Goals

• Review
  • Programming in Sketch
• Communication
  • SPI, I2C
• Data logging
  • SD cards
• Accessing the Linux host
  • Shells
  • GPIO

• Networking
  • LAN
  • WiFi

• Network Communication
  • REST
  • WebSockets
  • WebRTC

• Cloud
  • Data sharing
Goals

• **Review**
  • Programming in Sketch

• **Communication**
  • SPI, I2C

• **Data logging**
  • SD cards

• **Accessing the Linux host**
  • Shells
  • GPIO

• **Networking**
  • LAN
  • WiFi

• **Network Communication**
  • REST
  • WebSockets
  • WebRTC

• **Cloud**
  • Data sharing
IO Pins

- Two states (binary signal) versus multiple states (continuous signal)
- Digital – (LEDs, switches)
- Analog – resistive sensor data
Programming – Analog

- Can a digital devise produce analog output?

- Analog output can be simulated using pulse width modulation (PWM)

Image from Theory and Practice of Tangible User Interfaces at UC Berkley
Pulse Width Modulation

- Can’t use digital pins to directly supply say 2.5V, but can pulse the output on and off really fast to produce the same effect.

- The on-off pulsing happens so quickly, the connected output device “sees” the result as a reduction in the voltage.
Remember that it isn't all about programming! Wire your circuit wrongly, and you can fry some board pins and smell them too!
Standard Libraries

- **EEPROM** - reading and writing to "permanent" storage
- **Ethernet** - for connecting to the internet using the Arduino Ethernet Shield
- **Firmata** - for communicating with applications on the computer via serial protocol.
- **GSM** - for connecting to a GSM/GRPS network with the GSM shield.
- **LiquidCrystal** - for controlling liquid crystal displays (LCDs)
- **SD** - for reading and writing SD cards
- **Servo** - for controlling servo motors
- **SPI** - for communicating with devices using Serial Peripheral Interface (SPI) Bus
- **SoftwareSerial** - for serial communication on any digital pins. Version 1.0 and later of Arduino incorporate Mikal Hart's NewSoftSerial library as SoftwareSerial.
- **Stepper** - for controlling stepper motors
- **TFT** - for drawing text, images, and shapes on the Arduino TFT screen
- **WiFi** - for connecting to the internet using the Arduino WiFi shield
- **Wire** - Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors.
Goals

- Review
  - Programming in Sketch

- **Communication**
  - SPI, I2C

- Data logging
  - SD cards

- Accessing the Linux host
  - Shells
  - GPIO

- Networking
  - LAN
  - WiFi

- Network Communication
  - REST
  - WebSockets
  - WebRTC

- Cloud
  - Data sharing
Communication Interfaces
Communication Protocols

I2C

Serial Peripheral Interface (SPI)
I2C Code Example

**Master**

```cpp
#include <Wire.h>

void setup()
{
  Wire.begin(); // join i2c (address optional for master)
}

byte x = 0;

void loop()
{
  Wire.beginTransmission(4); // transmit to device #4
  Wire.write("x is "); // sends five bytes
  Wire.write(x); // sends one byte
  Wire.endTransmission(); // stop transmitting
  x++;
  delay(500);
}
```

**Slave**

```cpp
#include <Wire.h>

void setup()
{
  Wire.begin(4); // join i2c bus with address #4
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600); // start serial for output
}

void loop()
{
  delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  while(1 < Wire.available()) // loop through all but the last
  {
    char c = Wire.read(); // receive byte as a character
    Serial.print(c); // print the character
  }
  int x = Wire.read(); // receive byte as an integer
  Serial.println(x); // print the integer
}
```
Serial Peripheral Interface (SPI)

• 3-Wire (plus 1 chip-select) with unique chip-select lines

• Typically there are three lines common to all the devices:
  • MISO (Master In Slave Out)
    • The Slave line for sending data to the master,
  • MOSI (Master Out Slave In)
    • The Master line for sending data to the peripherals,
  • SCK (Serial Clock)
    • The clock pulses which synchronize data transmission generated by the master
  • SS (Slave Select)
    • The pin on each device that the master can use to enable and disable specific devices.
    • When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.
SPI Peripheral Types

- Converters (ADC, DAC)
- Memories (EEPROM, RAM’s, Flash)
- Sensors (Temperature, Humidity, Pressure)
- Real Time Clocks
- Misc- Potentiometers, LCD controllers, UART’s, USB controller, CAN controller, amplifiers
SPI Code Example

#include <SPI.h>

// By default, 11 = MOSI, 12 = MISO, 13 = CLK
// set pin 10 as the slave select
const int slaveSelectPin = 10;

void setup() {
  pinMode(slaveSelectPin, OUTPUT);
  // initialize SPI:
  SPI.begin();
}

void loop() {
  ... digitalPotWrite(channel, level);
  ...
}

void digitalPotWrite(int address, int value) {
  // take the SS pin low to select the chip:
  digitalWrite(slaveSelectPin, LOW);
  // send in the address and value via SPI:
  SPI.transfer(address);
  SPI.transfer(value);
  // take the SS pin high to de-select the chip:
  digitalWrite(slaveSelectPin, HIGH);
}
Goals

