CHAPTER XX

Virtual World Objects and Arduino Devices with Unity 4.0

Unity is game-building framework that allows a single code base to make 3D interfaces for the web, Mac OS X, Windows, iPhone, Android, and game consoles. One of its strengths is the ease with which it can incorporate content from multiple game design tools. This chapter will explore the use of Arduino with data and visualization. First, it will focus on using serial data from the Arduino to interact with a sample environment. Then it will use the same Arduino technology to create multiple networkable Arduino data inputs into one Unity project using Open Sound Control over UDP (User Datagram Protocol). It will use Node.js to gather serial data and send OSC events from the Arduino to Unity.

Virtual world objects are defined as objects that can correspond to real-world objects and exchange information with virtual objects; the game creates a virtual space for these objects. For example, a lamp can be turned off in the real world and changed in the virtual space. You can use digital information like on and off, as well as analog data that represents a range, such as the intensity of light. Information gathered from an RFID reader can be used to set multiple settings in the virtual world space.

This chapter will introduce you to an example project called Box Bomber, which can be a starter for creating a basic project that can be evolved into many different projects. You will use the Arduino to build a game controller that provides a joystick, an accelerometer, potentiometers, and buttons to control in-world objects, physics, and the world plane. Additionally, you will use a breadboarded circuit and a custom PCB version of the controller, of which the Pro Arduino site will include a 3D-printable case.

Key Technologies

This chapter introduces several key technologies that make it possible to connect an Arduino with a game engine like Unity. The most logical way to make this connection is using serial communications, and the next step would be to find a way to network the communications so more than one device can play a game at the same time. Once you have either of these options available, you have a complete toolbox of options.

In order to reduce the complexity of programming the Arduino, it helps to have a common data format that you can use over the network or serial connection. You'll use Java Serial Object Notation (JSON) for that. In the general case, using serial communications, you can talk directly to the hardware from inside Unity. In the second solution, using network communications, you need to have something that will translate the output from the Arduino into serial data. For this you will use a short script written in Node.js. This is an open source, multiplatform JavaScript language you can use to talk to the Arduino serial port and then convert the data to send over the network.

Serial Communications

A common way of talking with Arduino is via serial communications. There are three common ways of getting serial data into Unity. Since Unity is written in Mono, which is an open source version of .NET from Microsoft, Mono can directly talk to a serial port. One version of this example will talk directly to a serial interface. This is ideal for creating an ad hoc controller and connecting it to your Unity game or world. However, there is an outstanding issue with Unity's version of Mono that does not allow Mac OS X to communicate with the serial port. So, for now, this example will exclude Mac OS X compatibility.

Network Communications

Using a network protocol like UDP is an excellent workaround that can be used with multiple operating systems and platforms across the Internet. The issue, however, is that you would have to invent your own communication standard. So instead, you will use the Open Sound Control (OSC) protocol, which is a multimedia communication standard. It is designed to be a reliable method for sending messages across multimedia devices within a network.

JavaScript Object Notation Format

Regardless of how you send the data, you need a simple and easy way to interface with the sending mechanism. You will be doing this exclusively with JSON. This is a technique used to produce transactions in a form that humans can read and understand, but that are generated by machines and parsed with little effort. This allows you to write the same Arduino code when you use serial and network communications.

Listing XX-1 shows two data chunks. The first row between the curly braces ({ and }) shows a controller ID of 110, with a `btn1` button 1 press value of 0. The next row shows that the controller 110 button is now pressed, with a value of 1.

Listing XX-1. *JSON Data Format*

```
{
    "controllerid": 110,
    "btn1": 0
}
{
    "controllerid": 110,
    "btn1": 1
}
```

The Node.js Programming Language

Node.js is a JavaScript-based programming language that uses the Google V8 JavaScript engine to run JavaScript as a programming language on a computer supported by Windows, Mac OS X, or Linux. It is fast and has the library support necessary for serial and OSC communications. With a few short lines of code, you will be able to capture the serial data, send it from Arduino, and enable it for use over the network with Unity.

Getting Unity

Unity software needs to be downloaded and installed. It will work on Mac OS X and Windows. The download for Unity can be found at *http://Unity.com/unity/download/*.

The examples here will be configured to work with versions 3.5 and 4.0 of the Unity software. Once you've installed and configured Unity, you can move on to the example.

