Serial API

The host based sample applications are described in the respective SDK overview documents. Refer to [2], [3] or [4].

The serial API leverages on the Z-Wave Protocol API. The serial API introduces additional messages related to inter-host communications. Mapping of serial API commands to the Z-Wave Protocol API calls can be found in [5], [6] or [7]. Dedicated serial API commands are presented in section 7.

Serial API based applications MUST ensure that the required features are available in the actual Z-Wave library, using the “Capabilities Command” see 7.1. Refer to [2], [3] or [4] for SDK library variants.

4 COMMUNICATION INTERFACE

The following sections describe the Serial API.

4.1 Communication Channel Settings

4.1.1 RS-232 Serial port

A host communicating to a Serial API library via a serial port MUST use the following settings.

Parameter	Value
Baud rate	115200 bits/s
Parity	No
Data bits	8
Stop bits	1

Table 1. Serial API RS-232 parameters

The least significant bit (LSB) b0 of each byte MUST be transmitted first on the physical wire.

4.1.2 USB Serial port

A host communicating to a Serial API library via a USB connection MUST obey the guidelines for the USB communications device class (USB CDC). In many cases, Linux distributions and MacOS releases will immediately present the Z-Wave chip USB interface as a serial port to applications.

Windows releases may need an .inf file structure in order to present the Z-Wave chip USB interface as a serial port to applications:

Key	Value
[Version]	Signature="$Windows NT$" Class=Ports ClassGuid={4D36E978-E325-11CE-BFC1-08002BE10318} Provider=%manu% DriverVer=02/17/2010,0.0.3.0
[Manufacturer]	%manu%=ZComDev, NTx86, NTamd64
[ZComDev.NTx86]	%dev%=ZComInst, USB\VID_0658&PID_0200
[ZComDev.NTamd64]	%dev%=ZComInst, USB\VID_0658&PID_0200
[ZComInst]	include=mdmcpq.inf CopyFiles=FakeModemCopyFileSection AddReg=LowerFilterAddReg,SerialPropPageAddReg
[ZComInst.Services]	include = mdmcpq.inf AddService = usbser, 0x00000002, LowerFilter_Service_Inst
[SerialPropPageAddReg]	HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider"
[Strings]	manu = "Sigma Designs" dev = "UZB" svc = "UZB"

Table 2. Serial API USB Windows .inf file structure

5 FRAME LAYOUT

The host and the Z-Wave chip (ZW) communicates via a simple protocol which uses four frame types: the ACK, NAK, CAN and Data frame types.

5.1 ACK frame

The ACK frame indicates that the receiving end received a valid Data frame.

The host MUST wait for an ACK frame after transmitting a Data frame to the Z-Wave chip.
In case of transmission errors or race conditions, the host may receive other frames or no frames at all. The host MUST be robust towards such events. The host SHOULD queue up requests for processing once the expected ACK frame has been received or timed out. The host MUST wait for a period of 1500ms before timing out waiting for the ACK frame.

A receiving Z-Wave chip MUST return an ACK frame in response to a valid Data frame.

7 (MSB)	6	5	4	3	2	1	0 (LSB)
ACK (0x06)							

Figure 2. ACK frame

5.2 NAK frame

The NAK frame indicates that the receiving end received a Data frame with errors.

If a transmitting host or Z-Wave chip receives a NAK frame in response to a Data frame, it MAY retransmit the Data frame.

7 (MSB)	6	5	4	3	2	1	0 (LSB)
NAK (0x15)							

Figure 3. NAK frame

A transmitting host or Z-Wave chip receiving a NAK frame MUST wait for a period before retransmitting the Data frame. Refer to 6.3

5.3 CAN frame

The CAN frame indicates that the receiving end discarded an otherwise valid Data frame. The CAN frame is used to resolve race conditions, where both ends send a Data frame and subsequently expects an ACK frame from the other end.

.7 (MSB)	6	5	4	3	2	1	0 (LSB)
CAN (0x18)							

Figure 4. CAN frame

If a Z-Wave chip expects to receive an ACK frame but receives a Data frame from the host, the Z-Wave chip SHOULD return a CAN frame. A host which receives a CAN frame MUST consider the dataframe lost. The host MUST wait for a period before retransmitting the Data frame. Refer to 6.3

5.4 Data frame

The Data frame contains the Serial API command including parameters for the command in question.

Each Data frame MUST be composed of a Serial API command including parameters for the command prepended with Start Of Frame (SOF), Length and Type fields and a Checksum byte appended.

A transmitting host or Z-Wave chip may time out waiting for an ACK frame after transmitting a Data frame. The transmitting end MUST wait for ACK frame for a period. If no ACK frame is received, the Data frame MAY be retransmitted; refer to 6.3.

7 (MSB)	6	5	4	3	2	1	0 (LSB)
SOF (0x01)							
Length							
Type							
Serial API Command ID							
Serial API Command Parameter 1							
…							
Serial API Command Parameter n							
Checksum							

Figure 5. Data frame

5.4.1 Start Of Frame (SOF)

The Start Of Frame (SOF) field is used for synchronization. The SOF field MUST have a value of 0x01. A host or a Z-Wave chip waiting for new traffic MUST ignore all other byte values than 0x06 (ACK), 0x15 (NAK), 0x18 (CAN) or 0x01 (Data frame). This way, both receivers will flush garbage bytes from the receive buffer to get back in sync after a connection glitch or a firmware restart in one of the ends.

5.4.2 Length

The Length field MUST report the number of bytes in the Data frame. The value of the Length Field MUST NOT include the SOF and Checksum fields.
A host or a Z-Wave chip receiving a Data frame SHOULD validate the length field by comparing the number of received bytes and the Length field (expecting a difference of 2 bytes).

5.4.3 Type

The Type field MUST indicate if the Data frame type is Request or Response.

Value	Type	Description
0x00	REQ	Request. This type MUST be used for unsolicited messages. API callback messages MUST use the Request type.
0x01	RES	Response. This type MUST be used for messages that are responses to Requests.
0x02..0xFF	*Reserved*	Reserved values MUST NOT be used. A receiving end MUST ignore reserved Type values.

Table 3. Data frame :: Type values

5.4.4 Serial API Command ID

The Serial API Command ID field MUST carry one of the valid API function codes defined in 7.
A host or Z-Wave chip MUST report the same Serial API Command ID in a response Data frame (refer to 5.4.3.

5.4.5 Serial API Command Parameters

The Serial API Command Parameters field MAY carry a variable number of bytes. The field MUST be at least one byte long. A receiving end MUST derive the actual number of bytes from the Length field; refer to 5.4.2.
Information carried in the Serial API Command Parameters field MUST comply with the API function prototype for the Serial API Command ID carried in the Serial API Command ID field; refer to 5.4.4

API function prototypes may be found in 7.

5.4.5.1 funcID Parameter

Some Serial API calls contain a funcID parameter. Any funcID value different from zero is returned in the callback function making it possible to correlate the callback with the original request. Setting funcID to zero disables callback function via serial API.

5.4.6 Checksum

The Checksum field MUST carry a checksum to enable frame integrity checks.
The checksum calculation MUST include the **Length**, **Type**, **Serial API Command Data** and **Serial API Command Parameters** fields.

The checksum value MUST be calculated as an 8-bit Longitudinal Redundancy Check (LRC) value.
The RECOMMENDED way to calculate the checksum is to initialize the checksum to 0xFF and then XOR each of the bytes of the fields mentioned above one at a time to the checksum value.

$$\text{Checksum} = \text{0xFF} \oplus \text{Length} \oplus \text{Type} \oplus \text{Cmd ID} \oplus \text{Cmd Parm[1]} \oplus \ldots \oplus \text{Cmd Parm[n]}$$

A Data frame MUST be considered invalid if it is received with an invalid checksum. Refer to 5.4.6.
A host or Z-Wave chip MUST return a NAK frame in response to an invalid Data frame.

6 TRANSMISSION

6.1 Initialization

To make sure the host and the Z-Wave module are in sync at application startup, the host should begin an initialization sequence. The initialization sequence differs a little depending on if the host has access to a module hard reset.