- Review
  - Programming in Sketch
- Communication
  - SPI, I2C
- Data logging
  - SD cards
- Accessing the Linux host
  - Shells
  - GPIO
- Networking
  - LAN
  - WiFi
- Network Communication
  - REST
  - WebSockets
  - WebRTC
- Cloud
  - Data sharing
Data Logging Procedure

- Initialize SPI
- Initialize SD Card
- Format the SD card
  - FAT
  - FAT32
- Read/Write with APIs
- Starting from there, you will have to build your own file system...
#include <SD.h>
const int chipSelect = 4;

void setup()
{
    // Open serial communications and wait for port to open:
    Serial.begin(9600);
    while (!Serial)
    {
        // wait for serial port to connect. Needed for Leonardo only
    }

    Serial.print("Initializing SD card...");
    // make sure that the default chip select pin is set to
    // output, even if you don't use it:
    pinMode(10, OUTPUT);

    // see if the card is present and can be initialized:
    if (!SD.begin(chipSelect))
    {
        Serial.println("Card failed, or not present");
        // don't do anything more:
        return;
    }

    Serial.println("card initialized.");

    // open the file. note that only one file can be open at a time,
    // so you have to close this one before opening another.
    File dataFile = SD.open("datalog.txt");

    // if the file is available, write to it:
    if (dataFile)
    {
        while (dataFile.available())
        {
            Serial.write(dataFile.read());
        }
        dataFile.close();
    }
    // if the file isn't open, pop up an error:
    else
    {
        Serial.println("error opening datalog.txt");
    }
}

void loop() {}
Data Logging with Timestamps
Time-stamped Data

- Keep track of time even after the power has been disconnected
- Useful to analyze data over time

<table>
<thead>
<tr>
<th></th>
<th>Date</th>
<th>Time</th>
<th>Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>New Log Started!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Date</td>
<td>Time</td>
<td>Phrase</td>
</tr>
<tr>
<td>4</td>
<td>4/8/2013</td>
<td>23:24:05</td>
<td>Hello There!</td>
</tr>
<tr>
<td>5</td>
<td>4/8/2013</td>
<td>23:24:07</td>
<td>Hello There!</td>
</tr>
<tr>
<td>8</td>
<td>4/8/2013</td>
<td>23:24:14</td>
<td>Hello There!</td>
</tr>
<tr>
<td>14</td>
<td>4/8/2013</td>
<td>23:24:26</td>
<td>Hello There!</td>
</tr>
<tr>
<td>16</td>
<td>4/8/2013</td>
<td>23:24:30</td>
<td>Hello There!</td>
</tr>
<tr>
<td>18</td>
<td>4/8/2013</td>
<td>23:24:34</td>
<td>Hello There!</td>
</tr>
</tbody>
</table>
Goals

• Review
  • Programming in Sketch

• Communication
  • SPI, I2C

• Data logging
  • SD cards

• Accessing the Linux host
  • Shells
  • GPIO

• Networking
  • LAN
  • WiFi

• Network Communication
  • REST
  • WebSockets
  • WebRTC

• Cloud
  • Data sharing
Accessing Linux from a Sketch

- The loop reads the command string you want executed as typed in the serial monitor window.
- Upon detecting a return character, the command string is executed on Arduino using the `system()` command.
- The output of the command is returned back to the monitor by the redirection to `/dev/ttySG0`.

```c
void loop()
{
    if (Serial.available())
    {
        char c = Serial.read();
        if (c == '\n')
        {
            Serial.println();
            if (offset >= MAXLINE)
            {
                Serial.println("Too many characters in line");
            }
            else
            {
                buf[offset] = 0;
                strcat(buf, redirect);
            }
        }
        offset = 0;
    }
    else
    {
        if (offset < MAXLINE)
        {
            buf[offset++] = c;
            Serial.write(c);
        }
    }
}
```
Intel Galileo - GPIO From Linux

- Example code for accessing GPIO pins from a shell
  - NOTE!!! GPIO #s here do not correspond to physical board pin layout and numbers

- Subscribe into /sysfs
  
  ```
  root@clanton:~# echo -n "4" > /sys/class/gpio/export
  
  root@clanton:~# echo -n "out" > /sys/class/gpio/gpio4/direction
  root@clanton:~# echo -n "1" > /sys/class/gpio/gpio4/value
  root@clanton:~# echo -n "0" > /sys/class/gpio/gpio4/value
  ```

- Unsubscribe from /sysfs
  
  ```
  root@clanton:~# echo -n "4" > /sys/class/gpio/unexport
  ```

- Check out [http://www.malinov.com/Home/sergey-s-blog](http://www.malinov.com/Home/sergey-s-blog) for more information!
Intel Galileo - IO Mapping Between Pins and Linux