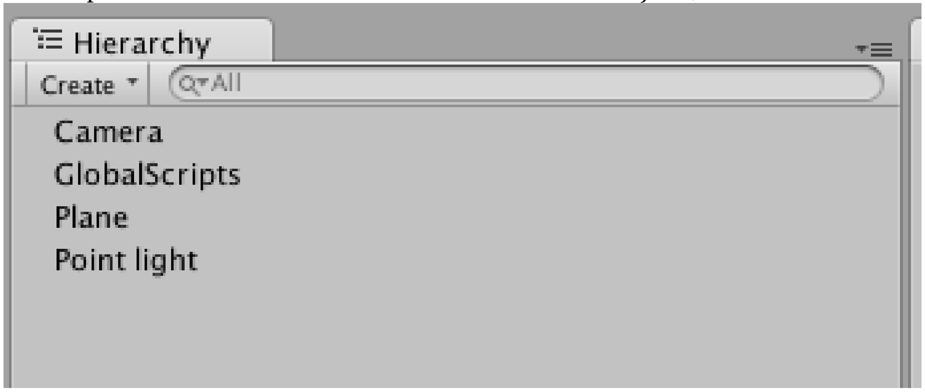
Unity Project: Box Bomber

The Box Bomber example comes as a complete Unity project that you can load directly into Unity and get started with quickly. You can modify it and build your projects from it too (see Figure XX-1).

This project is built on a flat plane—there is a series of invisible boxes in a grid that can be stacked on red boxes. These boxes are stacked when the player presses the main button on the Arduino control. The boxes are then placed wherever there are x,y-coordinates. As the boxes are stacked, the second button is used to drop a spherical green bomb that crashes into the boxes. Lastly, the accelerometer provides additional forces on the boxes to push them around.

At the beginning, this example has three modes to select from. Mode 1 is Mouse mode, which can be used to see how a normal input device works and check the default behaviors. Mode 2 is Serial Data mode, which reads the JSON data from the Arduino control. Mode 3 is the OSC version, which accepts data from OSC clients. Each version will be described following.

A Unity project consists of scenes. *Scenes* are the game views that organize your games into sections, like chapters of a book. Each scene has a common set of objects, which will be discussed shortly.



Camera
GlobalScripts
Plane
Point light

Figure XX-1. *The Camera, GlobalScripts, Plane, and Point Light.*

Loader Scene

The main scene for the application is the Loader scene. It will load the three different example play modes: Serial Data, OSC, and Mouse (Figure XX-2).

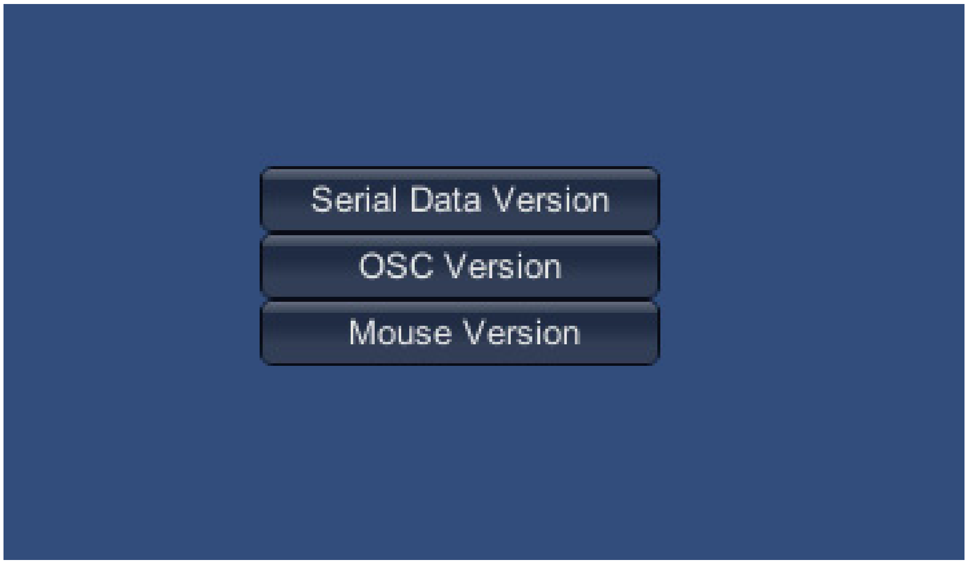


Figure XX-2. *Three different play modes*

In this case, you define three scenes that can be played in three different game modes. One is with a mouse, the other uses serial data from the Arduino, and the last uses the OSC network data. This example can be used as a premade starter for other projects. In this way, you can develop many variations on the presented theme.

