NUR APPLICATION NOTE 001 (AN001)

GETTING STARTED: BASIC COMMANDS EXPLAINED

SCOPE

This application note extends the NUR protocol documentation by showing simple packets that implement the following commands and their responses:

Command	Value	Description
Ping	0x01	Check the presence of the module; "alive" command.
Get mode	0x04	Returns the current state of the module i.e. whether it is running application or bootloader.
Reader information	0x09	Returns the basic information about the reader
Get device capabilities	0x0B	Return information about the device capabilities.

GENERAL COMMAND STRUCTURE

The protocol documentation shows the detailed information about the parts of the NUR protocol. As a recap here are the three main parts of a packet and response:

Part	Value (HEX)	Description
Header	6 bytes	Start byte, payload+ CRC length (16 bits), flags (16-bit) and header checksum.
Payload	n bytes	Command and possible parameters *OR* Response, status byte + response contents or error information.
CRC-16	16-bits in little-endian format.	Calculation at the end of the document.

C-STRUCTURES

The C-structures in this document are in little-endian format and packed. In Visual studio the structure are packed like

```
#pragma pack(push, 1)

/* Structures/unions here… */

#pragma pack(pop)
```

With GCC (__GNUC__ defined) packing can be done for example

```
#define __packed __attribute__ ((packed))

/* … */

struct __packed CMD_STRUCTURE …
```

BASIC COMMANDS: PING (01)

Ping is used as a communication test. The ping response consists of two bytes consisting of ASCII character 'O' and 'K'.

PING PACKET IN HEX

```
A5 03 00 00 00 59 01 D1 F1
```

PING PACKET IN C

```c
const unsigned char pingCmd[] =
{ 0xA5, 0x03, 0x00, 0x00, 0x00, 0x59, 0x01, 0xD1, 0xF1 };
```

PING COMMAND CONTENTS

Field	Value	Description	
Header	A50300000059 (6 bytes)	Header consisting of:	
		A5	
		0300 = 0x0003	Payload + CRC length
		0x0000	Command flags
		0x59	Header check sum
Command	0x01	Ping.	
Payload CRC	0xD1 0xF1	Little endian; value is **0xF1D1**.	

Header checksum = 0xA5 XOR 0x03 XOR 0x00 XOR 0x00 XOR 0x00 XOR 0x00 = 0x59.

THE PING COMMAND'S RESPONSE

The ping when the module responds correctly states "OK".

PING RESPONSE PACKET

```
A5 06 00 00 00 5C 01 00 4F 4B 29 16
```

PING RESPONSE'S C-STRUCTURE EXAMPLE

```
struct PACKED NUR_PINGRESP
{
  unsigned char start;  /* Start byte */
  unsigned short payLen; /* Payload + CRC length */
  unsigned short flags; /* command/response flags */
  unsigned char echo; /* Command echo */
  unsigned char status; /* Command execution status */
  char ok[2]; /* { 'O', 'K' } */
  unsigned short crc; /* Payload CRC-16 */
};
```

PING RESPONSE CONTENTS

Field	Value	Description	
Header	A5060000005C (6 bytes)	Header consisting of:	
		A5	
		0600 = 0x0006	Payload + CRC length
		0x0000	Command flags
		0x5C	Header check sum
Command echo	0x01	Ping.	
Command execution status	0x00	No error (0).	
Response payload	0x4F 0x4B	ASCII characters 'O' (0x4F) and 'K' (0x4B).	
Payload CRC	0x29 0x16	Little endian; value is **0x1629.**	

GET MODE COMMAND (04)

The get mode command indicates whether the module is currently running application or bootloader.

GET MODE PACKET IN HEXADECIMAL

```
A5 03 00 00 00 59 04 74 A1
```

GET MODE IN C

```
const unsigned char getmodecmd[] =
{ 0xA5, 0x03, 0x00, 0x00, 0x00, 0x59, 0x04, 0x74, 0xA1 };
```

GET MODE COMMAND CONTENTS

Field	Value	Description	
Header	A50300000059 (6 bytes)	Header consisting of:	
		A5	
		0300 = 0x0003	Payload + CRC length
		0x0000	Command flags
		0x59	Header check sum
Command	0x04	Get mode.	
Payload CRC	0x74 0xA1	Little endian; value is **0xA174**.	