<table>
<thead>
<tr>
<th>Arduino (DIE ID)</th>
<th>Source</th>
<th>GPIO</th>
<th>Pin</th>
<th>PWM</th>
<th>Dir</th>
<th>Mixed with</th>
<th>Initial Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>I00</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>50</td>
<td>N/A</td>
<td>BI</td>
<td>UART0_RXD</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I01</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM1</td>
<td>51</td>
<td>N/A</td>
<td>BI</td>
<td>UART0_TXD</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I02</td>
<td>Sec</td>
<td>GPD0&lt;6&gt;</td>
<td>54</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I03</td>
<td>Sec</td>
<td>GPD0&lt;7&gt;</td>
<td>55</td>
<td>1</td>
<td>BI</td>
<td>PWM</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I04</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM1</td>
<td>28</td>
<td>-</td>
<td>BI</td>
<td>-</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I05</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>17</td>
<td>5</td>
<td>BI</td>
<td>PWM</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I06</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM1</td>
<td>24</td>
<td>6</td>
<td>-</td>
<td>PWM</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I07</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM1</td>
<td>24</td>
<td>6</td>
<td>-</td>
<td>PWM</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I08</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>26</td>
<td>-</td>
<td>BI</td>
<td>-</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I09</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM1</td>
<td>19</td>
<td>1</td>
<td>BI</td>
<td>PWM</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I10</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>16</td>
<td>7</td>
<td>BI</td>
<td>PWM</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I11</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>25</td>
<td>4</td>
<td>BI</td>
<td>PWM</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I12</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>26</td>
<td>-</td>
<td>BI</td>
<td>SPI_MISO</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I13</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>26</td>
<td>-</td>
<td>BI</td>
<td>SPI_MISO</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I14</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>44</td>
<td>-</td>
<td>BI</td>
<td>AD7298_VIN0</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I15</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>45</td>
<td>-</td>
<td>BI</td>
<td>AD7298_VIN1</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I16</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>46</td>
<td>-</td>
<td>BI</td>
<td>AD7298_VIN2</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I17</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>47</td>
<td>-</td>
<td>BI</td>
<td>AD7298_VIN3</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I18</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM2</td>
<td>48</td>
<td>-</td>
<td>BI</td>
<td>AD7298_VIN4</td>
<td>w/ pull up off</td>
</tr>
<tr>
<td>I19</td>
<td>Cypr</td>
<td>GPORT1_BTS_PWM4</td>
<td>49</td>
<td>-</td>
<td>BI</td>
<td>AD7298_VIN5</td>
<td>w/ pull up off</td>
</tr>
</tbody>
</table>

* See Galileo I/O Function Muxing table below.
Most of GPIO capabilities of Galileo board are exposed through Linux Sysfs interface, and can be controlled using file based I/O. I will show how to use some of these capabilities using simple shell commands. Of course instead of shell you can implement I/O using file manipulations from your program written using your favorite programming language.

Digital GPIO - Sysfs Interface

GPIO Information

The following command gives information about GPIO in the system and shows if an I/O port was allocated to a module or Sysfs (user).

```bash
root@clanton:~# cat /sys/kernel/debug/gpio
GPIOs 0-1, platform/sch_gpio.2398, sch_gpio_core:

GPIOs 2-7, platform/sch_gpio.2398, sch_gpio_resume:

GPIOs 8-15, intel_cli_gip_gpio:
gpio-8  (SPI_CS ) out  hi
gpio-10 (SPI_CS ) out  hi
gpio-13 (cy8c9540a-int ) in  hi

GPIOs 16-55, cy8c9540a, can sleep:
```

As you can see from the output all the GPIOs of Galileo board is divided into 4 chunks:

1. GPIOs 0-1 - Intel Quark X1000 - GPIO[9:8] pins. These are GPIO pins on Legacy I/O bridge. They are powered and active in S0 state only.
2. GPIOs 2-7 - Intel Quark X1000 - GPIO_SUS[5:0] pins. These are GPIO pins on Legacy I/O bridge. They are powered and active in S3 (suspend) and S0 states.
3. GPIOs 8-15 - Intel Quark X1000 - GPIO[7:0] pins. These are GPIO pins on GPIO controller. They are powered and active in S0 state only.
4. GPIOs 16-55 - Cypress CY8C9540A I/O Expander

Exporting a GPIO Port to Sysfs

To make GPIO port controllable from sysfs you'll need to export it. This is done by writing GPIO port number to /sys/class/gpio/export:

```bash
root@clanton:~# echo -n "27" > /sys/class/gpio/export
```

When this operation completes successfully a directory corresponding to the GPIO port number will appear in sysfs. In this case /sys/class/gpio/gpio27. Once you finished working with I/O you should un-export it by writing the GPIO port number to /sys/class/gpio/unexport.
Intel Galileo - Arduino and Linux Touchdown

/* This example shows how to read the temperature of the Quark SoC */

char temp_raw[6]; int temp;
void setup() {
    Serial.begin(115200);
}

void loop() {
    temp = getQuarkTemp();
    Serial.print("The temperature of the Quark SoC is ");
    Serial.print(temp);
    Serial.println(" degrees celcius.");
    delay(1000);
}

int getQuarkTemp() {
    FILE *fp;
    fp = fopen("/sys/class/thermal/thermal_zone0/temp", "r");
    fgets(temp_raw, 5, fp);
    fclose(fp);
    int temp = atoi(temp_raw);
    temp /= 100;
    return temp;
}
Goals

• Review
  • Programming in Sketch

• Communication
  • SPI, I2C

• Data logging
  • SD cards

• Accessing the Linux host
  • Shells
  • GPIO

• Networking
  • LAN
  • WiFi

• Network Communication
  • REST
  • WebSockets
  • WebRTC

• Cloud
  • Data sharing
Intel Galileo WiFi

http://www.malinov.com/Home/sergey-s-blog/intelgalileo-addingwifi
Configuring Wireless In Linux

