NUR APPLICATION NOTE 6 (NUR AN006)

GETTING STARTED: WRITING TAG'S MEMORY

Nordic ID Oy | Myllyojankatu 2 A | FI-24100 Salo | Finland
Office +358 2 727 7700 | Fax + 358 2 727 7720 | info@nordicid.com

SCOPE

This document shows several cases of tag memory writing. This includes non-selected (no tag singulation) write and singulation based writing both with and without password.

Scenario/part	Description
General write packet contents	Write commands' various blocks: common, singulation (selection) and write.
Write command response	Description of write command response
Write command errors	Description of typical errors in write.
User memory write with EPC selection	Tag is selected by its EPC contents and 4 words (8 bytes) of the user memory is written.
Writing kill password using access password and EPC selection	Tag is selected by its EPC contents and 2 words (4 bytes) of the password (reserved) memory is written (the kill part of the password memory) using EPC tag selection.
Writing EPC contents to a tag without tag selection	Tag's EPC contents are replaced with a new EPC. Assumes that only tag is in the field thus no selection is made.
Error examples	Parameter error examples.

GENERAL WRITE PACKET CONTENTS

The write command consists of write command code that is followed by:

- Control flags and password (block is mandatory)
- Singulation (tag selection) block (optional, presence indicated by the control flags)
- Write instruction block: bank, address, word count and data to write (block is mandatory)

WRITE COMMAND BLOCK'S CONTENTS

The offsets are presented from each block's base i.e. starting from 0. The common block starts right after the command code which is **`0x34 (52)`**.

COMMON BLOCK

Common block is always present for any read or write (or alike command such as lock or kill).

COMMON BLOCK CONTENTS

Byte(s)	Is	Description	
0	Flags	Bit flags controlling the write operation:	
		Bit	**Is**
		0 (mask 0x01)	If '1', the operation uses tag accessing by the following password.
		1 (mask 0x02)	If '1', this part is then followed by a singulation block defining how the tag is selected.
		2 (mask 0x04)	If '1', then any operation that requires addressing (namely read/write) within a memory bank uses 64-bit addressing; '0' (default) means 32-bit addressing.
		3 (mask 0x08)	If '1', then the singulation uses 64-bit addressing; '0' (default) means 32-bit addressing.
		4…7	Not used; set to '0'.
1…4	Password.	32-bit unsigned: password in little-endian format. Always present whether used or not.	

COMMON BLOCK C-STRUCTURE EXAMPLE

NOTE: this example here also includes the command field. This structure is common for all read and write (and alike) commands.

```
struct __packed COMMON_RW_BLOCK
{
  /* Read/write/lock/kill command value (read=0x33) */
  unsigned char rwlkCmd;
  /* Control flags. */
  unsigned char flags;
  /* Password value fo secured tag accessing if used. */
  unsigned int password;
};
```

SINGULATION (TAG SELECTION) BLOCK

The application note NUR_AN010: tag selection explains the bit level operation of the tag singulation. The leftmost column shows the byte offsets when using 32-bit addressing and the following shows the byte offsets when using 64-bit addressing.

SINGULATION BLOCK CONTENTS

Byte(s) (32-bit addr)	Byte(s) (64-bit addr)	Is	Description
0	0	Bytes to follow	Number of bytes to follow = this block's size in bytes - 1.
1	1	Bank	Selection bank: 1 (EPC), 2 (TID) or 3 (user memory).
2…5	2…9	Mask address	The *bit address* within the selection bank where the
6…7	10..11	Mask length	The *bit length* of the selection mask.
8…n	12…n	Mask data.	The bit mask data used for selection. Length must be the bit length divided by 8; if length *mod 8 is not zero* then 1 is added.

SELECTION BLOCK'S C-STRUCTURE EXAMPLE

```
/* Block that defines tag selection parameters: 32-bit */
struct __packed NUR_TAGSELBLOCK32
{
  uint8_t size;       /* Number of bytes to follow. */
  uint8_t bank;       /* Selection bank. */
  uint32_t bitAddr;   /* 32-bit address of the selection mask. */
  uint16_t bitLen;    /* Bit length of the selection mask. */
  uint8_t bitBuf[1]; /* Variable length bit buffer data. */
};

/* Block that defines tag selection parameters: 64-bit */
struct __packed NUR_TAGSELBLOCK64
{
  uint8_t size;       /* Number of bytes to follow. */
  uint8_t bank;       /* Selection bank. */
  uint64_t bitAddr;   /* 64-bit address of the selection mask. */
  uint16_t bitLen;    /* Bit length of the selection mask. */
  uint8_t bitBuf[1]; /* Variable length bit buffer data. */
};
```