6.1.1 With hard reset

1) Close host serial port if it is open
2) Assert module reset
3) Open the host serial port at 115200 baud 8N1.
4) Release module reset
5) Wait 500ms

6.1.2 Without hard reset

1) Close host serial port if it is open
2) Open the host serial port at 115200 baud 8N1.
3) Send the NAK
4) Send SerialAPI command: FUNC_ID_SERIAL_API_SOFT_RESET
5) Wait 1.5s

This solution is not recommended because it rely on retrieval and execution of the SerialAPI command FUNC_ID_SERIAL_API_SOFT_RESET.

6.2 Frame timing

6.2.1 Data frame reception timeout

A receiving host or Z-Wave chip MUST abort reception of a Data frame if the reception has lasted for more than 1500ms after the reception of the SOF byte. A host or Z-Wave chip MUST NOT issue a NAK frame after aborting reception of a Data frame.

6.2.2 Data frame delivery timeout

A host or Z-Wave chip MUST wait for an ACK frame after transmitting a Data frame.
The receiver may be waiting for up to 1500ms for the remains of a corrupted frame (6.2.1).
Therefore, the transmitter MUST wait for at least 1600ms before deeming the Data frame lost.

The loss of a Data frame MUST be treated as the reception of a NAK frame; refer to 6.3.
The transmitter MAY compensate for the 1600ms already elapsed when calculating the retransmission waiting period.

6.3 Retransmission

A transmitter may time out waiting for an ACK frame after transmitting a Data frame or it may receive a NAK or a CAN frame. In either case, the transmitter SHOULD retransmit the Data frame.
A waiting period MUST be applied before the retransmission.

The waiting period MUST be calculated per the following formula:

$$T_{waiting} = 100ms + n*1000ms$$

where n is incremented at each retransmission. n=0 is used for the first waiting period.

A host or Z-Wave chip MUST NOT carry out more than 3 retransmissions. It should be noted that a host MAY choose to do a hard reset of the Z-Wave module if it is not able to do a successful frame delivery after 3 retransmissions. It is also recommended to flush/reopen the serial port after the 3 retransmissions.

6.4 Exception handling

6.4.1 Unresponsive Z-Wave module

In the unlikely event that the Z-Wave module becomes unresponsive for more than 4 seconds, it is RECOMMENDED to issue a hard reset of the module. A module may be deemed unresponsive if it has not responded with any character after three consecutive frame retransmissions, each with a 1600ms interval. See section 6.1

6.4.2 Persistent CRC errors

If a host application detects an invalid checksum three times in a row when receiving data frames, the host application SHOULD invoke a hard reset of the device. If a hard reset line is not available, a soft reset indication SHOULD be issued for the device.

6.4.3 Missing callbacks

I some situations a serial API callback may be lost due to an overflow in the UART transmit buffer. This condition may occur if a lot of unsolicited traffic comes in from the Z-Wave side. For this reason a SerialAPI based host application SHOULD guard all its callbacks with a timer. The timer values are given in INS11095 for each of the Z-Wave API functions which use callbacks.

6.5 Frame Flow

The frame flow between a host and a Z-Wave module (ZW) running the Serial API embedded sample code depends on the API call. There are two classes of communication between a host and a Z-Wave chip: Unsolicited and Request/Response. Each of the classes is presented below.

6.5.1 Unsolicited frame flow

The most basic frame flow is a Request (REQ) Data frame that is acknowledged by an ACK frame.

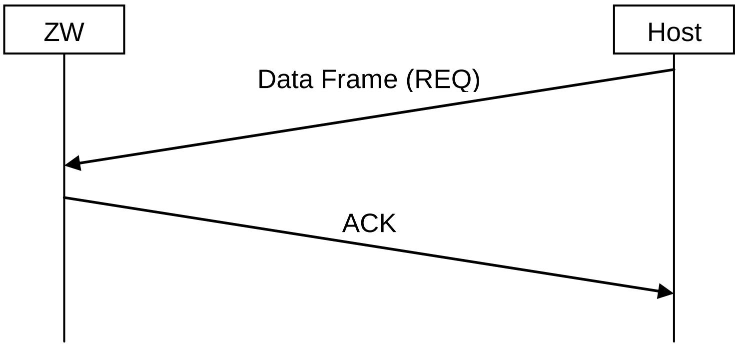


Figure 6. Unsolicited Data frame

An example of the frame flow outlined in the figure above could be the API call **ZW_SetExtIntLevel**.

A variant of the REQ Data frame flow is a request (REQ) Data frame in one direction followed by a request (REQ) Data frame in the opposite direction. The first Data frame is acknowledged before a Data frame is transmitted in the opposite direction.

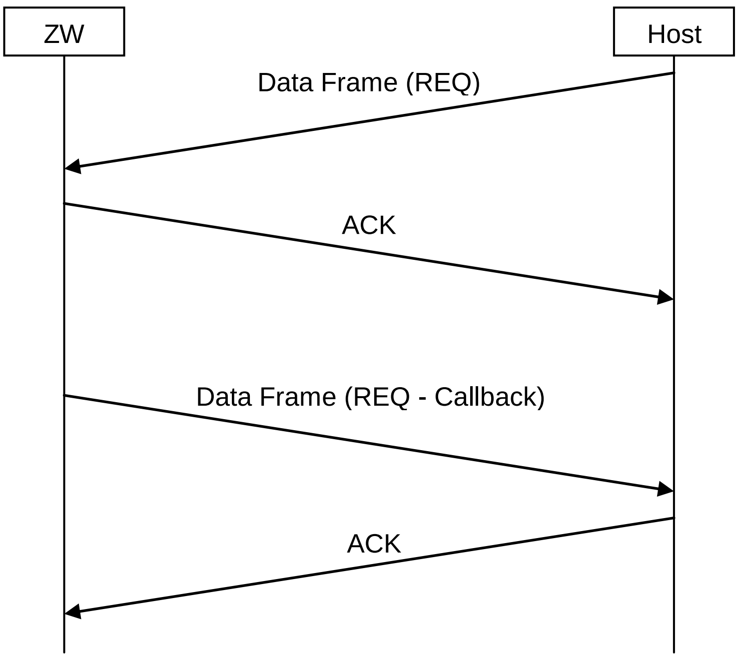


Figure 7. Unsolicited Data frame followed by unsolicited Data frame

Typically, the REQ Data frame in the opposite direction will follow after some time.

An example of the frame flow outlined in the figure above could be the API call **ZW_SetDefault**, where the second Data frame is carrying a callback message indicating the completion of the operation.

6.5.2 Request/Response frame flow

A Request (REQ) Data frame may be followed by a Result (RES) Data frame within an interval of a few seconds. This flow is used for all functions which have a non-void return value. Note that due to the simple nature of the simple acknowledge mechanism, only one REQ->RES session is allowed.

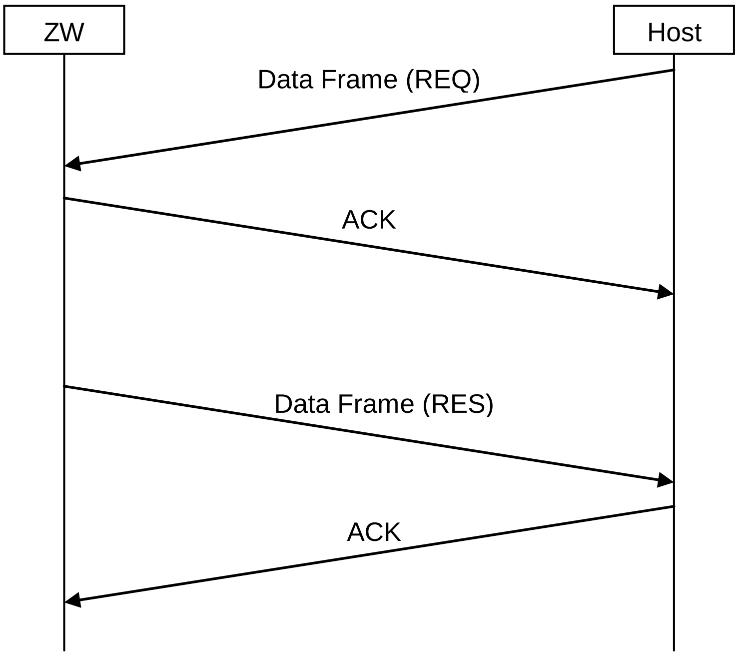


Figure 8. Request/Response Data frames

An example of the frame flow outlined in the figure above could be the API call **ZW_GetControllerCapabilities**, where the Result Data frame is carrying the requested controller capabilities.

