The most up-to-date versions of these files should be in the grand-central-arduino GitHub repository.

The StormNetGeneric sketch can be used as a baseline sketch for the sensors (and lights) that we’ve discussed. There are a few nice features that allow you to:

1. Start with a simple set of commands for demonstrating basic reads and writes
2. Talk to the device directly via either USB or Serial without duplicating any code
3. Jump to TODO comments to show where to put different elements

To add a new command stream, you need to define a new mode, a command to recognize the mode, and then a handler to implement the details. For example, to add a new US sensor you might create MODE\_ULTRASONIC, set the command to set that mode to “U” and write a handleUltrasonicRequest() function that writes the float output of the different sensors into the array.

Here is a slightly simplified snippet of the interrupt handlers for reference:

**void** **receiveEvent**(**int** howMany) { // handles i2c write event from master  
 **char** c = readIt();  
 **switch** (c) {  
 **case** 'P':  
 commandMode = MODE\_PING;  
 **break**;  
 **case** 'F':  
 commandMode = MODE\_FAST;  
 **break**;  
 // TODO - recognize additional modes   
 **case** '\0':  
 **default:**  
 commandMode = MODE\_DEFAULT;  
 }  
}  
  
**void** **requestEvent**() {  
 **switch**(commandMode) {  
 **case** MODE\_PING:  
 handlePingRequest();  
 **break**;  
 **case** MODE\_FAST:   
 handleFastRequest();  
 **break**;  
 // TODO - add mode handlers   
 **case** MODE\_DEFAULT:  
 **default:**  
 handleDefaultRequest();  
 }  
}

And one of the handlers:

**void** **handleDefaultRequest**() {  
 writeIt((**void**\*)&counter, **sizeof**(**unsigned** **int**), true);  
 counter++;  
}

The writeIt function will send data to USB if that is connected and the magical “serialMode” variable is set to true (this has to be changed manually right now). The same function will send data over the I2C bus if not using USB. The last argument to the function indicates that this is binary data (rather than text) so that the USB output can be understood, at least at the byte level.

The sketch incorporates the status LED approach we’ve seen, and also implements a basic ping, and two commands to change the speed of the LED. The default command (what we send in response to an unrecognized request) just increments a counter. This is helpful to demonstrate that something interesting is happening on the device.

The StormNetMaster.ino sketch is a very basic master that already knows the built-in commands.You can hook the usb to the master device and it can talk I2C over to the slave. It can be extended in a very similar way to understand new commands running on the slave as well. The master will be replaced with the java code running on the roborio, but is useful for debugging without the robot.

**Known issues**

I noticed that the blink update isn't 100% reliable. I think the problem is that blinkInterval gets updated in the interrupt for requestEvent and that sets up a race condition in the loop where the blink itself is controlled(?). That block of code (and similar blocks for other sensors) should probably be bracketed by

noInterrupts();

…

interrupts();

to queue the event handler until after the checks are done. We need to be smart about this, though, since timing is important. We wouldn't want the whole time for the US sensor ping in one of these blocks, for example.

[Later]

That appears to be it. I’m keeping the note above so we are aware of the potential for racing problems, but the blink seems to be resolved. I also made a number of variables that could be changed by the interrupt routines into “volatile” variables. This is a hint to the compiler to avoid certain optimizations since the value of the variable might change at unexpected times.