Collaborative Mesh Networking

Instructor: Rob Faludi
Week 10

Final Project

- Discuss final project options
    - Must include collaboration
    - Should include publishing info that is used by others

Research Results: Mysteries Solved

- Startup doesn't initiate a new channel scan
- Channels are set by bits, described by hex codes
- Channels are not reset until an ATFR or ATNR0

- If no PAN is found on the selected channels, ATCH remains 0
- ATNR1 resets whole network, but if coordinator doesn't get the message the net is orphaned. Yikes!
- Good idea to pick a channel and stick with it, same as with PAN ID

Research Results: Remote Control

- Remote AT commands work!
- I/O ports can be remotely polled
  (Justin asked about this last week)
- Remember you can only use these from the XBee Series 2 API firmware
- Example code is a proof-of-concept hack
  - A proper library will be much more usable

API Mode

- Application Programming Interface
  - “An application programming interface (API) is a source code interface that an operating system or library provides to support requests for services to be made of it by computer programs.”

    http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=43487

- XBees in API mode are ready to talk to computers and microcontrollers
  - structured
  - predictable
  - reliable

API Structure

- Used in serial communications with the XBee radio
- Frames of data
  - envelope structure contains data with metadata inside a constrained format
- Radio must be in API Mode
  - AT command APAI 1 on Series 1 radios
  - API firmware on Series 2 radios

Why API

- Rather than:

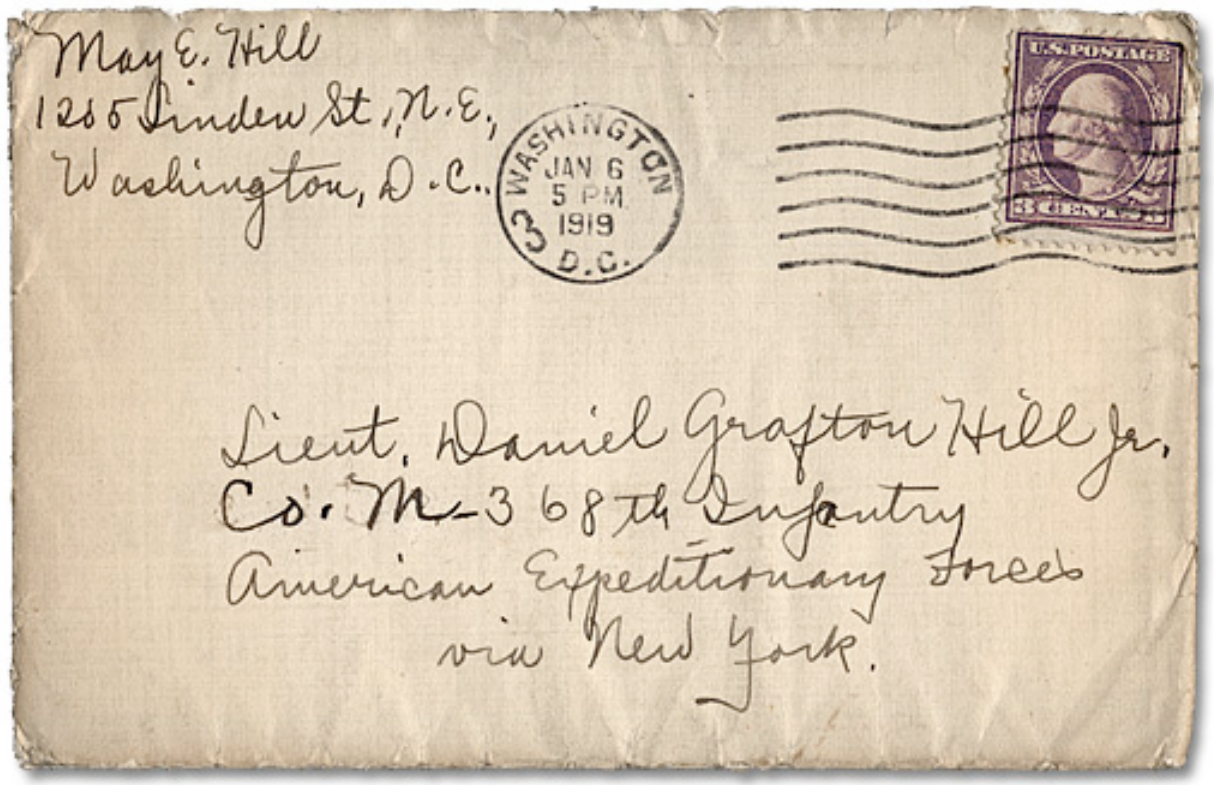
```
delay(1100);
// put the XBee in command mode
Serial.print("+++");
delay(1100);
if (checkFor("OK", 1000)) {
  Serial.println("ATID7777,CN");
   if (checkFor("OK", 1000)) {
      // if an OK was received then continue
      debugPrintln("SetupOK");
      success = true;
  }
}
```