Intel Supplied SD Card Image

Intel supplied SD card image only includes iwlwifi-135-6.ucode and iwlwifi-6000g2a-6.ucode firmware. In my case for the N6235 card I needed to get iwlwifi-6000g2b-6.ucode from http://wireless.kernel.org/en/users/Drivers/iwlwifi (it is in iwlwifi-6000g2b-ucode-18.168.6.1.tgz file). To install firmware on my Galileo I copied the iwlwifi-6000g2b-ucode-18.168.6.1.tgz archive to SD card, booted the Galileo, unzipped and copied firmware to the /lib/firmware directory:

```
root@clanton:/tmp# tar xzvf /media/mmcblk0p1/iwlwifi-6000g2b-ucode-18.168.6.1.tgz
```

Once proper firmware is installed use the following steps to configure wireless (this assumes using WPA security).

1. Login to your Galileo as root
2. Run the following command to generate wpa_supplicant configuration file for your network. Replace MyWiFi with the SSID of your wireless network and MyPassPhrase with the real passpharse:

   ```
   root@clanton:~# wpa_passphrase MyWiFi << EOF > /etc/wpa_supplicant.conf
   > MyPassPhrase
   > EOF
   ```

3. If you wish to configure your Galileo to connect to the wireless network automatically, edit /etc/network/interfaces file and add auto wlan0 line somewhere in the file. I suggest putting it right before iface wlan0 line:

   ```
   # /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
   
   # The loopback interface
   auto lo
   iface lo inet loopback
   
   # Wireless interfaces
   auto wlan0
   iface wlan0 inet dhcp
       wireless_mode managed
       wireless_ssid any
       wpa-driver wext
       wpa-conf /etc/wpa_supplicant.conf
   ```

4. Restart networking:

   ```
   root@clanton:/etc# /etc/init.d/networking restart
   Running /etc/init.d/networking restart is deprecated because it may not enable again some interfaces
   Reconfiguring network interfaces...
   ifdown: interface wlan0 not configured
   ifdown: interface eth0 not configured
   uhdpcp (v1.20.2) started
   ```
Uno WiFi Shield

- Connection via: 802.11b/g networks
- Encryption types: WEP and WPA2 Personal
- Connection with Arduino on SPI port
- on-board micro SD slot
- ICSP headers
- FTDI connection for serial debugging of WiFi shield
- Mini-USB for updating WiFi shield firmware
Programming the WiFi Shield

#include <WiFi.h>

char ssid[] = "yourNetwork";     // your network SSID (name)
char pass[] = "12345678";        // your network password
int status = WL_IDLE_STATUS;     // the Wifi radio's status

void setup() {
    // initialize serial:
    Serial.begin(9600);

    // attempt to connect using WPA2 encryption:
    Serial.println("Attempting to connect to WPA network...");
    status = WiFi.begin(ssid, pass);

    // if you're not connected, stop here:
    if ( status != WL_CONNECTED) {
        Serial.println("Couldn't get a wifi connection");
        while(true);
    } // if you are connected, print out info about the connection:
    else {
        Serial.println("Connected to network");
    }
}

void loop() {
    // do nothing
}
Ethernet Shields
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };  
byte ip[] = { 192, 168, 1, 177 };  // change to a valid address for your network
byte server[] = { 209,85,229,104 };  // Google

// see text for more on IP addressing

EthernetClient client;

void setup()
{
  Serial.begin(9600);         // start the serial library:
  Ethernet.begin(mac,ip);
  delay(1000);               // give the ethernet hardware a second to initialize

  Serial.println("connecting...");

  if (client.connect(server,80)) {
    Serial.println("connected");
    client.println("GET /search?q=arduino HTTP/1.0");  // the HTTP request
    client.println();
  }
  else {
    Serial.println("connection failed");
  }
}

void loop()
{
  if (client.available()) {
    char c = client.read();
    Serial.print(c);  // echo all data received to the Serial Monitor
  }

  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    for(;;)
      ;
  }
}
```cpp
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };  
byte ip[] = { 192, 168, 1, 177 };  // IP address of this web server

EthernetServer server(80);

void setup()
{
    Etherent.begin(mac, ip);
    server.begin();
}

void loop()
{
    EthernetClient client = server.available();
    if (client) {
        // an http request ends with a blank line
        boolean current_line_is_blank = true;
        while (client.connected()) {
            if (client.available()) {
                char c = client.read();
                // if we've gotten to the end of the line (received a newline
                // character) and the line is blank, the http request has ended,
                // so we can send a reply
                if (c == '\n' && current_line_is_blank) {
                    // send a standard http response header
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                    client.println();

                    // output the value of each analog input pin
                    for (int i = 0; i < 6; i++) {
                        client.print("analog input ");
                        client.print(i);
                        client.print(" ");
                        client.print(analogRead(i));
                        client.println("<br />");
                    }
                    break;
                }
            }
            if (c == '\n') {
                // we're starting a new line
                current_line_is_blank = true;
            } else if (c != '\r') {
                // we've gotten a character on the current line
                current_line_is_blank = false;
            }
        }
        // give the web browser time to receive the data
        delay();
        client.stop();
    }
}
```

