Understanding the zForce ASN.1 Serialization Protocol

All communication with the Touch Sensor Module is serialized with Neonodes ASN.1 serialization protocol, and when implementing your own solution for the sensor module it is vital to know how to encode and decode it. This article explains the basics of ASN.1 DER/BER encoding and then walks you through the encoding of a DeviceConfiguration Message from the zForce ASN.1 Protocol.

Downloading the definition file

Download the zForce® PDU definition file here.

Type, Length, Value

All ASN.1 messages are structured by type, length and value:

- Type describes the type of the whole message or a subpart of a message. Type includes information on class and tag number.
- Length defines how many bytes there are in the message or in other words, the size of the value.
- Value is the actual value you are sending to the device, it is either a primitive value or a list. The primitive value types that are used in this protocol are:
    - Integer
    - Boolean
    - Octet String
    - Bit String

The Type Byte(s)

Type bytes can be constructed or primitive. A constructed type is a list, and in this protocol, all lists are defined by a sequence, and will hereafter be referred to as sequences.
Class tags and tag numbers are used to encode a type byte. The following ASN.1 class tags are used:

Class	Bit 8	Bit 7	Description
Universal	0	0	Not used in our protocol
Application	0	1	Shared
Context-specific	1	0	Local. For example specific to an Application
Private	1	1	Only used to describe the type of the whole message, Request/Response/Notification

The encoding for tag numbers up to and including 30 (higher code numbers are encoded differently, but those are not used in our protocol):

Bit position		8th	7th	6th	5th	4th	3rd	2nd	1st
Specific bit value	**Binary representation**	1000 0000	0100 0000	0010 0000	0001 0000	0000 1000	0000 0100	0000 0010	0000 0001
	Decimal representation	128	64	32	16	8	4	2	1
	Hexadecimal representation	0x80	0x40	0x20	0x10	0x08	0x04	0x02	0x01
Description		Reserved for class tag	Reserved for class tag	Reserved for primitive/sequence	Reserved for tag number	Reserved for tag number	Reserved for tag number	Reserved for tag number	Reserved for tag number

In short this means that the 8th and 7th bits are reserved to specify the class, and the 6th bit is reserved to show if it is a sequence. Bits 5 to 1 are used to specify the tag number.

Example: The type byte for the command deviceConfiguration

The command as seen in the protocol:

```
deviceConfiguration [APPLICATION 19] Sequence
```

This tells us that the deviceConfiguration is a sequence with the class tag Application and 19 as tag number. The deviceConfiguration type byte looks like this when it is represented as an octet:

Bit position	8th	7th	6th	5th	4th	3rd	2nd	1st
Bit	0	1	1	1	0	0	1	1
Hexadecimal Value		0x40	0x20	0x10			0x02	0x01
Description	Class tag	Class tag	Sequence tag	Tag number value	Tag number value	Tag number value	Tag number value	Tag number value

All of the hexadecimal numbers are then added together to get the type byte 0x40 + 0x20 + 0x10 + 0x02 + 0x01 = 0x73.

The Length Byte(s)

The length byte defines the number of bytes in the value byte(s) that follows it, and if the number is 127 or below, the length byte is only one byte. If the number is 128 or higher, the length byte splits into two pieces: The first piece, is the first byte that describes the amount of length bytes that follows it, and the second piece is the unsigned integer that holds the whole length value.

Example: The length byte for a value that is 50 bytes long

The decimal value 50 translates to 0x32 in hexadecimal representation and 0011 0010 in binary representation

Bit position	8th	7th	6th	5th	4th	3rd	2nd	1st
Bit	0	0	1	1	0	0	1	0
Hexadecimal Value			0x20	0x10			0x02	
Description	If this bit is set the length of the value is 128 bytes or longer	Value	Value	Value	Value	Value	Value	Value

Example: The length bytes for a value that is 300 bytes long

The length bytes for a value that is 300 bytes long consists of three bytes. The first byte indicates that the following two bytes ((0x01) 0000 0001 (0x2C) 0010 1100) should be added together. The first byte is described in the following table:

Bit position	8th	7th	6th	5th	4th	3rd	2nd	1st
Bit	1	0	0	0	0	0	1	0
Hexadecimal Value	0x80						0x02	
Description	If this bit is set, the length of the value is 128 bytes or longer	Number of length bytes	Number of length bytes	Number of length bytes	Number of length bytes	Number of length bytes	Number of length bytes	Number of length bytes

The Value Byte(s)

The value byte(s) can be as few as one byte or they can be a whole sequence following the Type Length Value format:

- Integers are represented by one or more bytes. An integer between 0 and 127 is represented by one byte (00 to 7F). An integer between 128 and 32767 is represented by two bytes (00 80 to 7F FF).
- A Boolean only requires one byte, since it is either true or false. The Boolean is either 0x00 or 0xFF.
- An Octet String takes up just as many bytes as the number of octets.
- Bit string: the number of bytes needed to represent a bit string depends on the number of bits and is easily explained in code:

```
int valueLength = 0;

 void CalculateValueLength()
 {
         valueLength = bitString.Length / 8;

         if((bitString.Length % 8) != 0)
         {
                valueLength++;
         }
 }
```

Request, Response, and Notification

When the host is communicating with the sensor module, all of the messages are defined as either request, response, or notification.

- Request is a message that is sent to the sensor module from the host.
- Response is a message that responds to the request.
- Notification is a message that is read by the host and is generated without any input from the host, for example a touch message or a boot complete message.