- With a library you just write:

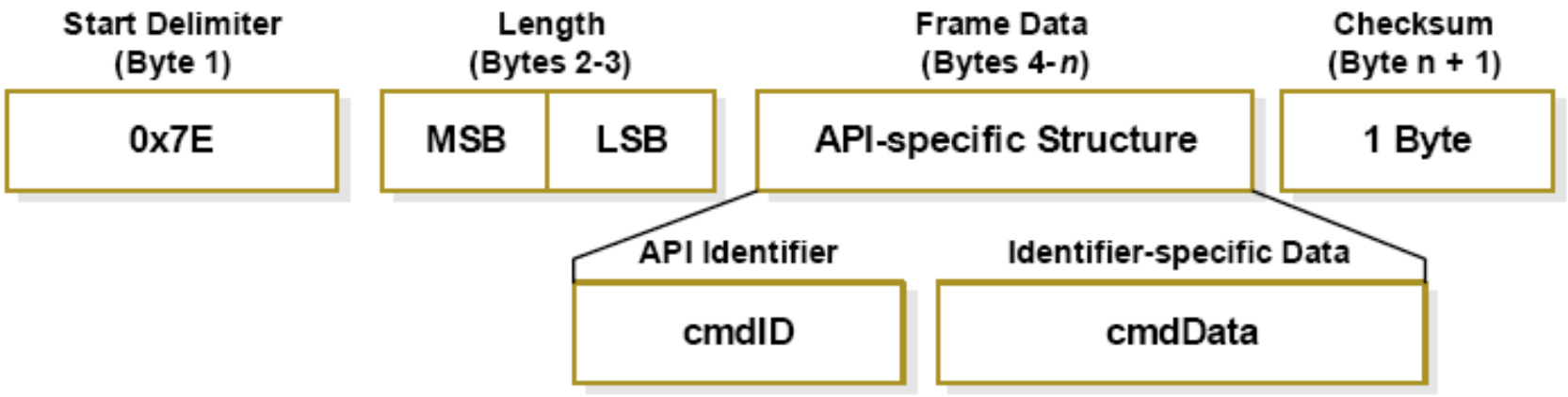
```
sendCommand(ID,0x7777);
```

Envelope Has:

- From address, to address, outside, inside, size, contents, error check



API Basic Frame Envelope



Start Byte

- 0x7E --> also known as the tilde in ASCII: ~

- First thing to do is look for it:

```
  // ARDUINO VERSION:
if (Serial.available() > 0) { // if a byte is waiting in the buffer
    inByte = Serial.read(); // read a byte from the buffer
    if (inByte == 0x7E) {
       // we're at the start of an API frame!
       // add more code here
     }
  }


  // PROCESSING VERSION:
if (port.available() > 0 {
  int inByte = port.read();
    if (inByte == 0x7E) {
       // we're at the start of an API frame!
       // add more code here
}
```

Length Bytes

- MSB: the Most Significant Byte
    - the big number
- LSB: the Least Significant Byte
    - the small number
- bit shift MSB to the right and add it to LSB

```
 // PROCESSING VERSION:
int lengthMSB = port.read(); // high byte for length of packet
int lengthLSB = port.read(); // low byte for length of packet

int lengthTotal = (lengthMSB << 8) + lengthLSB; // bit shift and add for total
```

API Identifier

- Specifies the remaining structure of the frame
  - modem status: 0x8A
  - AT command (immediate): 0x08
  - AT command (queued): 0x09
  - AT command response: 0x88
  - TX request (64 bit): 0x00
  - TX request (16 bit): 0x01
  - TX status response: 0x89
  - RX packet (64 bit): 0x80
  - RX packet (16 bit): 0x81
  - RX packet I/O data (64 bit): 0x82
  - RX packet I/O data (16 bit): 0x83

```
 // PROCESSING VERSION:
int API_ID = port.read(); // API Identifier indicates type of packet received
```