Mouse Scene

The Mouse scene behavior is controlled by left-clicking the plane. Blocks will stack wherever the pointer is red. Then, you can right-click, hold, and then release. A green bomb will drop from the sky, and by turning the physics handler on, it will crush the blocks below. You can apply force to the blocks by using the W, A, S, and D keys on the keyboard. Camera controls are not enabled.

Serial Data Scene

The Serial Data scene will use the joystick to move the cursor over the grid, and red blocks will stack when button 1 is pressed. At any time, button 2 can be held, which will drop a bomb on the stack of red blocks. Camera controls are handled by the potentiometer for x and y, which can be turned to fine-tune the view plane, The accelerometer controls the force applied to the blocks, which can be pushed in the x, y, and z directions.

OSC Scene

This behaves identically to the Serial Data scene, except that data is sent via OSC and the identified controller is used. The Pro Arduino site shows an example of two controllers being used. The controller ID is used to keep the data between the two distinct.

As mentioned earlier, each scene has a comment set of objects, which I'll review now.

The Prefabs

Prefabs are reusable game objects that one can easily set for common properties that identify where the object is used in the game. Figure XX-3 shows several examples of prefabs.

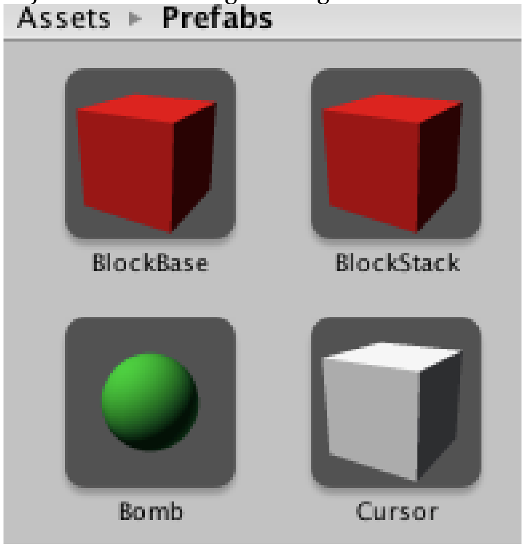


Figure XX-3. *Prefabs, which are the basic game objects*

In this case, the Cursor prefab indicates where the pointer is located on the field. The BlockBase is the first block upon which another can be stacked. The BlockStacks are the blocks, which pile, or stack, on top of each other. The Bomb is the object that comes crashing down after the user right-clicks or presses the secondary button. Each prefab has set default properties that can be customized as needed. For instance, you can change the mass of the bomb to vary its striking power.

This project also has a basic set of scripts that manage the behavior, procedural settings, and generation of the objects, as needed by the controls. These scripts are

- `SceneSelect.cs`: This selects the scene: Mouse, Serial Data, or OSC.
- `GenerateGrid.cs`: This generates the virtual grid for each scene above the field of play.
- `SpawnBlock.cs`: This controls the spawning of the blocks.
- `PushMe.cs`: This is where the push code is defined. For instance, the accelerometer data will the direction and force where the push occurs.

- `DaBomb.cs`: This spawns a Bomb prefab at the determined location and size.
- `StoreHeight.cs`: This tracks the stack heights.
- `InputManager.cs`: The Input Manager indicates the kind of input. If it's the Mouse scene, then the mouse is selected for input. If it is the Serial Data scene, then Unity talks directly to the controller's serial port. If it is the OSC scene, it lives for the incoming network data.

Enabling the Building of the Project and Access to Serial Data

Unity is written in Mono, which is an open source implementation of .NET. Unity utilizes both JavaScript and C#. However, Unity defaults to a subset of .NET for the projects to work, Unity must be enabled to have access to `System.IO.Ports`, which is in the full Mono package. This must be enabled in the project, or there will be an error that reporting that `System.IO.Ports` cannot be found.

Step one is to select the Build Settings option, as shown in Figure XX-4. This brings up the Player Settings menu.