A variant of the Request/Response Data frame flow is a flow where an unsolicited Data frame follows after the Request/Response Data frame pair. Typically, the REQ Data frame in the opposite direction will follow after some time.

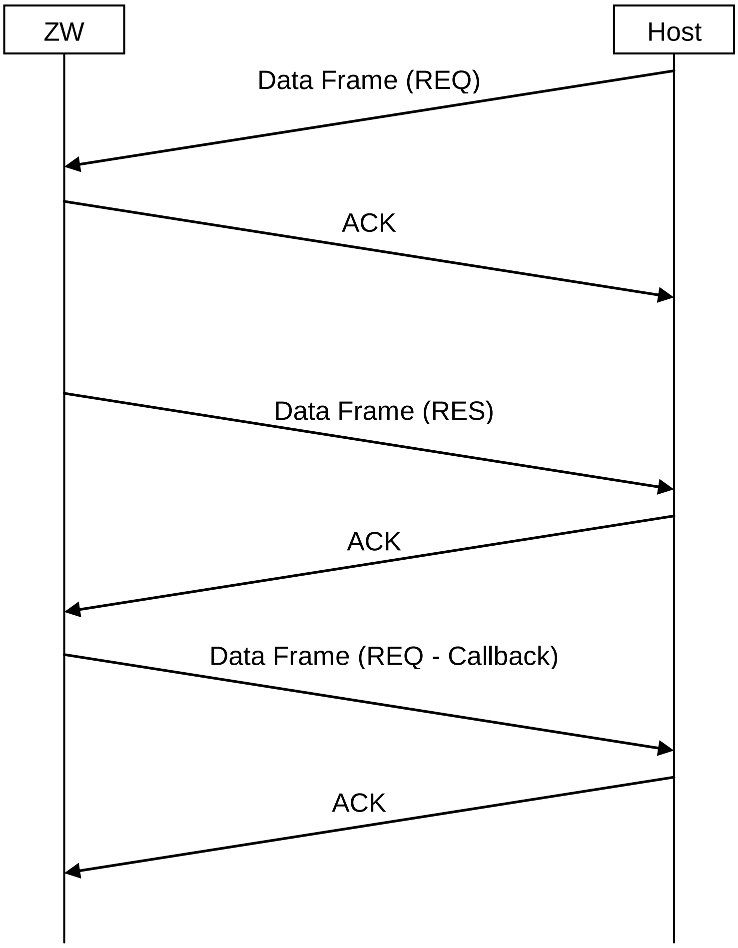


Figure 9. Request/Response Data frames followed by unsolicited Data frame

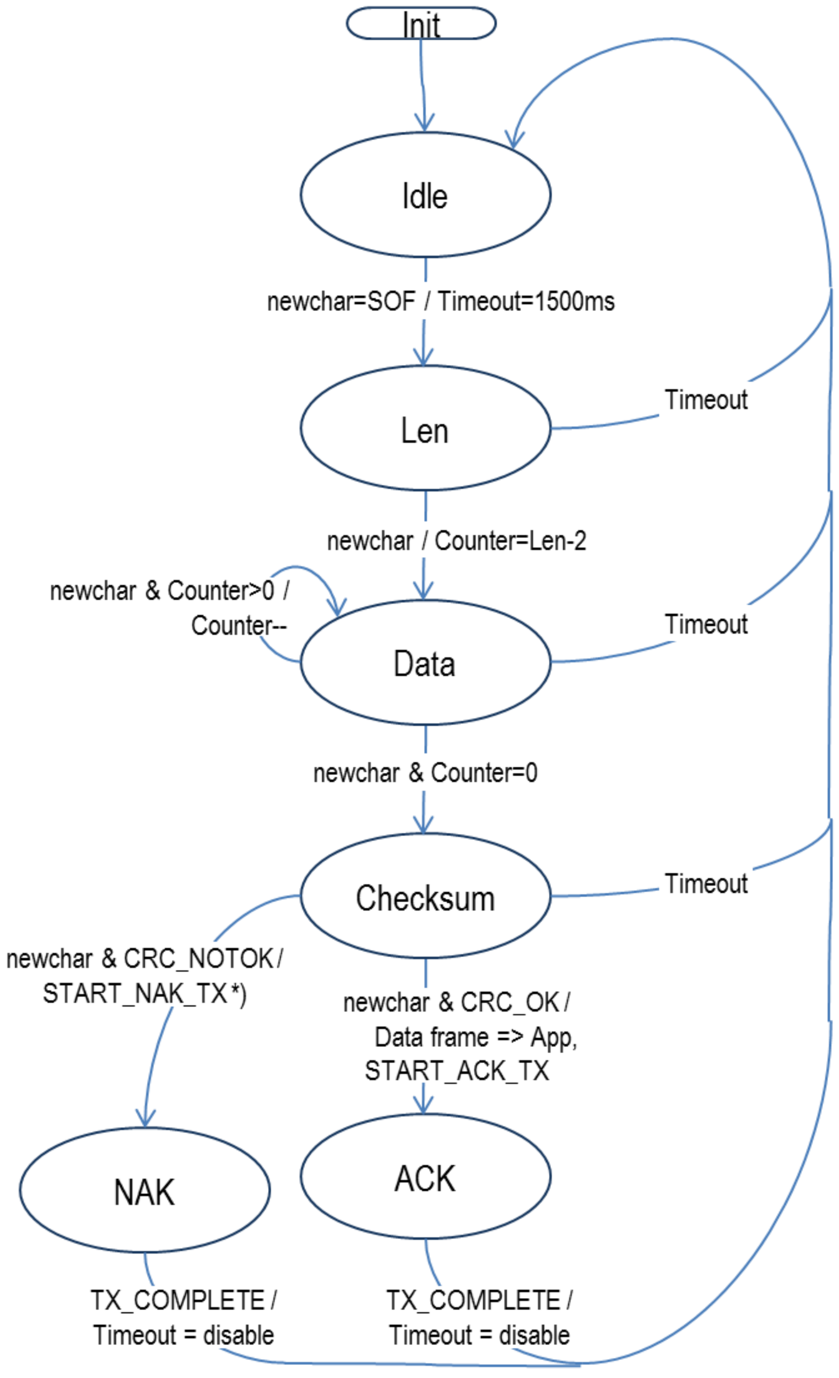
An example of the frame flow outlined in the figure above could be the API call **ZW_SendSUCID,** where the Result Data frame is carrying the requested controller capabilities and the second Data frame is carrying a callback message indicating the completion of the operation.

If a host application repeatedly receives a reception timeout error indication rather than a valid response data frame, the host application SHOULD invoke a hard reset of the device. If a hard reset line is not available, a soft reset indication SHOULD be issued for the device.

6.6 State Diagrams

This chapter outlines a transmission and reception of Control and Data frames.

6.6.1 Host Data Frame Reception



*) If a host application detects an invalid checksum three times in a row when receiving data frames, the host application SHOULD invoke a hard reset of the device. If a hard reset line is not available, a soft reset indication SHOULD be issued for the device.