THE WRITE BLOCK

WRITE BLOCK'S CONTENTS

Byte(s) (32-bit addr)	Byte(s) (64-bit addr)	Is	Description
0	0	Bytes to follow	Number of bytes to follow = this block's size in bytes (including data length) - 1.
1	1	Bank	The bank to write to: 0 (password/reserved), 1 (EPC), 2 (TID) or 3 (user memory).
2…5	2…9	Word address	The word address to write to; 16-bit aligned as per Gen2 specification.
6	10	Word count	Number of 16-bit words to write.
7…	11…	Bytes to write	The number of bytes must be aligned by 2 (2 bytes = 1 16-bit word). Data is written "as is" thus any endian considerations must be made by the host application.

WRITE BLOCK'S C-STRUCTURE EXAMPLE

```
/* Block that defines tag write parameters: 32-bit addressing */
struct __packed NUR_TAGWRITEBLOCK32
{
  uint8_t size;       /* Number of bytes to follow. */
  uint8_t bank;       /* Bank to write. */
  uint32_t wordAddr;  /* 32-bit word address to write to. */
  uint16_t wordCount;   /* Word count. */
  /* Cast struct from byte buffer in order to handle.*/
  uint8_t wrData[1];
};

/* Block that defines tag write parameters: 64-bit addressing */
struct __packed NUR_TAGWRITEBLOCK64
{
  uint8_t size;       /* Number of bytes to follow. */
  uint8_t bank;       /* Bank to write. */
  uint64_t wordAddr;  /* 64-bit word address to write to. */
  uint16_t wordCount;   /* Word count. */
  /* Cast struct from byte buffer in order to handle.*/
  uint8_t wrData[1];
};
```

THE WRITE COMMAND RESPONSE

A successful write response consists of command byte echo followed by

- status byte
- indication of successfully written 16-bit words

A SUCCESSFUL WRITE RESPONSE CONTENTS

Byte(s)	Is	Description
0	Command echo	Write command echo: 0x34.
1	Status	0 = OK
2	Word count	Number of words successfully written.

A SUCCESSFUL WRITE RESPONSE C-STRUCTURE EXAMPLE

The example also includes command echo and status.

```
struct __packed NUR_TAGWRITERESP
{
  uint8_t cmd;      /* Command echo. */
  uint8_t status;   /* Status (0=OK). */
  uint8_t wordCount; /*Number of words written. */
};
```

Such a response structure is assumed to be pointed via for example byte pointer in order to access the members correctly.

THE WRITE COMMAND ERROR RESPONSE

When the actual write execution fails (no parameter errors) the reader sends back the error code received from the tag if the error happened at the tag's side. This error code is followed by the number of successfully written words if any. The module's error code is then **0x42**: tag error. If a write is partially successful then the error code is **G2_PARTIAL_WRITE, 0x42 (65)**.

WRITE ERROR: TAG ERROR PACKET CONTENTS

Byte(s)	Is	Description
0	0x34	Write command echo: 0x34.
1	0x42	Tag error, code follows.
2	<error>	If present, can be either error code as specified by the Gen2 protocol specification or module's error flag set.
3	<ok_words>	Number of successfully written words; the error may be "partial write error" (0x41, 65) indicating that some of the words were written.
4…5	CRC	The response CRC-16.

TAG ERROR EXAMPLE C-STRUCTURE

```
struct __packed WRITETAG_ERRORRESP
{
  uint8_t cmd;    /* Command echo.*/
  uint8_t status; /* Module error code. */
  uint8_t error; /* Tag's error code. */
  uint8_t wordCount; /* Number of successfully written words.*/
};
```

PARTIAL WRITE ERROR EXAMPLE

Example shows a response after password memory's word address 3 is written with HEX data

`AA AA BB BB`

thus causing the part `BB BB` being out of the memory's last address (`BB BB` to words address 4 which is non-existent).

PARTIAL WRITE RESPONSE PACKET

`A5 05 00 00 00 5F 34 41 01 25 FB`

PARTIAL WRITE RESPONSE PACKET CONTENTS

Byte(s)	Value(s) HEX	Description	
0…5	A5050000005F (6 bytes)	Header consisting of:	
		A5	
		0500 = 0x0005	Payload + CRC length
		0x0000	Response flags
		0x5F	Header check sum
6	34	Command echo	
7	41	Status: 41 = partial write error	
8	01	Number of successfully written words: 1 (0xAA 0xAA)	
9…10	25 FB	Unsigned 16-bit, little-endian: packet CRC-16 = **0xFB25**.	