GET MODE RESPONSE

Get mode responds with either ASCII ‘A’ for application mode and ‘B’ for bootloader.

GET MODE RESPONSE PACKET

```
A5 05 00 00 00 5F 04 00 41 B9 48
```

GET MODE RESPONSE’S C-STRUCTURE EXAMPLE

```
struct PACKED NUR_PINGRESP
{
  unsigned char start;  /* Start byte */
  unsigned short payLen; /* Payload + CRC length */
  unsigned short flags; /* command/response flags */
  unsigned char echo; /* Command echo */
  unsigned char status; /* Command execution status */
  char mode; /* ‘A’ or ‘B’ */
  unsigned short crc; /* Payload CRC-16 */
};
```

GET MODE RESPONSE CONTENTS

Field	Value	Description	
Header	A5050000005F (6 bytes)	Header consisting of:	
		A5	
		0500 = 0x0005	Payload + CRC length
		0x0000	Command flags
		0x5F	Header check sum
Command echo	0x04	Get mode.	
Command execution status	0x00	No error (0).	
Response payload	0x41	ASCII ‘A’ states that module runs currently application.	
Payload CRC	0xB9 0x48	Little endian; value is **0x48B9**.	

GET READER INFORMATION (09)

The get reader information command returns basic information about the reader. These include:

- Reader information version magic
- Serial number
- Alternate serial number
- Reader name
- FCC identifier
- HW version
- SW version major, minor and build (e.g. 4.2-A)
- Number of GPIO pins (NUR module/evaluation board)
- Number of sensors (Sampo USB and Ethernet readers)
- Number of supported regions
- Number of antennas

GET READER INFORMATION COMMAND PACKET

```
A5 03 00 00 00 59 09 D9 70
```

GET READER INFORMATION IN C

```
const unsigned char riCmd[] =
{ 0xA5, 0x03, 0x00, 0x00, 0x00, 0x59, 0x09, 0xD9, 0x70 };
```

GET READER INFORMATION COMMAND CONTENTS

Field	Value	Description	
Header	A50300000059 (6 bytes)	Header consisting of:	
		A5	
		0300 = 0x0003	Payload + CRC length
		0x0000	Command flags
		0x59	Header check sum
Command	0x09	Get reader information.	
Payload CRC	0xD9 0x70	Little endian; value is **0x70D9**.	

GET READER INFORMATION RESPONSE

The response contains the reader information fields as described in the command’s general description.

AN EXAMPLE OF READER INFORMATION RESPONSE PACKET

```
A5 58 00 00 00 02 09 00 01 49 44 52 09 53 30 30 30 30 30 58 59 5A 0C 31
31 32 32 33 33 34 34 35 35 36 36 04 53 54 49 58 25 46 43 43 3A 20 53 43
43 4E 55 52 30 35 57 4C 32 20 2F 20 49 43 3A 20 35 31 33 37 41 2D 4E 55
52 30 35 57 4C 32 06 4E 4F 54 53 45 54 04 07 48 05 00 10 01 AC E9
```

GET READER INFORMATION RESPONSE CONTENTS

Field	Value	Description	
Header	A55800000002 (6 bytes HEX)	Header consisting of:	
		A5	
		58 00 = 0x0058	Payload + CRC length (88 bytes)
		0x0000	Command flags
		0x02	Header check sum
Command echo	0x09	Get reader information.	
Command execution status	0x00	No error (0).	
Version 1 magic	01494452	0x52444901 as little endian.	
Serial length	0x09	9 characters to follow.	
Serial number	53 30 30 30 30 30 58 59 5A (HEX)	"S00000XYZ"	
Alt serial length	0x0C	12 characters to follow.	
Alt serial	31 31 32 32 33 33 34 34 35 35 36 36 (HEX)	"112233445566"	
Name length	04	4 characters to follow.	
Name	0x53 0x54 0x49 0x58	"STIX" (Stix mini reader)	
FCC ID length	0x25	37 characters to follow.	
FCC ID string	46 43 43 3A 20 53 43 43 4E 55 52 30 35 57 4C 32 20 2F 20 49 43 3A 20 35 31 33 37 41 2D 4E 55 52 30 35 57 4C 32 (HEX)	"FCC: SCCNUR05WL2 / IC: 5137A-NUR05WL2"	
HW version length	0x06	6 characters to follow.	
SW version	0x04 0x07 0x48	= 4.7-H	
Nr of GPIOs	0x05	5 (possible GPIOs in this case; STIX does not have the GPIOs physically connected).	
Nr of sensors	0	No sensors available (sensors can be configured with Sampo table readers).	
Nr of regions	0x10	16 supported regions	
Nr of antennas	0x01	One antenna present / available.	
CRC-16	0xAC E9	**0xE9AC** in little-endian format.	