Figure 10. Host Data Frame Reception

States	
Idle	Waiting for a new char Ignore all other values than the SOF code. Event: newchar & [SOF] => New state: <Len> Actions: Enable receive timeout timer
Len	Waiting for the Len byte Event: newchar => New state: <Data> Actions: Set counter=Len-2 Event: timeout => New state: <Idle>
Data	Waiting for data: Type, Cmd and parameter fields. Information is passed on to the Serial API command handler when validated Event: newchar & Counter>0 => New state: <Data> Actions: Counter-- Event: newchar&Counter=0 => New state: <Checksum> Actions: (none) Event: timeout => New state: <Idle>
Checksum	Waiting for the Checksum byte Event: newchar & CRC_NOTOK => New state: <NAK> Actions: Initiate transmission of NAK frame Event: newchar & CRC_NOTOK => New state: <ACK> Actions: Forward Data frame to Serial API command handler Initiate transmission of ACK frame If a host application detects an invalid checksum three times in a row when receiving data frames, the host application SHOULD invoke a hard reset of the device. If a hard reset line is not available, a soft reset indication SHOULD be issued for the device.
NAK	Waiting for transmission of NAK frame Event: TX completion => New state: <Idle> Actions: Disable timeout
ACK	Waiting for transmission of ACK frame Event: TX completion => New state: <Idle> Actions: Disable timeout

6.6.1.1 Counter maintenance

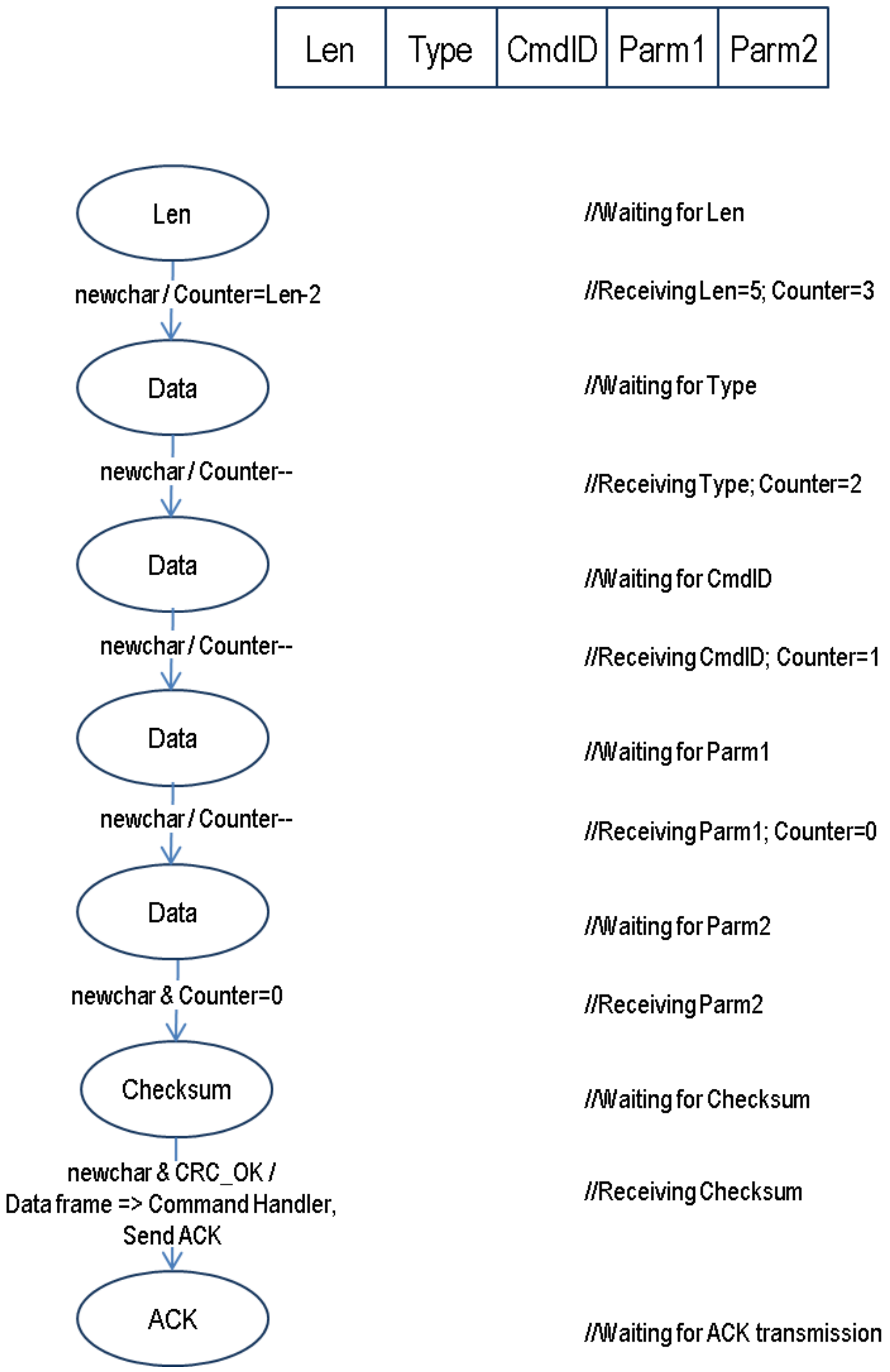


Figure 11. Counter Maintenance

6.6.2 Host Media Access Control

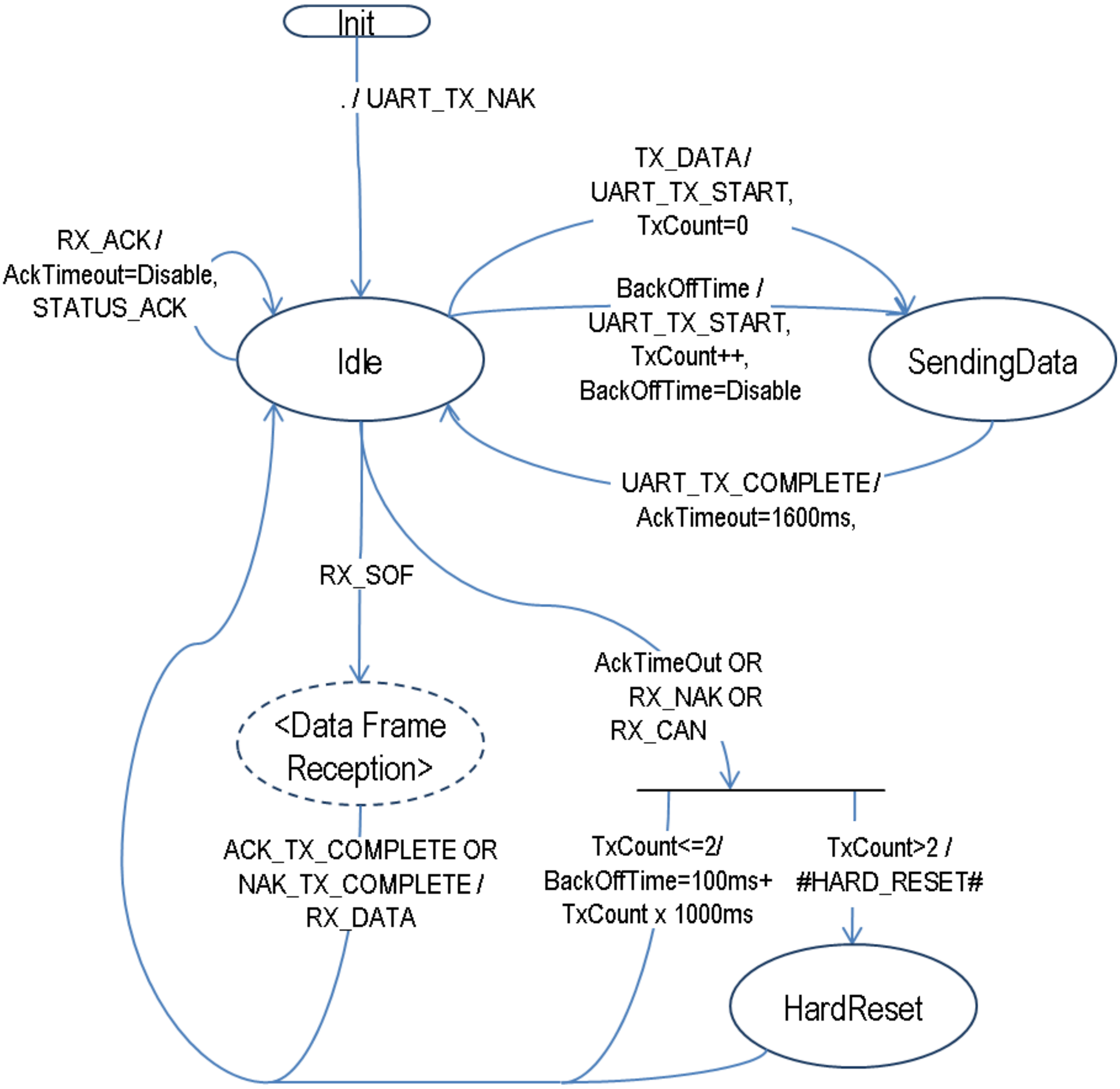


Figure 12. Host Media Access Control

States	

Idle	Waiting for events
	
	Event: (Init) => // Send NAK frame to force the remote end to re-send a Data frame if such
	// a frame is waiting for acknowledgment.
	