Identifier-specific Data

- Structures are different for each API identifier and might include:
  - addressing information (333B)
  - status information (received OK)
  - source information (broadcast packet)
  - unstructured data (“Hello World, this is Rob!”)
  - structured data (typically for I/O packets)

Checksum

- Simple check to detect errors
- To calculate: Not including frame delimiters and length, add all bytes keeping only the lowest 8 bits of the result and subtract from 0xFF.
- To verify: Add all bytes (include checksum, but not the delimiter and length). If the checksum is correct, the sum will equal 0xFF.

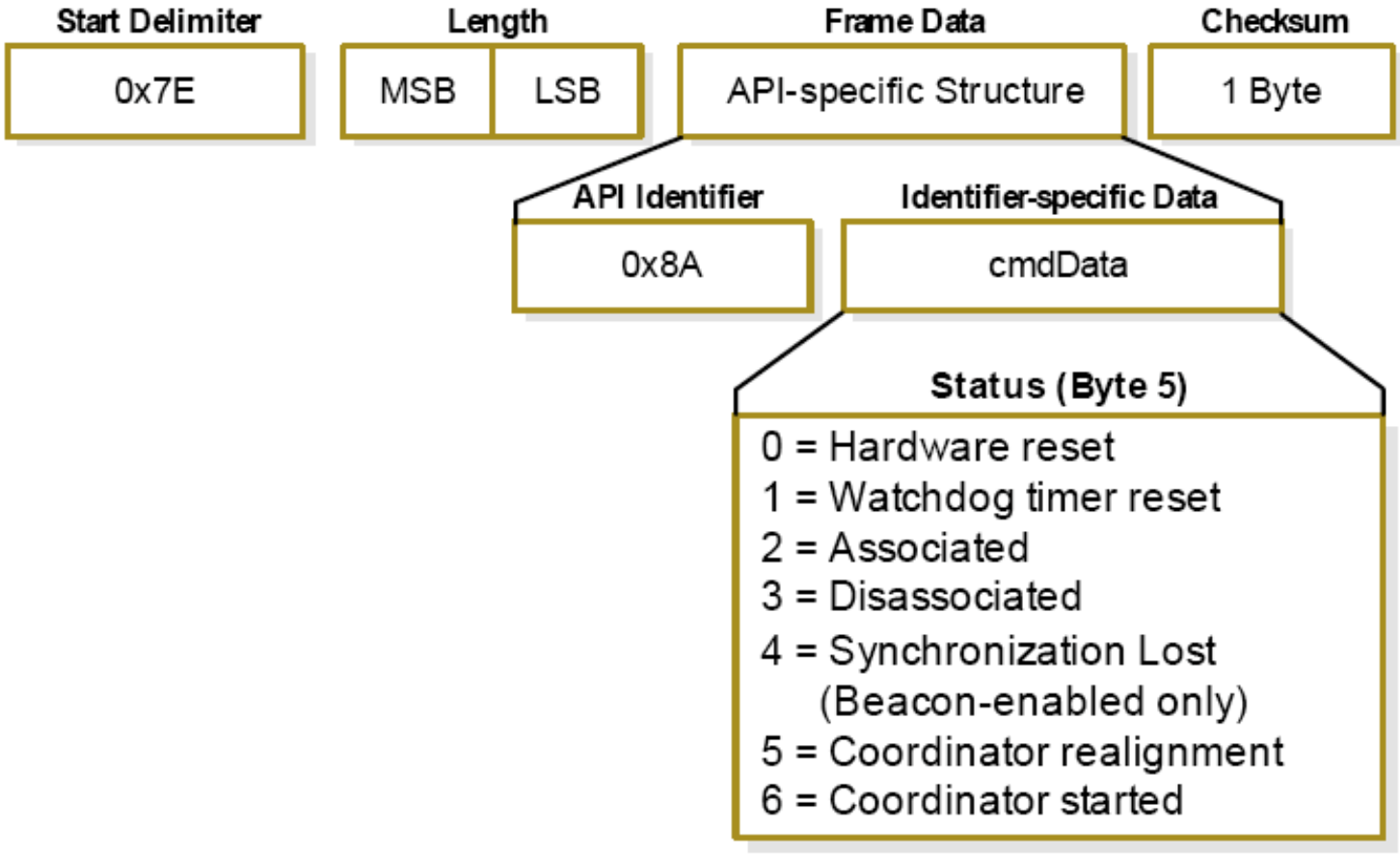
```
 // PROCESSING VERSION:
int localChecksum = (API_ID + addrMSB + addrLSB + RSSI + options + dataSum);

int checksum = port.read();
localChecksum = byte(0xFF -localChecksum);

if ( (byte) checksum - localChecksum == 0) {
  returnVal = dataADC[0];
}
else {
  print("\n\nchecksum error!  " + "\n\n");
}
```