The messages look like this in the protocol:

```
*Message Definition*
    ProtocolMessage ::= CHOICE {
        request [PRIVATE 14] Message,
        response [PRIVATE 15] Message,
        notification [PRIVATE 16] Notification
    }
```

All three have the class tag private, and they are all sequences which means that the 8th, 7th, and 6th bits are all set. In binary this evaluates to 1110 0000 which in hexadecimal translates to 0xE0. Now all that needs to be done is to define which type of message it is, which in this case is either tag number 14, 15, or 16. In order to define a request, tag number 14 (0x0E) needs to be added to 0xE0, which sums up to 0xEE.

Device Address

All messages include a Device Address. Most messages go to the Air Device, but some go to the Platform Device. In the protocol, the DeviceAddress looks like this:

```
*Device Address Definition*
    DeviceAddress ::= [APPLICATION 0] OCTET STRING (SIZE (2))
    -- Addressing information used when multiple touch devices are present
    -- in the system.
    -- Byte0 - deviceType, Byte1 - deviceIndex
    -- DeviceTypes:   0x00    -   Platform
    --                0x01    -    zForce Core
    --                0x02    -    Air (Touch Sensor Module)
    --                0x03    -    zForce Plus
    --                0x04    -    Lighting devices
```

Example: Evaluate the device address

How to evaluate the address:

1. Type byte: The deviceAddress has the class tag Application, the tag number 0 and contains two octet strings. An octet string is primitive, and therefore the 6th bit is set to 0. In binary this evaluates to 0100 0000 and in hexadecimal that is 0x40.
2. Length byte: 2 (two octets).
3. Value bytes: The first byte is deviceType, the second is deviceIndex.
   a. Device type: In this case this is the Touch Sensor Module (Air), represented by 0x02.
   b. Device index: On a Touch Sensor Module (Air) the index is always 0.

The complete device address then evaluates to 0x40 0x02 0x02 0x00:

Type	Length	Value
DeviceAddress	Value length	deviceType followed by deviceIndex
0x40	0x02	0x02 0x00

Encoding a Device Configuration Message

Device Configuration will be used as an example as it is a message that contains a large number of values with both context-specific primitive values and context-specific sequences.

```
*Device Configuration ASN.1 Protocol*
-- Instance specific settings for a device
   deviceConfiguration [APPLICATION 19] Sequence {
            -- Set / get the number of touches to be tracked:
            numberOfTrackedTouches    [0] INTEGER (0..255) OPTIONAL,
            -- Set / get the minimal distance for updating a tracked touch in move state
            trackingMinDistanceMove    [1] INTEGER (0..16383) OPTIONAL,
            -- Set / get the sub touch active area low bound in X coordinate
            subTouchActiveArea        [2] Sequence {
                -- Write Request and Read Response only:
                -- Set / get the sub touch active area low bound in X coordinate
                lowBoundX        [0] INTEGER (0..16383) OPTIONAL,
                -- Set / get the sub touch active area low bound in Y coordinate
                lowBoundY       [1] INTEGER (0..16383) OPTIONAL,
                -- Set / get the sub touch active area high bound in X coordinate
                highBoundX    [2] INTEGER (0..16383) OPTIONAL,
                -- Set / get the sub touch active area high bound in Y coordinate
                highBoundY    [3] INTEGER (0..16383) OPTIONAL,
```

All settings are optional and therefore does not require all settings to be defined in the message that is sent to the sensor module.

We want to set the following settings in the sensor module

SubTouchActiveArea:

- LowBoundX: 500
- LowBoundY: 500
- HighBoundX: 2000
- HighBoundY: 2000

This is how to do it (the length bytes are represented by XX, and are added in the last step):

Step	What to do	Details	Result	The message
1	Add the code for a **request**, since the message will be sent from the host to the sensor module: 0xEE 0xXX	See Request, response and notification.	EE XX	EE XX
2	Add the **device address** 0x40 0x02 0x02 0x00	See Device Address.	40 02 02 00	EE XX **40 02 02 00**
3	Add the bytes for **deviceConfiguration**	The command deviceConfiguration has the class tag Application, tag number 19 and is a sequence.	Binary: 0111 0011 Hexadecimal: 0x73.	EE XX 40 02 02 00 **73**
4	Add the bytes for **SubTouchActiveArea**	SubTouchActiveArea is a context-specific sequence with tag number 2.	Binary: 1010 0010 Hexadecimal: 0xA2	EE XX 40 02 02 00 73 **XX A2 XX**
5	Add the variables inside of SubTouchActiveArea..	All the variables are context-specific and primitive (binary 1000 0000, hexadecimal 0x80). Depending on which variable that is currently being added, add the specific tag number.	Per variable: Binary: 1000 0000 plus tag number. Hexadecimal: 0x80 plus tag number.	EE XX 40 02 02 00 73 XX A2 XX 80 XX 01 F4 81 XX 01 F4 82 XX 07 D0 83 XX 07 D0
6	Add the length bytes of the message.	Add the number of bytes for each specific part of the message.	-	EE **18** 40 02 02 00 73 **12** A2 **10** 80 **02** 01 F4 81 **02** 01 F4 82 **02** 07 D0 83 **02** 07 D0

Read More about the Communication Protocol

- Serialization Protocol Quick Start
- USB HID Transport
- I2C Transport
- zForce Message Specification
- Understanding the zForce ASN.1 Serialization Protocol

Read More