Display Info to a Website
void loop()
{
    EthernetClient client = server.available();  // try to get client

    if (client) {  // got client?
        boolean currentLineIsBlank = true;
        while (client.connected()) {
            if (client.available()) {  // client data available to read
                char c = client.read(); // read 1 byte (character) from client
                // last line of client request is blank and ends with \n
                // respond to client only after last line received
                if (c == '\n' && currentLineIsBlank) {
                    // send a standard http response header
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                    client.println("Connection: close");
                    client.println();
                    // send web page
                    webFile = SD.open("index.htm");        // open web page file
                    if (webFile) {
                        while(webFile.available()) {
                            client.write(webFile.read()); // send web page to client
                        }
                        webFile.close();
                    }
                    break;
                    // every line of text received from the client ends with \r\n
                    if (c == '\n') {
                        // last character on line of received text
                        // starting new line with next character read
                        currentLineIsBlank = true;
                    }
                    else if (c != '\r') {
                        // a text character was received from client
                        currentLineIsBlank = false;
                    }
                } // end if (client.available())
            } // end while (client.connected())
            delay(1);  // give the web browser time to receive the data
            client.stop(); // close the connection
        } // end if (client)
}
Goals

- Review
  - Programming in Sketch
- Communication
  - SPI, I2C
- Data logging
  - SD cards
- Accessing the Linux host
  - Shells
  - GPIO
- Networking
  - LAN
  - WiFi
- Network Communication
  - REST
  - WebSockets
  - WebRTC
- Cloud
  - Data sharing
Web Communication

- **REST**
  - Client/server
  - Make a request, receive a response
  - Browser can make requests, but cannot receive a request
  - Server can send and receive requests
  - Heavyweight

- **Web socket**
  - Bidirectional channel between server or browser
  - One endpoint must be server
  - Persistent connection
  - Lightweight communication

- **WebRTC data**
  - Bidirectional
  - Peer to Peer
    - No restrictions on endpoints
    - No restrictions on network
  - Video, audio, data
Web Communication

- **REST**
  - Client/server
  - Make a request, receive a response
  - Browser can make requests, but cannot receive a request
  - Server can send and receive requests
  - Heavyweight

- **Web socket**
  - Bidirectional channel between server or browser
  - One endpoint must be server
  - Persistent connection
  - Lightweight communication

- **WebRTC data**
  - Bidirectional
  - Peer to Peer
    - No restrictions on endpoints
    - No restrictions on network
  - Video, audio, data
Anatomy of an HTTP Transaction

```
unix> telnet www.aol.com 80
Trying 205.188.146.23...
Connected to aol.com.
Escape character is '^[].
GET / HTTP/1.1
host: www.aol.com

HTTP/1.0 200 OK
MIME-Version: 1.0
Date: Mon, 08 Jan 2001 04:59:42 GMT
Server: NaviServer/2.0 AOLserver/2.3.3
Content-Type: text/html
Content-Length: 42092

<html>
...
</html>
Connection closed by foreign host.
unix>
```

Client: open connection to server
Telnet prints 3 lines to the terminal

Client: request line
Client: required HTTP/1.1 HOST header
Client: empty line terminates headers.

Server: response line
Server: followed by five response headers

Server: expect HTML in the response body
Server: expect 42,092 bytes in the resp body
Server: empty line ("\r\n") terminates hdrs
Server: first HTML line in response body
Server: 766 lines of HTML not shown.
Server: last HTML line in response body
Server: closes connection
Client: closes connection and terminates
HTTP Requests

• HTTP request is a **request line**, followed by zero or more **request headers**

• **Request line:** `<method> <uri> <version>`
  • `<version>` is HTTP version of request (**HTTP/1.0** or **HTTP/1.1**)
  • `<uri>` is typically URL for proxies, URL suffix for servers.
  • `<method>` is either **GET**, **POST**, **OPTIONS**, **HEAD**, **PUT**, **DELETE**, or **TRACE**.
HTTP Requests (cont)

- **HTTP methods**
  - **GET**: Retrieve static or dynamic content
    - Arguments for dynamic content are in URI
    - Workhorse method (99% of requests)
  - **POST**: Retrieve dynamic content
    - Arguments for dynamic content are in the request body
  - **OPTIONS**: Get server or file attributes
  - **HEAD**: Like **GET** but no data in response body
  - **PUT**: Write a file to the server!
  - **DELETE**: Delete a file on the server!
  - **TRACE**: Echo request in response body
    - Useful for debugging
HTTP Responses

- HTTP response is a **response line** followed by zero or more **response headers**.

- Response line:

  \[
  \text{<version>} \ \text{<status code>} \ \text{<status msg>}
  \]

  - `<version>` is HTTP version of the response.
  - `<status code>` is numeric status.
  - `<status msg>` is corresponding English text.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK Request was handled without error</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden Server lacks permission to access file</td>
</tr>
<tr>
<td>404</td>
<td>Not found Server couldn’t find the file.</td>
</tr>
</tbody>
</table>

- **Response headers**:

  - `<header name>`: `<header data>`
    - Provide additional information about response

  **Content-Type**: MIME type of content in response body.
  **Content-Length**: Length of content in response body.
GET Request to Apache Server From IE Browser

GET /test.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)
Host: euro.ecom.cmu.edu
Connection: Keep-Alive
CRLF (\r\n)
GET Response From Apache Server

HTTP/1.1 200 OK
Date: Thu, 22 Jul 1999 04:02:15 GMT
Server: Apache/1.3.3 Ben-SSL/1.28 (Unix)
Last-Modified: Thu, 22 Jul 1999 03:33:21 GMT
ETag: "48bb2-4f-37969101"
Accept-Ranges: bytes
Content-Length: 79
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

</html>
<head><title>Test page</title></head>
<body>
<h1>Test page</h1>
</body>
*BAD* GET Request to Apache Server From IE Browser

GET /adduser?name=Robert HTTP/1.1

vs.