PARTIAL WRITE ERROR C-STRUCTURE EXAMPLE

```
struct __packed WRITETAG_PARTIALERROR
{
  uint8_t cmd;    /* Command echo.*/
  uint8_t status; /* Module error code. */
  uint8_t wordCount; /* Number of successfully written words.*/
};
```

WRITE EXAMPLE: USER MEMORY, EPC SELECTION

In the example the tag user memory is written with data (4 words, 8 bytes)

```
01 01 02 02 03 03 04 04
```

WRITE PACKET

```
A5 2B 00 00 00 71 34 02 00 00 00 00 13 01 20 00 00 00 60 00 20 00 02 01
01 00 00 00 00 00 06 DB 0E 03 00 00 00 00 04 01 01 02 02 03 03 04 04 CD
1E
```

WRITE PACKET CONTENTS

Byte(s)	Value(s) HEX	Description	
0…5	A52B00000071 (6 bytes)	Header consisting of:	
		A5	
		2B00 = 0x002B (43)	Payload + CRC length
		0x0000	Command flags
		0x71	Header check sum
Common block			
6	34	Write tag.	
7	02	Byte, control flags: bit 1 (mask 0x02) is set: the common block is followed by singulation block	
8…11	00 00 00 00	Unsigned 32-bit, little-endian: password, not used, set to 0.	
Singulation block			
12	13	Bytes to follow (19): size of singulation block – 1.	
13	01	Bank where the selection mask is applied to: 1 = EPC	
14…17	20 00 00 00	Unsigned 32-bit, little-endian: mask bit address, 0x00000020 (32): the EPC's bit start address.	
18..19	60 00	Unsigned 16-bit, little-endian: selection mask's bit length: 0x0060 (96)	
20…31	20 00 02 01 01 00 00 00 00 00 06 DB	Selection mask, 12 bytes (96 bits): the tag's EPC.	
Write block			
32	0E	Bytes to follow (14): size of write block – 1.	
33	03	Bank to write: 3 = user memory	
33…36	00 00 00 00	Unsigned 32-bit, little-endian: address of the first word to write.	
37	04	Byte: word count (Number of 16-bit words to write).	
38…45	01 01 02 02 03 03 04 04	Data to write: 4 words / 8 bytes.	
46…47	CD 1E	Unsigned 16-bit, little-endian: packet CRC-16 = **0x1ECD**.	

WRITE EXAMPLE: USER MEMORY RESPONSE

RESPONSE PACKET

```
A5 05 00 00 00 5F 34 00 04 7D 95
```

RESPONSE CONTENTS

Byte(s)	Value(s) HEX	Description	
0…5	A5050000005F (6 bytes)	Header consisting of:	
		A5	
		0500 = 0x0005	Payload + CRC length
		0x0000	Response flags
		0x5F	Header check sum
6	34	Command echo	
7	00	Status: 0 = OK	
8	04	Number of words written: 4.	
9…10	7D 95	Unsigned 16-bit, little-endian: packet CRC-16 = **0x957D**.	

WRITE EXAMPLE: WRITE KILL PASSWORD USING EPC SELECTION AND ACCESS PASSWORD

The example's tag is singulated by its EPC: `20 00 02 01 01 00 00 00 00 00 06 DB`

The password memory content is: **`00 00 00 00 00 00 AA BB`**

The password memory is locked for reading and writing thus the write command uses access password `0x`**`0000AABB`**. The new kill password will be **`0x0F0F0F0F`**.

WRITE PACKET

```
A5 27 00 00 00 7D 34 03 BB AA 00 00 13 01 20 00 00 00 60 00 20 00 02 01
01 00 00 00 00 00 06 DB 0A 00 00 00 00 00 02 0F 0F 0F 0F 6E 9C
```