GET DEVICE CAPABILITIES

Getting the device capabilities returns the following information:

Field	Description
Flag set 1	32 bits indicating presence of various features.
Flag set 2	32 bits indicating presence of various features (currently unused).
Max dBm	Maximum TX level in dBm. NUR05W, NUR05WL and NUR05WL2 = TX_{max} = 27dBm NUR10W = TX_{max} = 30dBm
Tx level step dBm	TX adjustment step in dBm.
Max mW	Maximum TX level in mW. NUR05W, NUR05WL and NUR05WL2 = TX_{max} = 500mW NUR10W = TX_{max} = 1W
TX levels	Number of TX adjustment steps.
Tag buffer size	Number of tags that can be stored. Tags are assumed to have 96-bit EPC.
Max antennas	Maximum number of antennas possible with current configuration.
Max GPIO	Maximum number of GPIO pins possible with current configuration.
Chip version	The RFID chip version.
Module type	Module type
Configuration	Module's internal configuration flags.

GET DEVICE CAPABILITIES PACKET

```
A5 03 00 00 00 59 0B 9B 50
```

GET DEVICE CAPABILITIES PACKET IN C

```
const unsigned char dcCmd[] =
{ 0xA5, 0x03, 0x00, 0x00, 0x00, 0x59, 0x0B, 0x9B, 0x50 };
```

GET DEVICE CAPABILITES COMMAND CONTENTS

<table>
<tr><th>Field</th><th>Value</th><th colspan="2">Description</th></tr>
<tr><td rowspan="5">Header</td><td rowspan="5">A50300000059
(6 bytes)</td><td colspan="2">Header consisting of:</td></tr>
<tr><td>A5</td><td></td></tr>
<tr><td>0300 = 0x0003</td><td>Payload + CRC length</td></tr>
<tr><td>0x0000</td><td>Command flags</td></tr>
<tr><td>0x59</td><td>Header check sum</td></tr>
<tr><td>Command</td><td>0x0B</td><td colspan="2">Get device capabilities.</td></tr>
<tr><td>Payload CRC</td><td>0x9B 0x50</td><td colspan="2">Little endian; value is <b>0x509B.</b></td></tr>
</table>

AN EXAMPLE OF DEVICE CAPABILITIES RESPONSE

The device capabilities structure currently has a fixed size of 128. Response bytes for example:

```
A5 84 00 00 00 DE 0B 00 26 00 00 00 CF 81 0F 00
00 00 00 00 1B 00 00 00 01 00 00 00 F4 01 14 00
B8 02 01 00 00 00 02 00 03 00 04 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 19 CC
```

GET READER CAPABILITIES C-STRUCTURE

```
#define SZ_DEVCAPS  128
#define SZ_DC_RES \
(SZ_NUR_DEVCAPS-(6*sizeof(int)-7*sizeof(unsigned short)))

struct __packed NUR_DEVICECAPS
{
  unsigned int dwSize; /* Size before reserved field. */
  unsigned int flagSet1;
  unsigned int flagSet2;

  int maxTxdBm;
  int txAttnStep;
  unsigned short maxTxmW;
  unsigned short txSteps;

  unsigned short szTagBuffer;
  unsigned short curCfgMaxAnt;
  unsigned short curCfgMaxGPIO;
  unsigned short chipVersion;
  unsigned short moduleType;
  unsigned int moduleConfigFlags;

  unsigned char res[SZ_DC_RES];
};
```