GET /test.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (com
Host: euro.ecom.cmu.edu
Representational State Transfer (REST)

- A style of software architecture for distributed hypermedia systems such as the World Wide Web

- Introduced in the doctoral dissertation of Roy Fielding
  - One of the principal authors of the HTTP specification

- A collection of network architecture principles which outline how resources are defined and addressed
Which Do You Think is Better?
REST - not a Standard

- REST is not a standard
  - JSR 311: JAX-RS: The JavaTM API for RESTful Web Services

- But it uses several standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/etc (Resource Representations)
  - text/xml, text/html, image/gif, image/jpeg, etc (Resource Types, MIME Types)
Main Concepts

Nouns (Resources)
unconstrained
i.e., http://example.com/employees/12345

Verbs
constrained
i.e., GET

Representations
constrained
i.e., XML
Naming Resources

- REST uses URI to identify resources

  - http://localhost/books/
  - http://localhost/classes
  - http://localhost/classes/cs2650
  - http://localhost/classes/cs2650/students

- As you traverse the path from more generic to more specific, you are navigating the data
Verbs

- Represent the actions to be performed on resources
- HTTP GET
- HTTP POST
- HTTP PUT
- HTTP DELETE
Representations

• XML

  • <COURSE>
    • <ID>CS2650</ID>
    • <NAME>Distributed Multimedia Software</NAME>
  • </COURSE>

• JSON

  • {course
    • {id: CS2650}
    • {name: Distributed Multimedia Software}
  • }
Why is it called REST? (i.e., "Representational State Transfer")

The Client references a Web resource using a URL. A representation of the resource is returned (in this case as an HTML document). The representation (e.g., Boeing747.html) places the client application in a state. The result of the client traversing a hyperlink in Boeing747.html is another resource accessed. The new representation places the client application into yet another state. Thus, the client application changes (transfers) state with each resource representation --> Representation State Transfer!
Architecture Style
Real Life Examples

- Google Maps
- Google AJAX Search API
- Yahoo Search API
- Amazon WebServices
RESTful but Heavyweight

- **HTTP is half-duplex**
  - Designed for document transmission
  - Request/Response mechanism

- **Not real-time**
  - **Polling (AJAX)**
    - Poll server for updates, wait at client
  - **Long Polling (Comet)**
    - Poll server for update, wait at server; uses two connections and is complex

- **Complex, Inefficient, Wasteful**
Web Communication

- **REST**
  - Client/server
  - Make a request, receive a response
  - Browser can make requests, but cannot receive a request
  - Server can send and receive requests
  - Heavyweight

- **Web socket**
  - Bidirectional channel between server or browser
  - One endpoint must be server
  - Persistent connection
  - Lightweight communication

- **WebRTC data**
  - Bidirectional
  - Peer to Peer
    - No restrictions on endpoints
    - No restrictions on network
  - Video, audio, data
What are WebSockets?

- Bidirectional, full-duplex, permanent TCP connections
  - 400 times less overhead
  - 1/3 of HTTP latency

- Standardized in HTML5 by W3C and IETF
  - Protects investments

- Single TCP port
  - saves 50% server resources

- Designed for interactive Web Applications

- Open for all kind of stationary and mobile browser based and native apps

- Not just a protocol, but a new paradigm
  - Request/Response Real-Time Comm.
WebSocket versus Polling

(WebSockets diagram with comparison to polling)
WebSocket API (in JavaScript)

```javascript
var websocket = new WebSocket("ws://www.host.com/path");
websocket.onopen = function(evt) { onOpen(evt) };
websocket.onclose = function(evt) { onClose(evt) };
websocket.onmessage = function(evt) { onMessage(evt) };
websocket.onerror = function(evt) { onError(evt) };

function onMessage(evt) { alert( evt.data); }
function onError(evt) { alert( evt.data); }

websocket.send("client to server");
```
Web Communication

- REST
  - Client/server
  - Make a request, receive a response
  - Browser can make requests, but cannot receive a request
  - Server can send and receive requests
  - Heavyweight

- Web socket
  - Bidirectional channel between server or browser
  - One endpoint must be server
  - Persistent connection
  - Lightweight communication

- WebRTC data
  - Bidirectional
  - Peer to Peer
    - No restrictions on endpoints
    - No restrictions on network
  - Video, audio, data
WebRTC

• Media stack: audio & video

• Real time communication
  • Audio, video, data

• Peer-to-peer

• Accessible from browser: easy & available
The WebRTC API Stack Overview
Three Main Features

• Media Stream
  • WebRTC allow you to create a media stream + audio stream from the user built-in webcam and microphone (if they have one)
  • Represent a media source that is containing one or more synchronized media stream tracks (e.g., camera and microphone)

• Peer Connection
  • WebRTC can establish a direct P2P connection between 2 clients
    • With little setup, can exchange raw data directly between users
    • Helpful when exchanging a lot of data or even small amount fast