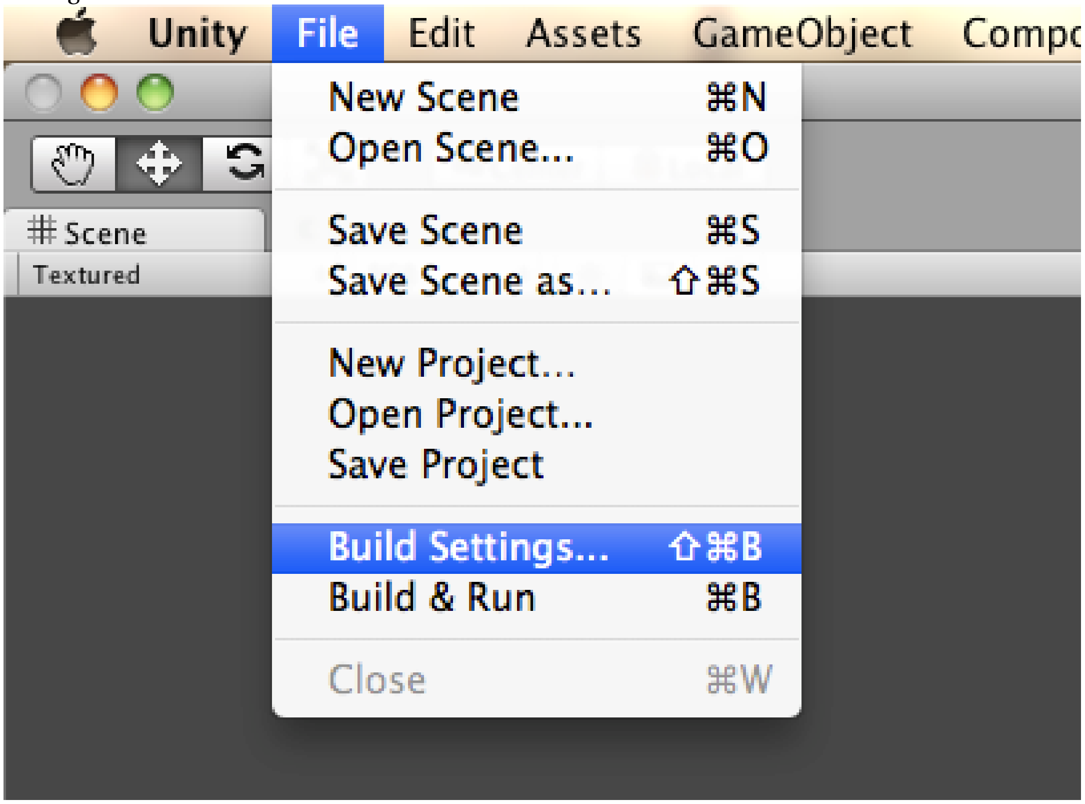


Figure XX-4. *Select Build Settings to enable the full .NET features.*

Select the player settings, as shown in Figure XX-5. Then the .NET optimization options appear.

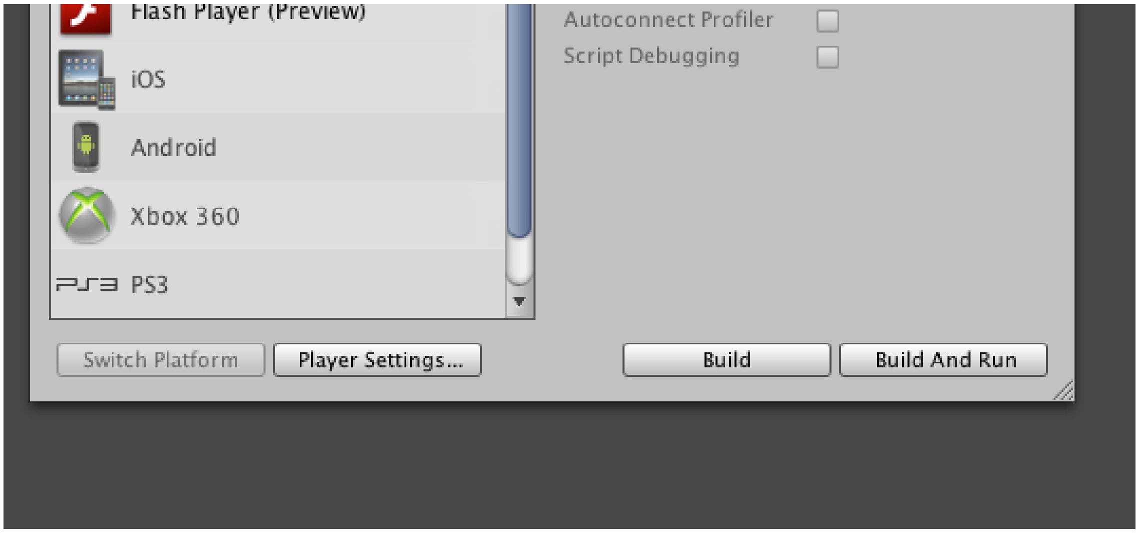


Figure XX-5. Select the player settings.