WRITE PACKET CONTENTS

Byte(s)	Value(s) HEX	Description	
0…5	A5270000007D (6 bytes)	Header consisting of:	
		A5	
		2700 = 0x0027 (39)	Payload + CRC length
		0x0000	Command flags
		0x7D	Header check sum
Common block			
6	34	Write tag.	
7	02	Byte, control flags: bit 0 is set (use password that follows) and bit 1 (mask 0x02) is set: the common block is followed by singulation block	
8…11	BB AA 00 00	Unsigned 32-bit, little-endian: password: 0x0000AABB	
Singulation block			
12	13	Bytes to follow (19): size of singulation block – 1.	
13	01	Bank where the selection mask is applied to: 1 = EPC	
14…17	20 00 00 00	Unsigned 32-bit, little-endian: mask bit address = 0x00000020 (32)	
18..19	60 00	Unsigned 16-bit, little-endian: selection mask's bit length: 0x0060 (96)	
20…31	20 00 02 01 01 00 00 00 00 00 06 DB	Selection mask, 12 bytes (96 bits): the EPC contents.	
Write block			
32	0A	Bytes to follow: size of write block – 1.	
33	00	Bank to write: 0 = password/reserved memory	
34…37	00 00 00 00	Unsigned 32-bit, little-endian: word address of first word to write.	
34	02	Byte: word count (Number of 16-bit words to write).	
35…38	0F 0F 0F 0F	4 bytes/2 words of kill password to write.	
39…40	6E 9C	Unsigned 16-bit, little-endian: packet CRC-16 = **0x9C6E**.	

WRITE EXAMPLE: KILL PASSWORD WRITE RESPONSE

RESPONSE PACKET

```
A5 05 00 00 00 5F 34 00 02 BB F5
```

RESPONSE CONTENTS

Byte(s)	Value(s) HEX	Description	
0…5	A5050000005F (6 bytes)	Header consisting of:	
		A5	
		0500 = 0x0005	Payload + CRC length
		0x0000	Command flags
		0x5F	Header check sum
6	34	Command echo	
7	00	Status: 0 = OK	
8	02	Number of words written: 2 (4 bytes).	
9…10	BB F5	Unsigned 16-bit, little-endian: packet CRC-16 = **0xF5BB**.	

WRITE EXAMPLE: WRITE A NEW EPC TO A TAG WITHOUT SELECTION

In this example a tag's EPC is re-written to:

```
23 45 02 01 01 00 00 00 01 02 03 04
```

There is no singulation i.e. tag selection involved; this effectively means that there is only one tag in the RF field for this command to execute successfully.

WRITE PACKET

```
A5 1B 00 00 00 41 34 00 00 00 00 00 12 01 02 00 00 00 06 23
45 02 01 01 00 00 00 01 02 03 04 3D B7
```

WRITE PACKET CONTENTS

Byte(s)	Value(s) HEX	Description	
0…5	A51B00000041 (6 bytes)	Header consisting of:	
		A5	
		1B00 = 0x001B (27)	Payload + CRC length
		0x0000	Command flags
		0x41	Header check sum
Common block			
6	34	Write tag.	
7	00	Byte, control flags, no bits set: not using singulation nor password, the write addressing is 32-bit.	
8…11	00 0 00 00	Unsigned 32-bit, little-endian: password not used, set to 0.	
Write block			
12	12	Bytes to follow (18): size of write block – 1.	
13	01	Bank to write: 1 = EPC memory	
14…17	02 00 00 00	Unsigned 32-bit, little-endian: word address of first word to write.	
18	06	Byte: word count (number of 16-bit words to write): 6 = 12 bytes, the EPC's length.	
19…30	23 45 02 01 01 00 00 00 01 02 03 04	12 bytes (6 words) of EPC data.	
31…32	3D B7	Unsigned 16-bit, little-endian: packet CRC-16 = **0xB73D**.	

When successful the response is like in the examples here and here.

ERROR EXAMPLES

DATA LENGTH MISMATCH

This error packet is received when the data length does not match the given number or words.

Error response packet: `A5 05 00 00 00 5F 34 05 06 CA 4A`

Error response contents:

Byte(s)	Value(s) HEX	Description	
0…5	A5050000005F (6 bytes)	Header consisting of:	
		A5	
		0500 = 0x0005	Payload + CRC length
		0x0000	Response flags
		0x5F	Header check sum
6	34	Command echo	
7	00	Status: 05 = invalid parameter error	
8	06	Specific error code 0x06: error in the write block (in this case caused by data length mismatch)	
9…10	CA 4A	Unsigned 16-bit, little-endian: packet CRC-16 = **0x4ACA**.	

INVALID SELECTION BANK

This error packet is received when the bank used for tag singulation is out of range 1…3.

Error response packet: `A5 05 00 00 00 5F 34 05 03 6F 1A`

Error response contents:

Byte(s)	Value(s) HEX	Description	
0…5	A5050000005F (6 bytes)	Header consisting of:	
		A5	
		0500 = 0x0005	Payload + CRC length
		0x0000	Response flags
		0x5F	Header check sum
6	34	Command echo	
7	00	Status: 05 = invalid parameter error	
8	03	Specific error code 0x03: error in the singulation block (in this case caused by invalid bank number)	
9…10	6F 1A	Unsigned 16-bit, little-endian: packet CRC-16 = **0x1A6F**.	