	Event: TX_DATA =>
	New state: <SendingData>
	Actions: Generate UART_TX_START event
	Initialize TxCount retransmission counter
	
	Event: BackOffTime =>
	New state: < SendingData >
	Actions: Generate UART_TX_START event
	Disable BackOffTime timer
	Increment TxCount retransmission counter
	
	Event: AckTimeOut OR RX_NAK OR RX_CAN =>
	New state: <Idle>
	Actions: IF (TxCount<=2) THEN
	Set BackOffTime = 100ms + TxCount x 1000ms
	ELSE
	Do a module hard reset
	ENDIF
	
	Event: RX_SOF =>
	New state: <Data Frame Reception> - SEPARATE CHART
	Actions: (none)
	
	Event: RX_ACK =>
	New state: <Idle>
	Actions: Disable ACK timeout timer

SendingData	Waiting for frame transmission to be completed Event: UART_TX_COMPLETE => New state: <Idle> Actions: Set ACK timeout = 1600ms
<Data Frame Reception>	Waiting for data frame reception to be completed <Data Frame Reception> is a self-contained state diagram. Stay here until finished. Event: ACK_TX_COMPLETE OR NAK_TX_COMPLETE => New state: <Idle> Actions: (none)
HardReset	Waiting for Hard Reset to be invoked Event: (Hard Reset) => New state: <Idle> Actions: (none)

6.6.3 Host Request/Response Session

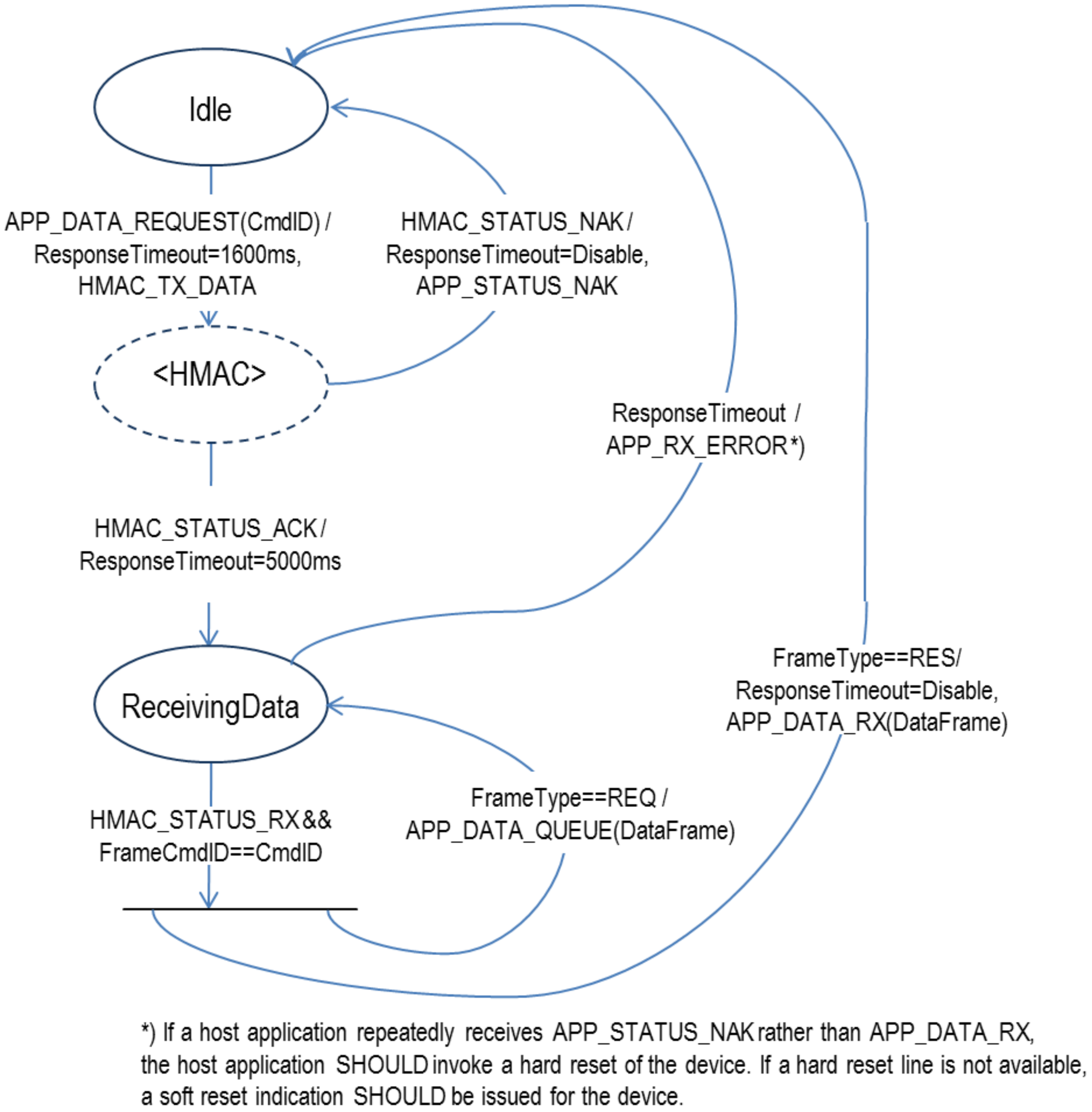


Figure 13. Host Request/Response Session

States	
Idle	Waiting for application request Event: APP_DATA_REQUEST(CmdID) => New state: <Host Media Access Control (HMAC)> Actions: Enable response timeout timer for ACK timeout period = 1600ms, Generate HMAC_TX_DATA event for <HMAC> state diagram.
<HMAC>	Waiting for transmission and acknowledgment of Data frame <HMAC> is a self-contained state diagram. Stay here until finished. Event: HMAC_STATUS_ACK => New state: <ReceivingData> Actions: Enable response timeout timer for response timeout period = 5000ms Event: HMAC_STATUS_NAK => New state: <Idle>, Actions: Disable response timeout timer, Generate APP_STATUS_NAK event for application.

ReceivingData	Waiting for response Data frame
	
	Event: HMAC_STATUS_RX && FrameCmdID<>CmdID && FrameType==REQ =>
	New state: <ReceivingData>
	Actions: Generate APP_DATA_QUEUE(Data frame) event for application.
	// Queue Data frame for later processing. Keep waiting for response.
	
	Event: HMAC_STATUS_RX && FrameCmdID<>CmdID && FrameType==RES =>
	New state: <Idle>
	Actions: Disable response timeout timer,
	Generate APP_DATA_RX(Data frame) event for application
	// Notify application that a response Data frame was received
	
	Event: ResponseTimeout =>
	New state: <Idle>
	Actions: Notify application that a response Data frame was not received
	NOTE: If a host application repeatedly receives APP_RX_ERROR rather than APP_DATA_RX, the host application SHOULD invoke a hard reset of the device. If a hard reset line is not available, a soft reset indication SHOULD be issued for the device.

7 SERIAL API COMMANDS

Besides the Z-Wave API function calls described in “Z-Wave application programmers’ guide” the SerialAPI support a set of additional commands.