• Data channel
  • Data channel enable you to create a virtual server and do what you used to do with Web sockets without server
    • no longer have to send data to server and then to clients again.
Media Streams

• Represent a media source that is containing one or more synchronized media stream tracks
  • e.g. stream from camera and microphone has synchronized video and audio tracks

• Media stream can be converted to an object URL, and passed to HTML5 <video> element

• Use the getUserMedia API to get a media stream/mic
Demo: Connect Camera to Local Browser Video Display

http://www.simpl.info/getusermedia/
Demo: Process Camera Data

http://idevelop.ro/ascii-camera/
Demo: Process Camera Data

http://shinydemos.com/facekat/
Demo: Data Channel

- Demo: http://www.simpl.info/rtcdatachannel/
API

- Set up peer connection
  - RTCPeerConnection

- Access local camera/audio
  - getUserMedia

- Add data channel
  - RTCDataChannel
WebRTC and IoT

- How could it be used in your project?
- IoT connects to physical world through sensors
- Audio/Video
- Peer to Peer
Goals

- Review
  - Programming in Sketch
- Communication
  - SPI, I2C
- Data logging
  - SD cards
- Accessing the Linux host
  - Shells
  - GPIO
- Networking
  - LAN
  - WiFi
- Network Communication
  - REST
  - WebSockets
  - WebRTC
- Cloud
  - Data sharing
Connect to the Cloud!

```
#include <SPI.h>
#include <WiFi.h>

void setup() {
    // initialize serial and wait for the port to open:
    Serial.begin(9600);
    while(!Serial) {
        // attempt to connect using WESP encryption:
        Serial.println("Initializing Wifi...");
        printMacAddress();
        // scan for existing networks:
        Serial.println("Scanning available networks...");
        listNetworks();
    }
}

void loop() {
    delay(10000);
    // scan for existing networks:
    Serial.println("Scanning available networks...");
    listNetworks();
}

void printMacAddress() {
    // the MAC address of your Wifi shield
    byte mac[6];
    // print your MAC address:
    WiFi.macAddress(mac);
    Serial.print("MAC: ");
    Serial.print(mac[5], HEX);
    Serial.print(”: “);
    Serial.print(mac[4], HEX);
    Serial.print(”: “);
    Serial.print(mac[3], HEX);
    Serial.print(”: “);
    Serial.print(mac[2], HEX);
    Serial.print(“: “);
    Serial.print(mac[1], HEX);
    Serial.println("\n");
```
Xively Cloud Services

- Supports hundreds of platforms, millions of gateways and a crazy number of smart devices

- Comprehensive and secure infrastructure services

- Online development tools and dev center
Xively

- Feed ID
- Channels
- API Keys
- Triggers
Connect an Arduino to Xively
Hooking up a Photocell
... // assume global variables here and setup() for Ethernet (as we saw earlier)

void loop() {
    ...

    // if you're not connected, and ten seconds have passed since
    // your last connection, then connect again and send data:
    if(!client.connected() && (millis() - lastConnectionTime > postingInterval)) {

        sendData(sensorReading);

    }
}

...
// this method makes a HTTP connection to the server:
void sendData(int thisData) {
    // if there's a successful connection:
    if (client.connect(server, 80)) {
        Serial.println("connecting...");

        // send the HTTP PUT request:
        client.print("PUT /v2/feeds/");
        client.print(FEEDID);
        client.println(".csv HTTP/1.1");
        client.println("Host: api.xively.com");
        client.print("X-XivelyApiKey: ");
        client.println(APIKEY);
        client.print("User-Agent: ");
        client.println(USERAGENT);
        client.print("Content-Length: ");
        int thisLength = 8 + getLength(thisData);
        client.println(thisLength);

        // calculate the length of the sensor reading in bytes:
        // 8 bytes for "sensor1," + number of digits of the data:
        client.println("Content-Type: text/csv");
        client.println("Connection: close");
        client.println();

        // here's the actual content of the PUT request:
        client.print("sensor1, ");
        client.println(thisData);
    }
    else {
        // if you couldn't make a connection:
        Serial.println("connection failed");
        Serial.println();
        Serial.println("disconnecting.");
        client.stop();
    }
    // note the time that the connection was made or attempted:
    lastConnectionTime = millis();
}
#include <SPI.h>
#include <WiFi.h>
#include <HttpClient.h>
#include <Xively.h>

char ssid[] = "SSID_HERE"; // your network SSID
char pass[] = "PASS_HERE"; // your network password

int status = WL_IDLE_STATUS;

char xivelyKey[] = "API_KEY_HERE";
#define xivelyFeed FEED_ID_HERE
char sensorID[] = "LIGHT_SENSOR_CHANNEL";
char ledID[] = "LED_CHANNEL";

// Analog pin which we're monitoring
#define sensorPin A2
//led connected pin
#define ledPin 9

// Define the strings for our datastream IDs
XivelyDatastream datastreams[] = {
    XivelyDatastream(sensorID, strlen(sensorID), DATASTREAM_FLOAT),
    XivelyDatastream(ledID, strlen(ledID), DATASTREAM_FLOAT),
};

// Finally, wrap the datastreams into a feed
XivelyFeed feed(xivelyFeed, datastreams, 2 /* # datastreams */);