From the options shown in Figure XX-6, select the full .NET 2.0 options.

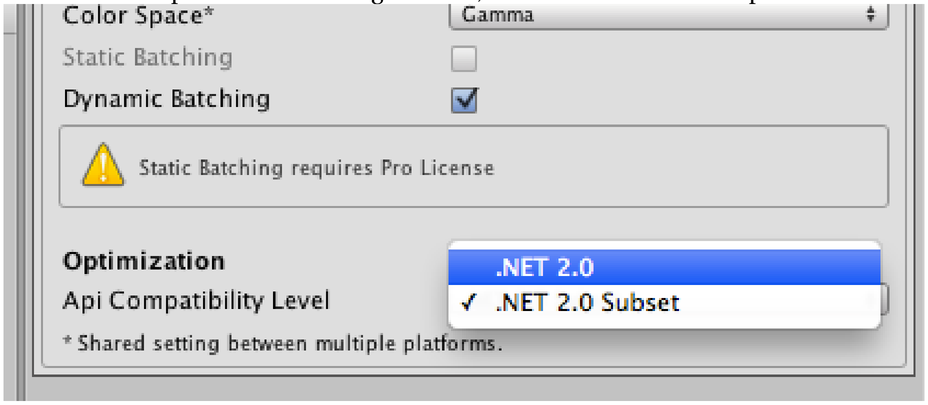

Color Space* Gamma
Static Batching ☐
Dynamic Batching ✓
Static Batching requires Pro License
Optimization
Api Compatibility Level .NET 2.0 / ✓ .NET 2.0 Subset
* Shared setting between multiple platforms.

Figure XX-6. Select the .NET 2.0 option.

Figure XX-7 shows a correctly set optimization level. Once you set the optimization level, you can perform serial communications with Windows and Mac OS X (though Mac OS X requires some additional configuration).

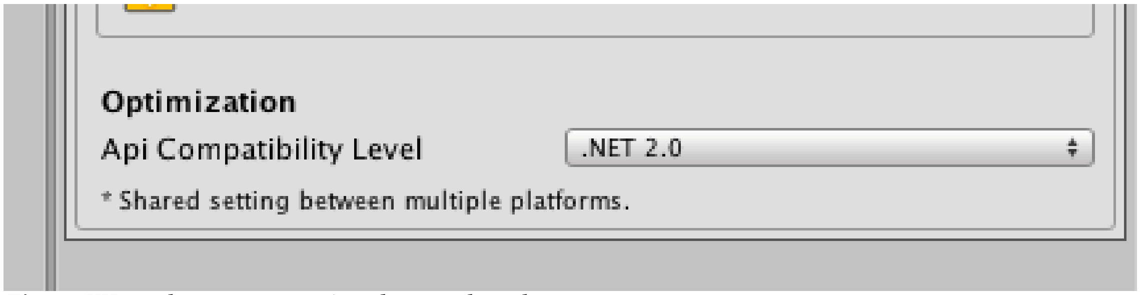

Optimization
Api Compatibility Level .NET 2.0
* Shared setting between multiple platforms.

Figure XX-7. The .NET 2.0 option shown selected

The Project Skeleton

Add intro text...

1. Build the Arduino controller.
2. Write an Arduino sketch that reads the data from the attached sensors.
3. Create a Node.js script that reads the serial number.
4. Create a Node.js script that reads the serial number and sends the information to OSC using node-SC.
5. Use the sample Unity project for the chapter.
    - Include JSON.
    - Include UnityOSC.
6. Update the input script in the project.

Digital and Analog Data Exchange with Unity

In this section, the controller you will use will collect real-world data from the user via a set of sensors like joysticks and potentiometers. These provide analog data that you will convert into position and camera information. This type of analog data needs to be shown moving smoothly and in time with its usage. Additionally, you will work with an accelerometer to determine tilt and shaking conditions. You will also be using puts to trigger a series of on-and-off digital values for buttons (e.g., pressing down to place a block or tapping to drop a bomb). This data will be encoded from the sensors by the Arduino into formatted serial data that will be sent to Unity to affect game play.

Building the Controller

The controller for this will be either the Arduino Leonardo, the Teensy 2.0, or the chipKIT Uno32, each of which has enough analog pins to support the amount of analog-to-digital conversion that you will be performing.