7.1 Application Node Information Command

As of Serial API protocol version 4 it is possible to call Serial API Application Node Information Command to store a new Node Information Frame (NIF). Prior to either start or join a Z-Wave network the HOST needs to initially setup the Node Information Frame (NIF) which should define the type of Z-Wave node the SerialAPI module is supposed to be. For the NIF to be stored in the protocol NVM area as well as in the application NVM area the HOST need to perform the following steps:

1. HOST->ZW: send **SerialAPI_ApplicationNodeInformation()** with NIF information

2. HOST->ZW: send **ZW_SetDefault()**

Serial API:

HOST->ZW: REQ **|** 0x03 **|** deviceOptionsMask **|** generic **|** specific **|** parmLength **|** nodeParm[ ]

For more details, refer to relevant Application Programming Guide [5], [6], [7] or [8].

7.2 Capabilities Command

As of Serial API protocol version 4 it is possible to call Serial API Capabilities Command to determine exactly which Serial API functions a specific Serial API Z-Wave Module supports with the FUNC_ID_SERIAL_API_GET_CAPABILITIES Serial API function:

Serial API:

HOST->ZW: REQ | 0x07

ZW->HOST: RES | 0x07 | SERIAL_APPL_VERSION | SERIAL_APPL_REVISION | SERIALAPI_MANUFACTURER_ID1 | SERIALAPI_MANUFACTURER_ID2 | SERIALAPI_MANUFACTURER_PRODUCT_TYPE1 | SERIALAPI_MANUFACTURER_PRODUCT_TYPE2 | SERIALAPI_MANUFACTURER_PRODUCT_ID1 | SERIALAPI_MANUFACTURER_PRODUCT_ID2 | FUNCID_SUPPORTED_BITMASK[ ]

SERIAL_APPL_VERSION is the Serial API application Version number.

SERIAL_APPL_REVISION is the Serial API application Revision number.

SERIALAPI_MANUFACTURER_ID1 is the Serial API application manufacturer_id (MSB).

SERIALAPI_MANUFACTURER_ID2 is the Serial API application manufacturer_id (LSB).

SERIALAPI_MANUFACTURER_PRODUCT_TYPE1 is the Serial API application manufacturer product type (MSB).

SERIALAPI_MANUFACTURER_PRODUCT_TYPE2 is the Serial API application manufacturer product type (LSB).

SERIALAPI_MANUFACTURER_PRODUCT_ID1 is the Serial API application manufacturer product id (MSB).

SERIALAPI_MANUFACTURER_PRODUCT_ID2 is the Serial API application manufacturer product id (LSB).

FUNCID_SUPPORTED_BITMASK[ ] is a bitmask where every Serial API function ID which is supported has a corresponding bit in the bitmask set to ‘1’. All Serial API function IDs which are not supported have their corresponding bit set to ‘0’. First byte in bitmask corresponds to FuncIDs 1-8 where bit 0 corresponds to FuncID 1 and bit 7 corresponds to FuncID 8. Second byte in bitmask then corresponds to FuncIDs 9-16 and so on.

7.3 Node List Command

As of Serial API protocol version 4 it is possible to call Serial API Node List Command to determine Serial API protocol version number, Serial API capabilities, nodes currently stored in the external NVM (only controllers) and chip used in a specific Serial API Z-Wave Module with the FUNC_ID_SERIAL_API_GET_INIT_DATA Serial API function:

Serial API:

HOST->ZW: REQ | 0x02

(Controller) ZW->HOST: RES | 0x02 | ver | capabilities | 29 | nodes[29] | chip_type | chip_version

(Slave) ZW->HOST: RES | 0x02 | ver | capabilities | 0 | chip_type | chip_version

The parameter 'ver' is the Serial API application Version number.

The parameter 'capabilities' is a byte holding various flags describing the actual mode.

Capabilities flags:
Bit 0: 0 = Controller API; 1 = Slave API
Bit 1: 0 = Timer functions not supported; 1 = Timer functions supported.
Bit 2: 0 = Primary Controller; 1 = Secondary Controller
Bit 3-7: reserved

Timer functions supported comprises of TimerStart, TimerRestart and TimerCancel.

'29' or '0' specifies the length of 'nodes[]' array

nodes[29] is a node bitmask. The chip used can be determined as follows:

Z-Wave Chip	Chip_type	Chip_version
ZW0102	0x01	0x02
ZW0201	0x02	0x01
ZW0301	0x03	0x01
ZM0401	0x04	0x01
ZM4102	0x04	0x01
SD3402	0x04	0x01
ZW050x	0x05	0x00

7.4 Set Timeouts Command

The timeout in the Serial API (as of SerialAPI version 4) can be set by using the FUNC_ID_SERIAL_API_SET_TIMEOUTS Serial API function:

Serial API:

HOST->ZW: REQ | 0x06 | RXACKtimeout | RXBYTEtimeout

ZW->HOST: RES | 0x06 | oldRXACKtimeout | oldRXBYTEtimeout

7.5 Power Management Commands

The Serial API Power Management Commands is designed for use in a system where a Z-Wave module is connected to a host CPU system via a serial port and a number of I/O pins are used for control of the power to the Host CPU system.

7.5.1 Overview

The power management API is designed for use in a system where a Z-Wave module is connected to a host CPU system via a serial port and a number of I/O pins are used for control of the power to the Host CPU system.

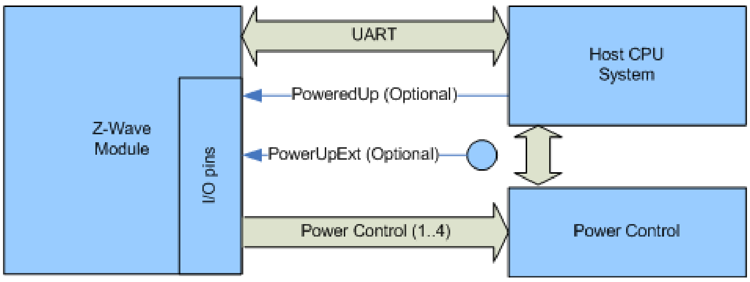


Figure 14. Power Management system

In a system like this it is necessary to have a communication protocol between the two CPU systems that ensures that the correct power state is selected and the Z-Wave module and the host CPU system always is in agreement about what power state they are using at all times.

All power management configuration and setup is done runtime using the serial API interface from the host processor system. The Z-Wave module must therefore be powered at all times in the system and decisions to power down the system always comes from the host CPU system. Power management is also possible on a Z-Wave module without external non-volatile memory.

7.5.1.1 I/O pins

A number of I/O pins on the Z-Wave module and the host processor system can be used for the power management API. No GPIO pins will be configured or changed before the host CPU configures the pin. All GPIO pins will be in their reset state (input, pull up enabled) until the host CPU issues an serial API command that configures or change status of a pin.

All GPIO's used as input on the Z-Wave module must be asserted for at least 20ms when changing level to allow the firmware to detect the change of the input pin status.

PoweredUp pin (Optional)

An input pin on the Z-Wave module is needed to communicate from the host processor to the Z-Wave module that the host processor system is now ready to be powered down. This pin is necessary if the host CPU system is not able to send commands on the UART during the power down sequence because the UART driver or the OS has been stopped. If configured the PoweredUp pin is set active on system power on.

Z-Wave module	Input
Host CPU	Output

PowerCtrl(1..4)

The PowerCtrl pins are used to control the power management hardware from the Z-Wave module.

Z-Wave	Output
Host CPU	N/A

7.5.1.2 Power management configuration sequence

When the serial API starts up for the first time it assumes that there is no power management present. The power management is activated in the Z-Wave module by configuring the power up mode.

See section 7.1 for a detailed description of the serial API commands.

When configuring the power management the following sequence of events should happen:

- The host configures the PoweredUp pin by using the Serial API Power Management Pin Configuration command. (Optional)
- The host configures the Power Up PowerCtrl pin(s) by using the Serial API Power Management Pin Configuration command
- The host configures the Wake up criteria's by using the Power Up on Z-Wave Configuration Command (see section 7.5.4) and/or the Power Up on Timer Configuration Command (see section 7.5.5).