WiFiClient client;
XivelyClient xivelyclient(client);

see next slide for cont.
```c
#include <SPI.h>
#include <WiFi.h>
#include <HttpClient.h>
#include <Xively.h>

char ssid[] = "SSID_HERE";  // your network SSID
char pass[] = "PASS_HERE";  // your network password
int status = WL_IDLE_STATUS;

char xivelyKey[] = "API_KEY_HERE";
#define xivelyFeed FEED_ID_HERE
char sensorID[] = "LIGHT_SENSOR_CHANNEL";
char ledID[] = "LED_CHANNEL";

// Analog pin which we're monitoring
#define sensorPin A2
//led connected pin
#define ledPin 9

// Define the strings for our datastream IDs
XivelyDatastream datastreams[] = {
    XivelyDatastream(sensorID, strlen(sensorID), DATASTREAM_FLOAT),
    XivelyDatastream(ledID, strlen(ledID), DATASTREAM_FLOAT),
};
// Finally, wrap the datastreams into a feed
XivelyFeed feed(xivelyFeed, datastreams, 2 /* # datastreams */);

WiFiClient client;
XivelyClient xivelyclient(client);

void loop() {
    //adjust LED level. set from Xively
    int getReturn = xivelyclient.get(feed, xivelyKey);
    if(getReturn > 0){
        Serial.println("LED Datastream");
        Serial.println(feed[1]);
    } else Serial.println("HTTP Error");

    //write value to LED - change brightness
    int level = feed[1].getFloat();
    if(level < 0){
        level = 0;
    }else if(level > 255){
        level = 255;
    }
    //actually write the value
    digitalWrite(ledPin, level);

    //read sensor values
    int sensorValue = analogRead(sensorPin);
datastreams[0].setFloat(sensorValue);
    //print the sensor value
    Serial.print("Read sensor value ");
    Serial.println(datastreams[0].getFloat());
    Serial.println("Uploading it to Xively");
    int ret = xivelyclient.put(feed, xivelyKey);
    //return message
    Serial.println("xivelyclient.put returned ");
    Serial.println(ret);
    Serial.println("");

    //delay between calls
    delay(15000);
}
```
• Automatically generates code

• Choreos
  • Choreos streamline processes for everyday programming tasks

• Lots of “choreos”
  • 2,000+
```python
from temboo.Library.XivelyReadWriteData import WriteData
from temboo.core.session import TembooSession

# Create a session with your Temboo account details
session = TembooSession("jveejay", "myFirstApp", "wGAzEWT5DA2bkFWc8XRFGg4sIf8L")

# Instantiate the Choreo
writeDataChoreo = WriteData(session)

# Get an InputSet object for the Choreo
writeDataInputs = writeDataChoreo.new_input_set()

# Execute the Choreo
writeDataResults = writeDataChoreo.execute_with_result(writeDataInputs)
```

```php
// Instantiate the Choreo, using a previously instantiated Temboo_Session object, eg:
// $session = new Temboo_Session('jveejay', 'myFirstApp', 'wGAzEWT5DA2bkFWc8XRFGg4sIf8L');

$writeData = new Xively_ReadWriteData_WriteData($session);

// Get an input object for the Choreo
$writeDataInputs = $writeData->newInputs();

// Execute Choreo and get results
$writeDataResults = $writeData->execute($writeDataInputs)->getResult();
```
Record a sound sensor value to a Google Spreadsheet

First, when should the sensor value be added to the spreadsheet?
If the sensor value is **0** on Pin **A0**

Now what's your Google account?
**Vijay**

And the name of your spreadsheet?
**Test**

NOTE: The spreadsheet must have at least one named column, like **this one**.

Also, do you want to trigger something on the Arduino board too?
Write **HIGH** to Pin **6**
Write **HIGH** to Pin **2**

How does your board connect to the internet?
**CC3000**

ARUINO SKETCH  Download
```c
#include <SPI.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include "TembooAccount.h" // Contains Temboo account information
```
```java
void runAppendRow(int sensorValue) {
    TembooChoreo AppendRowChoreo(client);

    // Set Temboo account credentials
    AppendRowChoreo.setAccountName(TEMBOOACCOUNT);
    AppendRowChoreo.setAppName(TEMBOOAPPKEYNAME);
    AppendRowChoreo.setAppKey(TEMBOOAPPKEY);

    // Set profile to use for execution
    AppendRowChoreo.setProfile("arduinoGenericSpreadsheet");

    // Set Choreo inputs
    String RowDataValue = String(sensorValue);
    AppendRowChoreo.addInput("RowData", RowDataValue);

    // Identify the Choreo to run
    AppendRowChoreo.setChoreo("/Library/Google/Spreadsheets/AppendRow'\n
    // Run the Choreo
    unsigned int returnCode = AppendRowChoreo.run();

    // A return code of zero means everything worked
    if (returnCode == 0) {
        Serial.println("\nTriggered! Calling /Library");
        Serial.println("Done!\n");
    } else {
        // A non-zero return code means there was an error
        // Read and print the error message
        while (AppendRowChoreo.available()) {
            char c = AppendRowChoreo.read();
            Serial.println(c);
        }
    }
}
```
“Find it. Try it. Use it.”

Welcome to Temboo Support

Try out our many tutorials and examples