The parts list is shown in Table 12-1.

Table 12-1. Controller Parts List

Amount	Part Type
1	Teensy 2.0
1	Thumb joystick from SparkFun
1	Triple-axis accelerometer breakout (ADXL335)
2	10kΩ resistor
2	Push button
2	Rotary potentiometer (small)

Figure XX-8 shows the controller wired up for use with the Teensy 2.0.

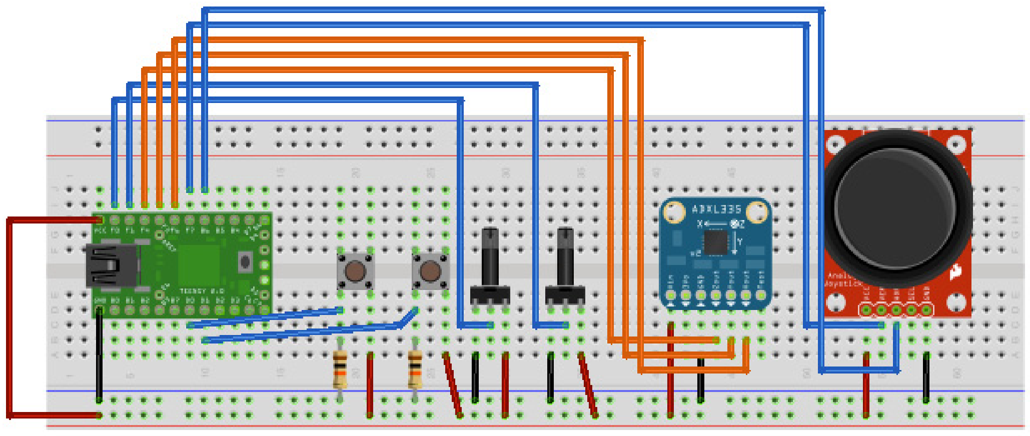


Figure XX-8. Breadboard project layout

Programming the Controller

The Arduino sketch for this is quite short, and all the buttons are analog devices. For the best results, the SparkFun joystick requires a 5V power supply. The AXL335 is 3.3V, but some breakout boards allow for a 5V connection. The accelerometer is the only device that needs calibration. There is a bit of mathematical computation and trial and error required to convert the analog x, y, and z data to angular x, y, and z data. Note that you can use the same Arduino sketch for the serial and OSC projects.

■ **Note** The accelerometer generates a set of vector data in the x-, y-, and z-axes. The Pro Arduino web site provides a calibration script to find the maximum and minimum values generated by the accelerometer.

The rest of the code ensures that the data from the controller is read fast enough that the game environment feels responsive. For these reasons, a baud rate of 57,600 was selected.

The last part includes reading and mapping the data, and then printing it into JSON format.

The code in Listing XX-2 defines the hardware pins, configures the Arduino to talk with the hardware, reads the values as quickly as possible in the loop code, and then outputs the result over the serial port.

Listing XX-2. Arduino Code That Captures the Analog and Digital, and Converts It to JSON Serial

```
#define pot1  A6
#define pot2  A5
#define pot3  A1
#define pot4  A0
#define but1  5
#define but2  6
#define accelX A4
#define accelY A3
#define accelZ A1


//Set the controller ID
const int controlID = 1;
// calibration result:  X(424,611)  Y(403,619)  Z(398,633
// Accelerometer... axis(min,max):    X(403,630)  Y(404,628)  Z(415,673
const int minX = 397;
const int maxX = 630;
const int minY = 403;
const int maxY = 653;
const int minZ = 415;
const int maxZ = 806;

double xx;
double yy;
double zz;

double getAngle(int angA, int angB)
{
  double dd = RAD_TO_DEG * (atan2(-angA, -angB) + PI);
  return dd;
}


void setup()
{
  Serial.begin(57600);
  pinMode(but1, INPUT);
  pinMode(but2, INPUT);

}

void loop()
{
  int joyX = analogRead(pot1);
  int joyY = analogRead(pot2);
  int camX = analogRead(pot3);
  int camY = analogRead(pot4);
  int accelXval = analogRead(accelX);
  int accelYval = analogRead(accelY);
  int accelZval = analogRead(accelZ);
  int but1val = digitalRead(but1);
  int but2val = digitalRead(but2);

  int angX = map(accelXval, minX, maxX, -90, 90);
  int angY = map(accelYval, minY, maxY, -90, 90);
  int angZ = map(accelZval, minZ, maxZ, -90, 90);

  xx = getAngle(angY, angZ);
  yy = getAngle(angX, angZ);
  zz = getAngle(angY, angX);


  Serial.print("{");
  Serial.print("\"controlID\" : ");
  Serial.print(controlID);
  Serial.print(", \"joyX\" : ");
  Serial.print(joyX);
  Serial.print(", \"joyY\" : ");
  Serial.print(joyY);
  Serial.print(", \"camX\" : ");
  Serial.print(camX);
  Serial.print(", \"camY\" : ");
  Serial.print(camY);
  Serial.print(", \"but1\" : ");
  Serial.print(but1val);
  Serial.print(", \"but2\" : ");
  Serial.print(but2val);

  Serial.print(", \"accelX\" : ");
  Serial.print(accelXval);
  Serial.print(", \"accelY\" : ");
  Serial.print(accelYval);
  Serial.print(", \"accelZ\" : ");
  Serial.print(accelZval);

  Serial.print(", \"angX\" : ");
  Serial.print(angX);
  Serial.print(", \"angY\" : ");
  Serial.print(angY);
  Serial.print(", \"angZ\" : ");
  Serial.print(angZ);

Serial.println("}");
```

```
  delay(20);

}
```