7.5.1.3 Power up sequence

When powering up the following sequence of events should happen:

1. The Z-Wave module receives a command via RF that triggers a power up of the system.
2. The Z-Wave module changes the state of the power control I/O pins to the POWER_MODE_RUNNING state
3. The Z-Wave module waits for the Serial API Ready command on the UART
4. The host CPU system powers up and sets the PoweredUp pin active. (Optional)

5. When ready the host CPU system sends the serial API Ready command.

6. When the Ready command is received the Z-Wave module sends the command that triggered the power up to the host CPU system.

7.5.1.4 Power down sequence

When powering down the following sequence of events should happen:

1. The host must have performed the configuration sequence specified in section 7.5.1.2

2. The host processor determines that the system should power down now (based on, activity, timer, received commands, etc.)

3. The host processor sends an Serial API Set Power Mode command to the Z-Wave module

4. The Z-Wave module starts to monitor the PoweredUp pin (if configured) and continues to next state in power down sequence when the PoweredUp pin goes NOT active.

5. The Z-Wave module changes the state of the power control I/O pins according to the power mode requested by the host.

7.5.1.5 Power modes

The power management API supports any number of power modes that the host CPU system wants to use. The power modes can be divided into 2 different groups:

POWER_MODE_RUNNING

In power mode running the host CPU system is running. The host CPU system can receive commands send from the Z-Wave module on the UART.

POWER_MODE_POWERDOWN

In power mode power down the host CPU system is unable to receive commands send on the UART. All Z-Wave RF commands received by the Z-Wave module will be discarded if they do not trigger a wakeup. The only transition of power mode from this mode it to go to the POWER_MODE_RUNNING.

7.5.2 Pin Configuration Command

The Pin Configuration Command is used to map the power management input pin PoweredUp to a physical IO pin.

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">FUNC_ID_POWER_MANAGEMENT</td></tr>
<tr><td colspan="8">PM_PIN_UP_CONFIGURATION_CMD</td></tr>
<tr><td colspan="8">IO Pin</td></tr>
<tr><td colspan="8">Active Level</td></tr>
</table>

IO pin (8bit):

The IO pin field specifies the physical I/O pin that should be used for this signal. The table of I/O pins is shown below

IO Pin defines	Value
PM_PHYSICAL_PIN_P00	0x00
PM_PHYSICAL_PIN_P01	0x01
PM_PHYSICAL_PIN_P02	0x02
PM_PHYSICAL_PIN_P03	0x03
PM_PHYSICAL_PIN_P04	0x04
PM_PHYSICAL_PIN_P05	0x05
PM_PHYSICAL_PIN_P06	0x06
PM_PHYSICAL_PIN_P07	0x07
PM_PHYSICAL_PIN_P10	0x10
PM_PHYSICAL_PIN_P11	0x11
PM_PHYSICAL_PIN_P12	0x12
PM_PHYSICAL_PIN_P13	0x13
PM_PHYSICAL_PIN_P14	0x14
PM_PHYSICAL_PIN_P15	0x15
PM_PHYSICAL_PIN_P16	0x16
PM_PHYSICAL_PIN_P17	0x17
PM_PHYSICAL_PIN_P22	0x22
PM_PHYSICAL_PIN_P23	0x23
PM_PHYSICAL_PIN_P24	0x24
PM_PHYSICAL_PIN_P30	0x30
PM_PHYSICAL_PIN_P31	0x31
PM_PHYSICAL_PIN_P32	0x32
PM_PHYSICAL_PIN_P33	0x33
PM_PHYSICAL_PIN_P34	0x34
PM_PHYSICAL_PIN_P35	0x35
PM_PHYSICAL_PIN_P36	0x36
PM_PHYSICAL_PIN_P37	0x37

Active Level (8bit):

The level the PoweredUp pin should have when it is active. Optional and not given then active state defaults to active Low.

0 – Low

1 – High

7.5.3 Power up Mode Configuration Command

The Power up Mode Configuration Command is used to configure the state of the PowerCtrl pins when the Serial API has to power up the host CPU system

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_MODE_CONFIGURATION_CMD							
Number of Pins (max 4)							
IO Pin 1							
Level 1							
IO Pin ..							
Level ..							
IO Pin x							
Level x							

Number of Pins (8 bit):

The number of pins that is contained in the command. The max number of pins is 4

IO Pin x (8 bit):

The physical pin that should be changed when the Serial API has to wake up the host CPU system. A full list of physical pins can be found in section 7.5.2.

Level x (8 bit):

The level the output pin should have when the specified power mode is set.

0 – Low

1 – High

7.5.4 Power Up on Z-Wave Configuration Command

The Power Up on Z-Wave Configuration Command is used to specify what Z-Wave command that should trigger a power up of the host CPU system. All Z-Wave commands received are checked if they match the wakeup values and masks configured.

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_ZWAVE_CONFIGURATION_CMD							
Wakeup Match Mode							
Number of match bytes (max 8)							
Wakeup Value 1							
Wakeup Value ..							
Wakeup Value x							
Wakeup Mask 1							
Wakeup Mask ..							
Wakeup Mask x							

Wakeup Match Mode (8bit):

PM_WAKEUP_ALL

Wake up on all Z-Wave application commands received by the Z-Wave module.

PM_WAKEUP_ALL_NO_BROADCAST

Wake up on all Z-Wave application commands received by the Z-Wave module, except frames send as broadcast frames.

PM_WAKEUP_MASK

Wake up the host CPU when receiving a Z-Wave command where the first 5 bytes of the frame matches the specified value and mask.

Wakeup Mode define	Value
PM_WAKEUP_ ALL	0x01
PM_WAKEUP_ALL_NO_BROADCAST	0x02
PM_WAKEUP_MASK	0x03

Number of Match Bytes (8bit):

Number of bytes used to match an incoming Z-Wave command with, to see if it should trigger a wakeup. The max number of match bytes is 8.

Wakeup Value n (8bit*x):

The wakeup value is the value an incoming Z-Wave frame should be checked against to see if it should trigger a wakeup.

Wakeup Mask n (8 bit*x):

The wakeup mask is a mask that can be used to mask out bits or bytes in the received Z-Wave frame before it is compared with the Wakeup value.

The Wakeup value and Wakeup mask are checked like this in the Serial API

If ((Z-Wave Frame & Wakeup Mask) == Wakeup Value)

DoWakeup();

Example:

If the host CPU wants to trigger a wakeup on an Simple AV Set command with the Command Power the following command should be send to the Z-Wave module.

The simple AV Set command has the following structure:

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">COMMAND_CLASS_SIMPLE_AV_CONTROL</td></tr>
<tr><td colspan="8">SIMPLE_AV_CONTROL_SET</td></tr>
<tr><td colspan="8">Sequence Number</td></tr>
<tr><td colspan="5">Reserved</td><td colspan="3">Key Attributes</td></tr>
<tr><td colspan="8">Item ID MSB</td></tr>
<tr><td colspan="8">Item ID LSB</td></tr>
<tr><td colspan="8">AV Command MSB,1</td></tr>
<tr><td colspan="8">AV Command LSB,1</td></tr>
</table>

In this Z-Wave command we want to match the command class, the command, the key attributes and the AV command. We do not care about the sequence number, the reserved field and the item ID. So the Power Up on Z-Wave command would look like this:

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_ZWAVE_CONFIGURATION_CMD							
PM_WAKEUP_MASK							
8 (Match the 8 first bytes)							
COMMAND_CLASS_SIMPLE_AV_CONTROL							
SIMPLE_AV_CONTROL_SET							
0 (don’t care)							
0 (key down)							
0 (don’t care)							
0 (don’t care)							
0 (AV Command MSB)							
0x27 (AV command Power)							
0xFF (match all bits)							
0xFF (match all bits)							
0x00 (don’t match)							
0x07 (match bits 0,1,2)							
0x00 (don’t match)							
0x00 (don’t match)							
0xFF (match all bits)							
0xFF (match all bits)							

7.5.5 Power Up on Timer Configuration Command

The Power Up on Timer Configuration Command is used to specify that the Z-Wave module should power up the host CPU system after a specified time has passed.