Many Kinds of Envelopes

Modem Status



AT Command

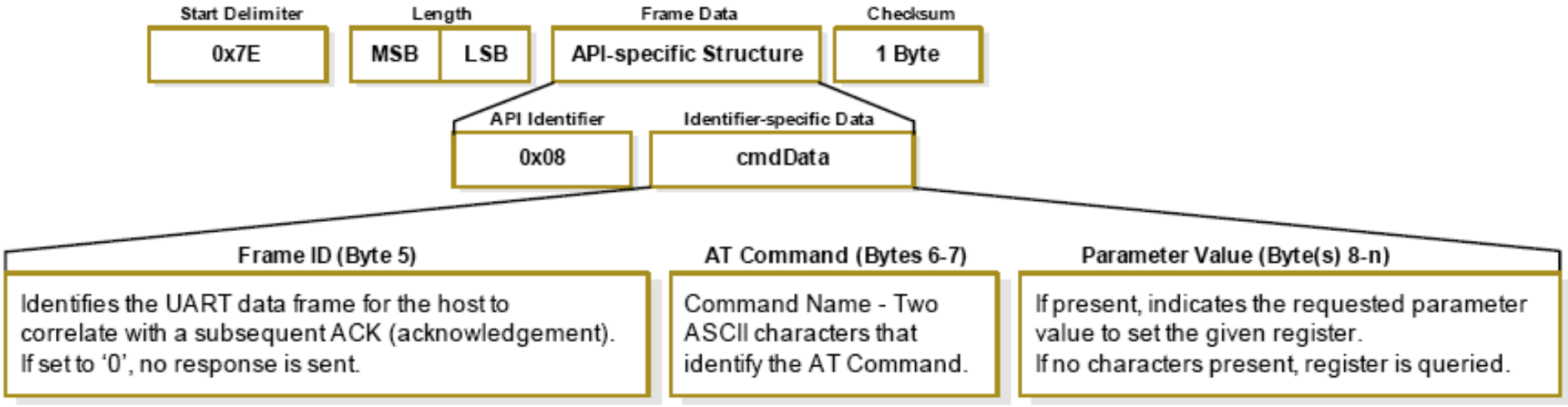
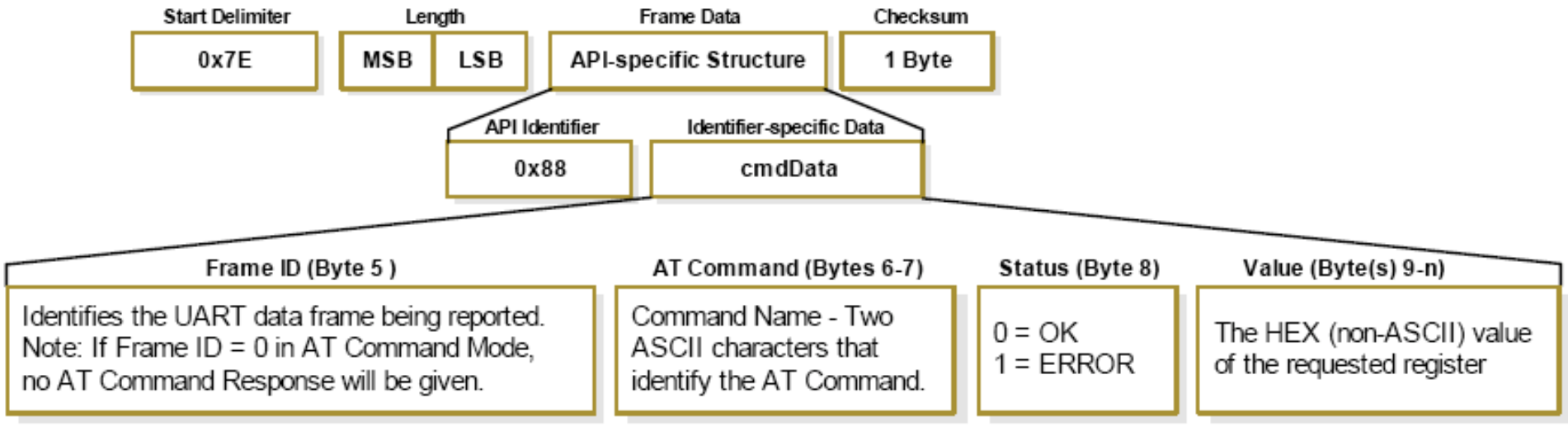