Reading Data from the Controller over Serial

In order to read the data, you need a simple and efficient way to get the data from the Arduino and allow it to interact with the Internet of Things. The JSON data from the Arduino can be routed to multiple web-friendly networks. In this example, you are going to use Node.js.

Installation

The home page for Node.js is `http://nodejs.org`, and the download link is directly on the main page. It is a standard install, so download using the standard steps.

Basic Usage

You can run node from the command line either in interpreted mode or via command execution. Here's a basic Hello World command:

```
console.log("Hello World!");
```

To execute the `helloworld.js`, use this code:

```
node helloworld.js
```

The result is the following:

```
Hello World!
```

You will use this technique to execute the code. Now you can install all the modules needed to communicate with the serial port, and you can use OSC to talk with Unity.

```
Npm install serialport
Npm install node-osc
```

Run the code in Listing XX-3, which will read data from the serial port.

Listing XX-3. Reading Serial Data from the Arduino Using Node.js

```
var SerialPort, count, serialport, sp, sys, tty;
  serialport = require("serialport");
  SerialPort = serialport.SerialPort;
  sys = require("sys");

  count = 0;
  tty = "/dev/tty.usbserial-A1009UMS";

  sp = new SerialPort(tty, {
    parser: serialport.parsers.readline("\n"),
    baudrate: 57600
  });
  sp.on("data", function(data) {
    return console.log("Count: " + count++ + " Data: " + data);
  });

  sp.on("error", function(error) {
    return console.log("Error: " + error);
  });
```

Once you have assigned the correct serial device path or COM port to the `tty` variable, it will read from your serial port Mac OS X, Linux, and/or Windows.

To run issue, use the following command:

```
node seriallist.js
```

The output of this function will be the JSON data from your Arduino controller.

In addition to reading serial data from the Arduino, you want to be able to send it to your Unity example. The first portion of the example reads serial data directly from the Arduino, but it only works on Windows. Hopefully, Mono will be updated soon, and you'll be able to read serial data from both Mac OS X and Linux, as well. In the meantime, you can explore a more powerful option that allows you to have a network of controllers and devices that can be integrated using Unity. This can be done using a module called node-OSC, which takes inputted data and sends it to an OSC server. In this case, the OSC server is in your Unity project.

The code for this is shown in Listing XX-4.

Listing XX-4. Reading Serial Data and Converting to OSC via Node.js

```
var SerialPort, count, dgram, osc, outport, serialport, sp, sys, tty, udp;
  osc = require("osc-min");
  dgram = require("dgram");
  udp = dgram.createSocket("udp4");
  serialport = require("serialport");
  sys = require("sys");

  SerialPort = serialport.SerialPort;

  outport = 10000;
  count = 0;
  tty = "/dev/tty.usbserial-A1009UMS";

  sp = new SerialPort(tty, {
    parser: serialport.parsers.readline("\n"),
    baudrate: 57600
  });

  sp.on("data", function(data) {
    var buf;
    buf = osc.toBuffer({
      address: "/ardcontroller",
      args: [data]
    });
    udp.send(buf, 0, buf.length, outport, "localhost");
    return console.log("Count: " + count++ + " Data: " + buf);
  });

  sp.on("error", function(error) {
     return console.log("Error: " + error);
   });
```

This is done in addition to setting the serial port variable to the correct device path or COM port. This will help you to configure a UDP OSC client. UDP is a fast network protocol that has no error checking and doesn't require order, as does TCP. The UDP port must be unblocked by the firewall in order for it to travel over the Internet. This is configured to send data to the `localhost` listening on port 10000.