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_TIMER_CONFIGURATION_CMD							
Timer Resolution							
Timer (MSB)							
Timer (LSB)							

Timer Resolution (8bit):

PM_TIMER_SECONDS The timer resolution is in Seconds.

PM_TIMER_MINUTES The timer resolution is in minutes.

Timer Resolution define	Value
PM_TIMER_SECONDS	0x01
PM_TIMER_MINUTES	0x02

Timer (16bit):

The time that should elapse before the host CPU is set to the POWER_MODE_RUNNING again

7.5.6 External Power Up Configuration Command

The External Power Up Configuration Command is used to specify that a level change on an input pin should trigger a power up of the host CPU system.

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">FUNC_ID_POWER_MANAGEMENT</td></tr>
<tr><td colspan="8">PM_POWERUP_EXTERNAL_CONFIGURATION_CMD</td></tr>
<tr><td colspan="8">IO Pin</td></tr>
<tr><td colspan="8">Power Up Level</td></tr>
</table>

IO pin (8bit):

The IO pin field specifies the physical I/O pin that should be used for this signal. The full table of I/O pins can be found in section 7.5.2

Power Up Level (8bit):

The level the input pin should trigger a power up of the host CPU system.

0 – Low

1 – High

7.5.7 Power down Mode Configuration Command

The Power down Mode Configuration Command is used to request that the Z-Wave module sets a specific power down mode. If the PoweredUp pin is configured the PowerCtrl pins will not be changed before the PoweredUp pin goes NOT active.

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERDOWN_MODE_CONFIGURATION_CMD							
Number of Pins (max 4)							
IO Pin 1							
Level ..							
IO Pin x							
Level 1							
IO Pin ..							
Level x							

Number of Pins (8 bit):

The number of pins that is contained in the command. The max number of pins is 4

IO Pin x (8 bit):

The physical pin that should be changed when the Serial API powers down the host CPU system. A full list of physical pins can be found in section 7.5.2.

Level x (8 bit):

The level the output pin should have when the specified power mode is set.

0 – Low

1 – High

7.6 Ready Command

The Ready Command is used by the host to inform the Z-Wave module that it is ready to receive command on the UART.

7	6	5	4	3	2	1	0
FUNC_ID_READY							
[SerialLinkState]							

SerialLinkState (8 bit):

Set the Serial link state between HOST and the SerialAPI Z-Wave module.

SERIAL_LINK_DETACHED – The Serial link state should be DETACHED or SerialAPI stops sending data to HOST until either READY is transmitted again in connected state or any valid SerialAPI command is received from HOST.

SERIAL_LINK_CONNECTED – The Serial link state should be CONNECTED or SerialAPI sends data to HOST when needed.

The SerialAPI Z-Wave module starts up after reset in the Serial link state DETACHED.

SerialLinkState define	Value
SERIAL_LINK_DETACHED	0x00
SERIAL_LINK_CONNECTED	0x01

7.7 Softreset Command

The host CPU system can make a software reset of the Z-Wave module by using the Softreset Command.

7	6	5	4	3	2	1	0
FUNC_ID_SERIAL_API_SOFT_RESET							

Wait 1.5 seconds after reset in order to ensure that module is ready for communication again.

Note: USB modules will to a disconnect/connect then issuing this command. This means that the module may get a new address on the USB bus. This will make the old filehandle to the USB serial interface invalid.

7.8 Watchdog Commands

Some PC based applications cannot guarantee kicking the watchdog before timeout causing the watchdog to reset the Z-Wave ASIC unintentionally. The following Watchdog Commands are therefore available to avoid this:

- Start watchdog: Enable watchdog and ApplicationPoll kick watchdog
- Stop watchdog: Disable watchdog and stop kick watchdog in ApplicationPoll

Watchdog handling disabled when powered up and Sleep/FLiRS mode will temporary stop watchdog.

The host CPU system can start watchdog functionality by using the Serial API function FUNC_ID_ZW_WATCHDOG_START:

7	6	5	4	3	2	1	0
FUNC_ID_ZW_WATCHDOG_START							

The host CPU system can stop watchdog functionality by using the Serial API function FUNC_ID_ZW_WATCHDOG_STOP:

7	6	5	4	3	2	1	0
FUNC_ID_ZW_WATCHDOG_STOP							

7.9 Restrictions on functions using buffers

The Serial API is implemented with buffers for queuing requests and responses. This restricts how much data that can be transferred through MemoryGetBuffer() and MemoryPutBuffer() compared to using them directly from the Z-Wave API.

The PC application should not try to get or put buffers larger than approx. 80 bytes.

If an application requests too much data through MemoryGetBuffer() the buffer will be truncated and the application will not be notified.

If an application tries to store too much data with MemoryPutBuffer() the buffer will be truncated before the data is sent to the Z-Wave module, again without the application being notified.

APPENDIX A SERIAL API FILES

The Serial API embedded sample code is provided on the Z-Wave Developer's Kit. Be aware that altering the function ID's and frame formats in the Serial API embedded sample code can result in interoperability problems with the Z-Wave DLL supplied on the Developer's Kit as well as commercially available GUI applications. Regarding how to determine the current version of the Serial API protocol in the embedded sample code please refer to the API call **ZW_Version**.

The Product\SerialAPI directory contains sample source code for controller/slave applications on a Z-Wave module. The application uses also a number of utility functions described in [2], [3] or [4] depending on SDK used.

Appendix A.1 Makefiles

MK.BAT

Make bat file for building the sample application in question. To only build applications using EU frequency enter: **MK “FREQUENCY=EU”** in command prompt.

Makefile

This is the Makefile for the sample application in question defining the targets built. Refer to [2], [3] or [4] for additional details depending on SDK used.

Makefile.common_ZW0x0x_supported_functions

This makefile makes a text file showing the supported serial API functions for the given target.

Appendix A.2 Application

config_app.h

This header file contains defines for application version and defines for Manufacturer Specific Command Class.

conhandle.h / conhandle.c

Routines for handling Serial API protocol between PC and Z-Wave module.

eeprom.h / eeprom.c

NVM layout.

make-supported-functions-include.bat

Windows batch script for generating SerialAPI defines for supported functions based on what exists in library.

Prodtest_vars.c

Critical memory vars used for production test.

serialapi-supported-func-list.txt

Template file for generating SerialAPI defines for supported functions based on what exists in library. Enable/disable support of a given Serial API function in serialappl.h header file.

serialappl.h / serialappl.c

This module implements the handling of Serial API protocol. That is, parses the frames, calls the appropriate Z-Wave API library functions and returns results etc. to the PC. Enable/disable support of a given Serial API function in serialappl.h header file.

Supported.bat

Batch file called by Makefile.common_ZW0x0x_supported_function to obtain delayed environment variable expansion when using SET in DOS prompt.

REFERENCES

[1] IETF RFC 2119, Key words for use in RFCs to Indicate Requirement Levels, http://tools.ietf.org/pdf/rfc2119.pdf
[2] SD, INS12351, Instruction, Z-Wave ZW0201/ZW0301 Series Developer’s Kit v4.54.01 Contents.
[3] SD, INS12035, Instruction, Z-Wave 400 Series Developer’s Kit v6.02.00 Contents.
[4] SD, INS11442, Instruction, Z-Wave 400 Series Developer’s Kit v6.11.00 (JP) Contents.
[5] SD, INS11095, Instruction, Z-Wave ZW0201/ZW0301 Appl. Prg. Guide v4.55.00.
[6] SD, INS12034, Instruction, Z-Wave 400 Series Appl. Prg. Guide v6.02.00.
[7] SD, INS10682, Instruction, Z-Wave 400 Series Appl. Prg. Guide v6.11.00 (JP).
[8] SD, INS12308, Instruction, Z-Wave 500 Series Appl. Prg. Guide v6.51.02.

CONFIDENTIAL

INDEX

CONFIDENTIAL