Figure 3-06. Example: API frames when reading the DL parameter value of the module.

Byte 1	Bytes 2-3	Byte 4	Byte 5	Bytes 6-7	Byte 8
0x7E	0x00 0x04	0x08	0x52 (R)	0x44 (D) 0x4C (L)	0x15
Start Delimiter	Length*	API Identifier	Frame ID**	AT Command	Checksum

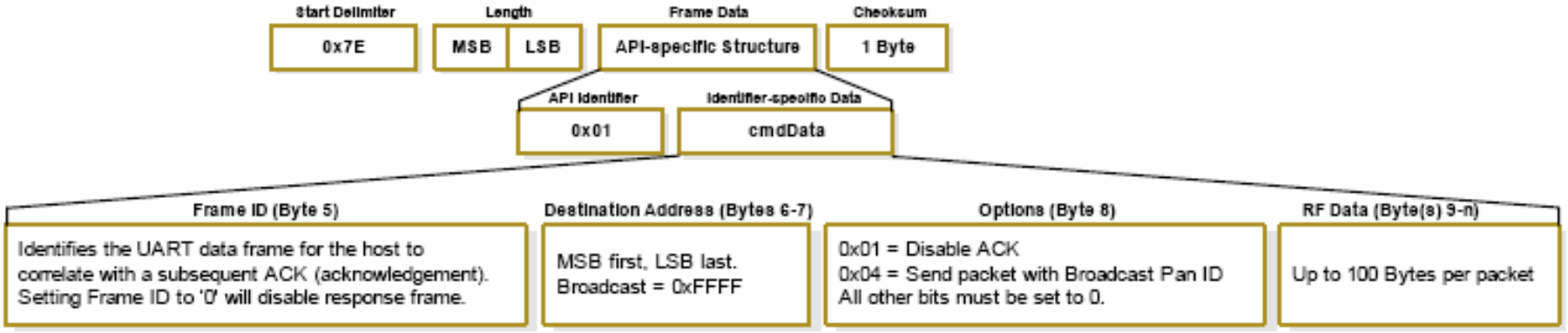
AT Response

- Frame ID for the response is the same as the matching AT Command request