GET READER CAPABILITIES RESPONSE CONTENTS

Field	Value	Description
Header	A584000000DE (6 bytes HEX)	Header consisting of: A5 84 00 = 0x0084 – Payload + CRC length (132 bytes) 0x0000 – Command flags 0xDE – Header check sum
Command echo	0x0B	Get read capabilities.
Status	0x00	No error (0).
Size	0x26 0x00 0x00 0x00	Size before reserved field = 0x26 = 38 bytes.
Flag set 1	0xCF 0x81 0x0F 0x00	Flag set 1. See flag set 1 0x000F81CF contents.
Flag set 2	0x00 0x00 0x00 0x00	Flag set 2. Currently unused.
Max dBm	0x1B 0x00 0x00 0x00	0x0000001B = 27; maximum level in dBm.
TX level step	0x01 0x00 0x00 0x00	Step is 0x00000001 = 1 dBm.
Max mW	0xF4 0x01	Maximum TX level in mW = 0x01F4 = 500 mW.
Tx level steps	0x14 0x00	Number of steps is 0x0014 = 20.
Tag buffer size	0xB8 0x02	Tag buffer size is 0x02B8 = 696 (tags having 96-bit EPC).
Max antennas	0x01 0x00	One antenna available.
Max GPIOs	0x00 0x00	No GPIOs available.
RFID chip version	0x02 0x00	1 = AS3992 2 = AS3993
Module type	0x03 0x00	
Configuration	0x04 0x00 0x00 0x00	
Reserved	All zeros	90 bytes.
CRC-16	0x19 0xCC	Little-endian; value is **0xCC19**.

DEVICE CAPABILITIES: FLAG SET 1 0X000F81CF CONTENTS

Flag bit	Mask value	Description (yes = present in 0x000F81Cf)
0	0x00000001	FM0 RX decoding: **yes**
1	0x00000002	M-2 RX decoding: **yes**
2	0x00000004	M-4 RX decoding: **yes**
3	0x00000008	M-8 RX decoding: **yes**
4	0x00000010	Link frequency (LF) of 40kHz: no
5	0x00000020	Link frequency of 80kHz: no
6	0x00000040	Link frequency of 160kHz: **yes**
7	0x00000080	Link frequency of 256kHz: **yes**
8	0x00000100	Link frequency of 320kHz: **yes**
9	0x00000200	Link frequency of 640kHz: no
10	0x00000400	Reserved LF 1.
11	0x00000800	Reserved LF 2.
12	0x00001000	Beeper present: no
13	0x00002000	Light sensor present: no
14	0x00004000	Tap sensor present: no
15	0x00008000	Can tune antenna: **yes**
16	0x00010000	Can scan channels: **yes**
17	0x00020000	Can do inventory + read: **yes**
18	0x00040000	Has per antenna power setting: **yes**
19	0x00080000	Has antenna -1…1 power offset capability: **yes**
20	0x00100000	Supports grid antenna: no
21	0x00200000	Tags can be fetched one by one: no
22…31	Mask: 0xFFC00000	Currently unused

CRC-16 CALCULATION

This is the C-implementation of the CRC-16:

```
#define CRC16_START       0xFFFF
#define CRC16_POLYNOMIAL  0x1021 /* CCITT */

static unsigned short crc16table[256];
static int crc16Init = 0;

static void CRC16Init()
{
  int i, j;
  unsigned int c;
  for (i = 0; i < 256; i++) {
    c = i << 8;
    for ( j = 0; j < 8; j++ ) {
      c = (c & 0x8000) ? CRC16_POLYNOMIAL ^ (c << 1) : (c << 1);
    }
    crc16table[i] = (unsigned short)c;
  }
}

unsigned short CRC16(unsigned short crc, const unsigned char *buf,
unsigned int len)
{
  if (!crc16Init) {
    crc16Init = TRUE;
    CRC16Init();
  }
  while (len--) {
    crc = (crc << 8) ^ crc16table[(crc >> 8) ^ *buf++];
  }
  return crc;
}
```