To run this, type the following text at the command line and press Return:

```
node node-osc-serial.js
```

Reading Serial Data from Unity

Reading serial data from Unity uses the basic .NET functions for reading a serial device. Unfortunately, this only works on Windows. However, many people have issued complaints, so there is a chance this will be fixed. In any case, it is very much worth doing.

Your `InputManager.cs` script has a section where the input type equals the serial, which the following code triggers:

```
if(inputType=="Serial")
{
        string[] ports = SerialPort.GetPortNames();
        stream = new SerialPort(ports[0], 57600);
        stream.Open();
        StartCoroutine("SlowUpdate");
}
```

The example code produces a list of serial ports and picks the first one. On Windows, these are COM ports. On Mac OS X and Linux, these are serial devices (such as `/dev/tty`). In any case, once the ports are opened, a thread is created. Since the Arduino can update at several different baud rates, you can choose to accept various data rates from the Arduino. The recommended baud rate is 57,600, which is necessary to transfer the data quickly. The remaining initialization code is shown here:

```
IEnumerator SlowUpdate()
{
        while(true)
        {
                string val = stream.ReadLine();
                Debug.Log(val);
                inputFromArduino = JsonMapper.ToObject(val);
                yield return new WaitForSeconds(.25f);
        }
}
```

In the `SlowUpdate` code, a line of serial data is read and then parsed as JSON data. This continues indefinitely.

Reading OSC UDP Data from Unity

Once the OSC Unity module is dragged into the project, the OSC code can be used inside the `InputManager.cs` code. In this case, you talk to the OSC server and pick up the latest information (see Listing XX-5).

Listing XX-5. OSC Update

```
IEnumerator OSCUpdate()
  {
   while(true)
   {

     OSCHandler.Instance.UpdateLogs();
    oscServers = OSCHandler.Instance.Servers;

     foreach(KeyValuePair<string, ServerLog> item in oscServers)
          {
              if(item.Value.log.Count > 0)
               {
                    int lastPacketIndex = item.Value.packets.Count - 1;

                    Debug.Log(String.Format("SERVER: {0} ADDRESS: {1} VALUE 0:
{2}",
                                                        item.Key,
item.Value.packets[lastPacketIndex].Address,

item.Value.packets[lastPacketIndex].Data[0].ToString()));

     Debug.Log("item.Key: " + item.Key);
     Debug.Log("item.Value: " + item.Value);
     Debug.Log("item.Value.packets[lastPacketIndex].Data[0].ToString()): " +
item.Value.packets[lastPacketIndex].Data[0].ToString());
     string oscData = item.Value.packets[lastPacketIndex].Data[0].ToString();
     inputFromArduino = JsonMapper.ToObject(oscData);
     UnityEngine.Debug.Log(inputFromArduino);

               }
          }
    yield return new WaitForSeconds(.25f);
   }
  }
```

The code is updated after it talks with the server, and it unpacks the first item and converts it to a string. In this case, the retrieved string data is actually JSON data, which you can then pass from the OSC source as your Arduino data, since it is identical to that which you used for the Serial Data scene. This means you can use the same Arduino sketch for both kinds of communication.

Summary

This chapter outlined the ease and power of creating custom controllers using Arduino. You used the cutting-edge language Node.js, which is efficient and provides a succinct information exchange from serial data to the OSC data using JSON as the data format between the various devices. Using JSON provides one standard data format from the device (e.g., Arduino) to the game system (e.g., Unity). The 32u4 chip used by the Leonardo and the Teensy has extra analog pins that allow for gathering more analog data for joysticks, potentiometers, and in the future, temperature sensors. Future projects could evens use RFID to interact with games screens or unlock in-world content. The next steps in this area create bidirectional communication and multiple controlled network controllers. For instance, building sensor networks can be integrated into a virtual environment that shows the status of the building in full interactive 3D. Ultimately, the Internet of Things can integrate the real world, the virtual world, and game data using the basic techniques demonstrated here.