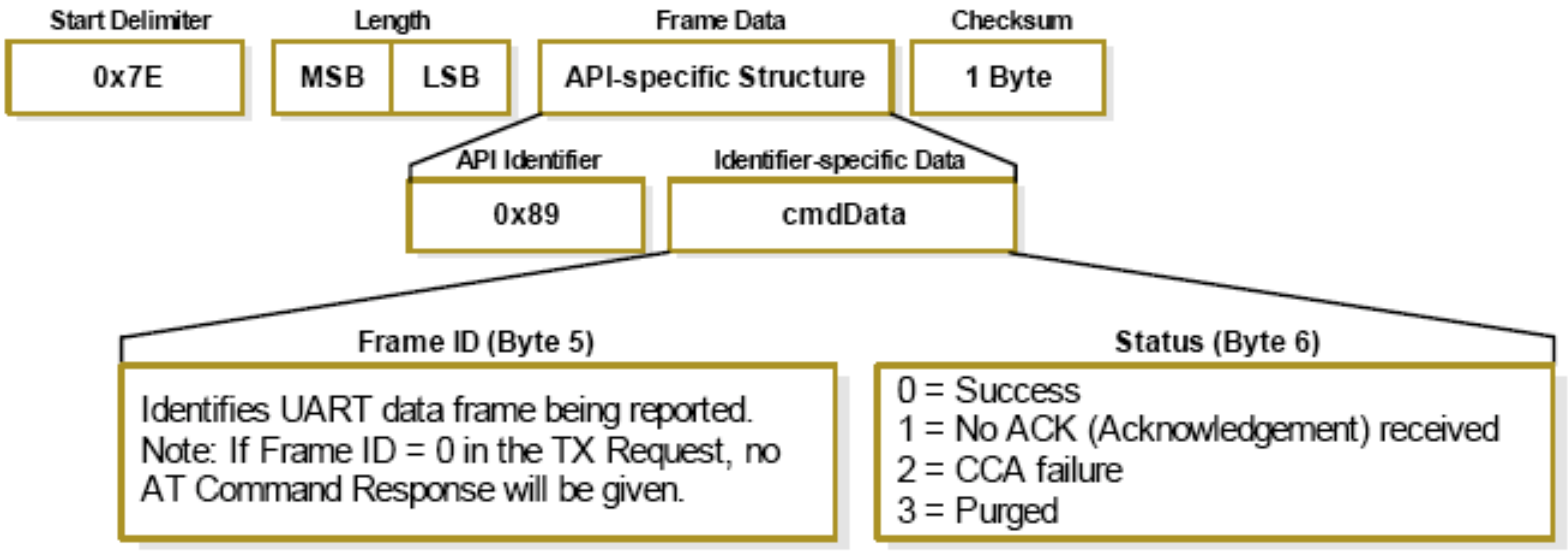
TX (Transmit) Request

- Remember that this is a request
- Also need to check for results by Frame ID



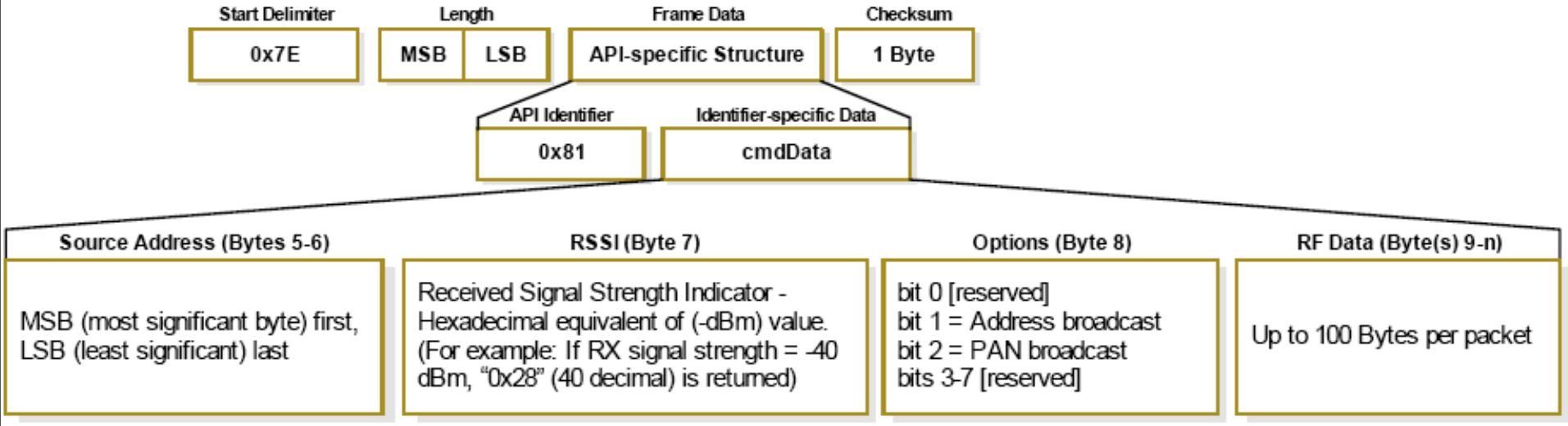
TX Status (Results)

- See if your message was transmitted or not
- Use your Frame ID to see which message is being described



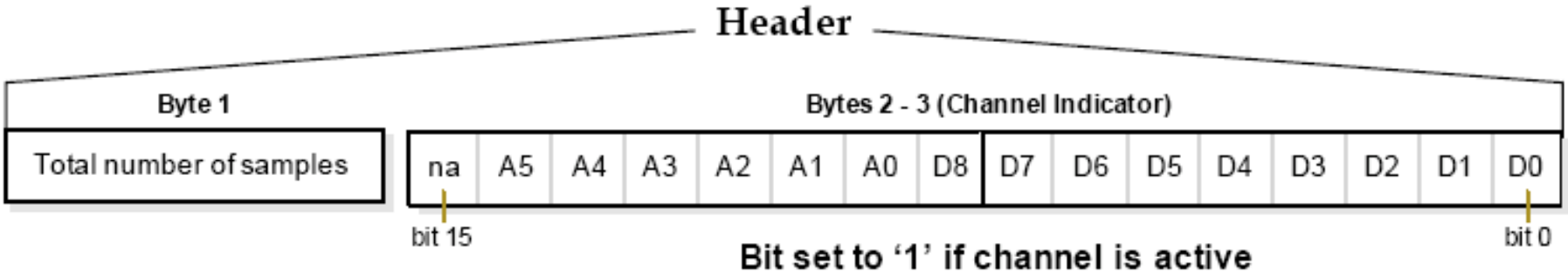
RX Packet (16 bit addressing)

- Maximum of 100 bytes of data per packet
- RF Data section is basis for I/O packets



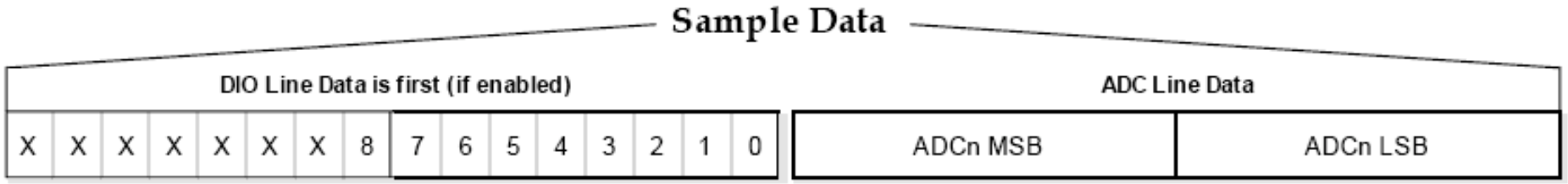
I/O Data Header

- Inside the RF Data section of the RX Packet
  - Total number of samples set with ATIS
  - Channels set with ATD0...9



I/O Data Sample

- Follows the header
- Two bytes of digital data IF ANY DIGITAL CHANNELS ENABLED followed by...
- ...two bytes for EACH analog channel enabled...
- ...then repeats for each sample



- How many bytes if ATIS5 ATD02 ATD12 ATD23?

I/O Code: Basic

- Fixed parameters make for easier programming

- Assume we are just reading a single sample of one ADC channel at a time:

```
 // PROCESSING VERSION:
int totalSamples = port.read(); // this is the number of samples we're receiving
int channelIndicatorHigh = port.read(); // this tells us which analog channels
                  // are in use (and one digital channel)
int channelIndicatorLow = port.read();  // this tells us which digital channels
                  // are in use.

int  dataADCMSB = port.read(); // read in the most significant ADC byte
int dataADCLSB = port.read(); // read in the least significant ADC byte
int dataADC = (dataADCMSB << 8) + dataADCLSB;  // bit shift the MSB into
                  // position and add it to the LSB

      print(dataADC);  // print the information
}
```

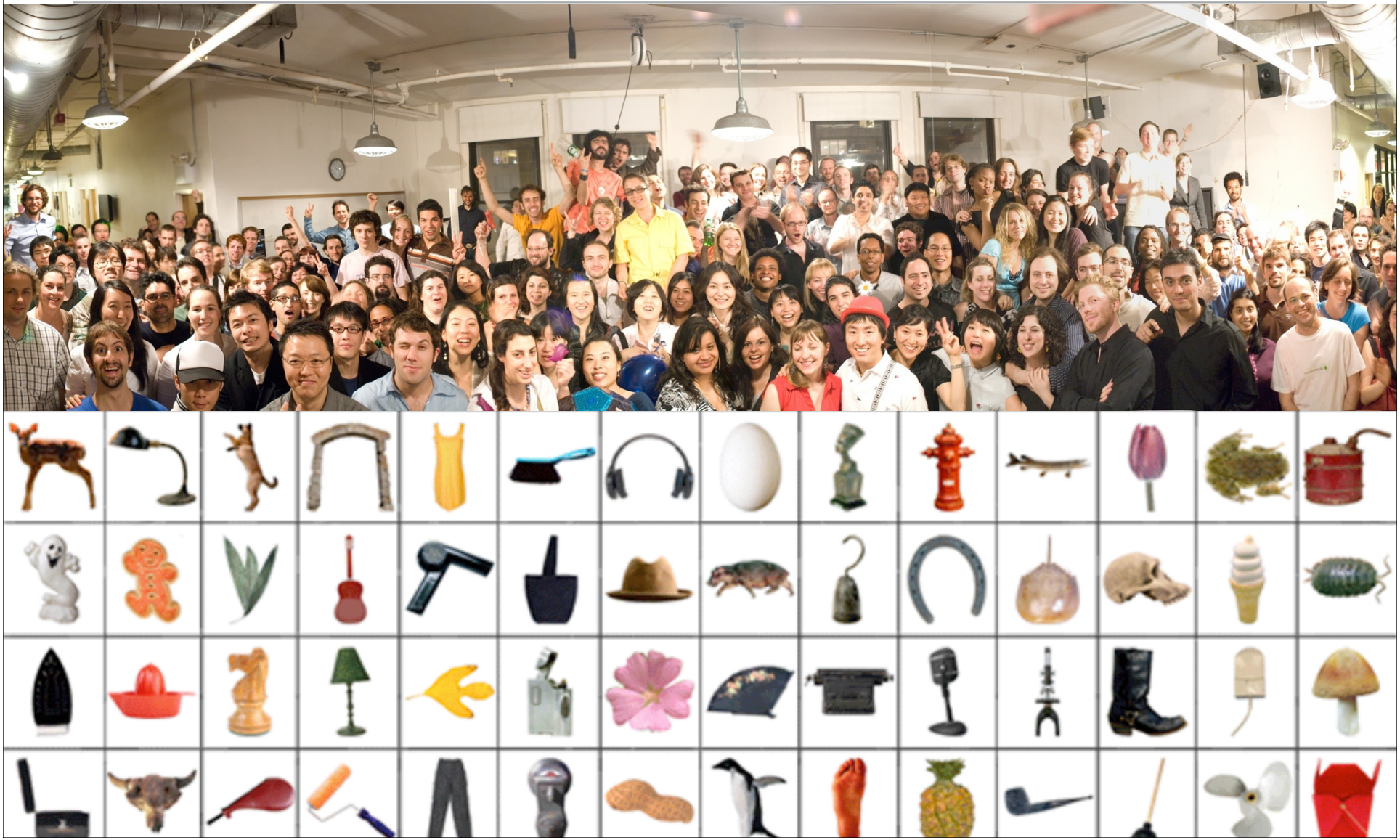
Sleeping the XBee: Review

- Why Sleep?
- ATSM
  - 1: pin hibernate, <10 µA, 13.2 ms wakeup, uses pin 9
  - 2: pin doze, <50 µA, 2 ms wakeup
  - 3: <nothing>
  - 4: cyclic sleep, also <50 µA, 2 ms wakeup, module must be idle
  - 5: cyclic sleep with pin wakeup
- ATSP: Sleep Period (* 10 ms)
- ATST: Time before Sleep (* 1 ms)

Sleeping the XBee: Example

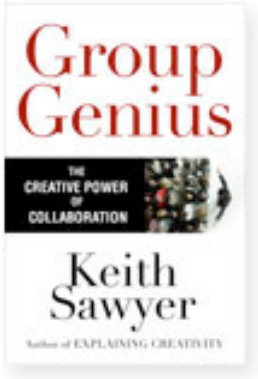
- ATSM5,SP64,ST14
  - Will wake up on pin 9 high, and also every 1000 ms for 20 ms
- Use in conjunction with I/O readings
  - Wakeup will always trigger an I/O sample
  - More samples if ATIR allows it during the awake period
  - More samples if ATIT (Samples before TX) is set
- ATIC (Pin Change Detect) will not affect wakeup

Group Genius



Readings and Assignments

- Readings
  - Group Genius by Keith Sawyer, Part 1
    Read critically!



- Assignment
  - Final project plan