

TUTORIAL A

Context Sensing with Phidgets and building the Pressalit Hygiene Sensor

INTRODUCTION:

Phidgets (<http://www.phidgets.com/>) are used for hardware prototyping pervasive computing prototypes. They belong to a family of hardware prototyping toolkits that include Ardunio, Netdunio, Sun SPOT, FEZ and related technologies.

Phidgets are slightly more sophisticated as compared to most other prototyping toolkits, as they provide extensive programming API's for most modern OS and programming platforms, including Windows, Linux, iOS, Android, iPhone, as well as MatLab and LabView for research purposes.

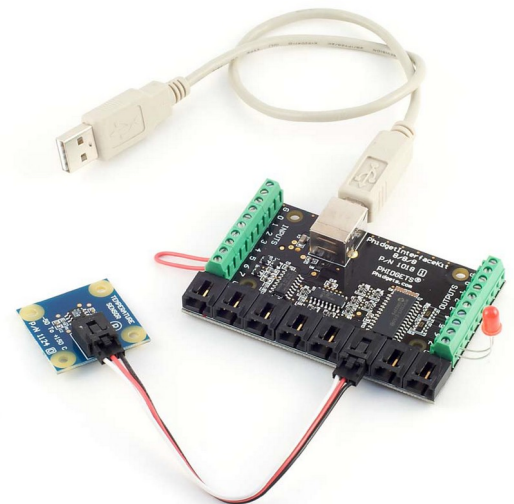
The Phidget Interface Kit family of boards allow for easy USB connectivity to a wide range of sensors and actuators, allowing programming of these using your preferred programming environment.

The Phidget SBC family of boards allow for independent operation of devices, by using a Linux based OS. Rather than using USB, it is now also possible to connect using Ethernet or WiFi connections. The Phidget SBC board can be accessed via a native Web service providing a high degree of access transparency, or computation can be done on board, using Java, C# (Mono.Net) or C++.

PHIDGET INTERFACE KIT:

The PhidgetInterfaceKit 8/8/8 allows you to connect devices to any of 8 analog inputs, 8 digital inputs and 8 digital outputs. It provides a generic, convenient way to interface your PC with various sensor devices. A variance of this is the PhidgetInterfaceKit 2/2/2 which is smaller but cheaper. You shall try to use both types, as well as the SBC kit, and relay switching boards.

The analog inputs are used to measure continuous quantities, such as temperature, humidity, position, pressure, etc. Phidgets offers a wide variety of sensors that can be plugged directly into the board using the cable included with the sensor. These sensors are the same that you may find for Ardunio and other open hardware projects, but Phidgets allow for quite easy access.



Procedure for the exercises:

1. Work in your groups!
2. Start with only one kit in the group so that you help each other getting it to work.
3. End up with all of you having tried installing the devices and having coded something relevant
4. Go through the main exercise scenarios (a-f). Focus on at least getting a-c done today.
5. **As there are not enough of all Phidget Interface Kits, at least one group (group 1) should start with exercise scenario b), and then switch with another group when they are done**

Operating Systems Drivers

 - Windows	[+]
 - macOS	[+]
 - Linux	[+]
 - iOS	[+]
 - Android	[+]
 - Phidget SBC	[+]

a) Exercises for Phidgets Interface Kit 2/2/2

1. Start with reading the product manual http://www.phidgets.com/docs/1011_User_Guide and download drivers from: http://www.phidgets.com/docs/Operating_System_Support (find your OS and follow the instructions). For Phidget22 for Windows, it is: https://www.phidgets.com/docs/OS_-_Windows#Quick_Downloads (choose 64 bit for most of your laptop PC's)
2. Next, install the Phidget Interface Kit drivers and connect to them using the Phidget Control Panel
3. Try out the example applications available from Phidgets (either C#, Java, C++ as you please) https://www.phidgets.com/?view=code_samples&lang=CSharp
4. Find an IR reflective sensor, which has a detection range of 4 mm. Try it out and discuss with your group what should a device can be used for?

b) Exercises for Phidget Interface Kit 8/8/8:

1. Start with reading the product manual
http://www.phidgets.com/documentation/Archive/1018_1_Product_Manual.pdf and download drivers from: http://www.phidgets.com/docs/Operating_System_Support (find your OS and follow the instructions)
2. Next, install the Phidget Interface Kit drivers and connect to them using the Phidget Control Panel
3. Try out the example applications available from Phidgets (either C#, Java, C++ as you please)
4. Try using the interface kit for accessing the output port by lighting two LED's in different colors to showcase ambient feedback to a user – by making your own console application
5. Try using the interface kit for accessing the input port by using the pressure sensor mat.
6. Find a Temperature/Humidity sensor, sample the values every 1 second, convert them to celcius and “percentage of humidity” – and display it in a Java, Android, iOS, or Windows C# WPF or Forms program (only the latter two are supported on class).
7. Find a distance sensor or ultrasound sensor and test the precision and accuracy of the sensor. Can you use it to see if a user is moving towards a touch screen computer mounted on the wall?



c) Exercises with the RFID kit

The PhidgetRFID Read-Write reads RFID tags that are brought in close proximity to the reader and returns the tag identification number. Writing data to T5577 tags is also supported.

RFID (radio frequency identification) systems use data strings stored inside RFID tags to uniquely identify people or objects when their tags are scanned by an RFID reader. These types of systems are found in many applications such as passport protection, animal identification, inventory control systems, and secure access control systems.

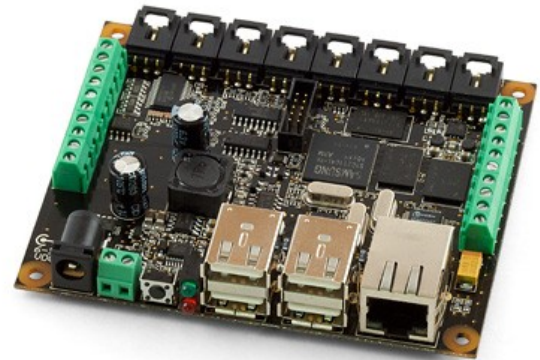
The PhidgetRFID Read-Write supports reading and writing in 3 protocols; EM4100, ISO11785 FDX-B, and PhidgetTag. The PhidgetTag protocol simply stores up to 24 ASCII characters to the tag, eliminating the necessity for a table of corresponding tag numbers and names in your program. Phidgets sells EM4100 read-only tags that can be read with either of our RFID readers, and writable tags which can be written with the 1024 using any protocol. Any 3rd-party EM4100 or ISO11785 tags can be read.

Because passive tags require a strong RF field to operate, their effective range is limited to an area in close proximity to the RFID reader. The distance over which the RFID tag is usable is affected by such things as the tag shape and size, materials being used in the area near the reader, and the orientation of the reader and tag in respect to each other and in their operating environment. The smaller a tag, the closer it must be to the reader to operate.

1. Find an RFID kit and connect it with the phidget panel. Try out holding a tag near to it.
2. Make a C# (or other) console application that will react to seeing an RFID tag placed on it and provide some feedback to the user. It could be a user logging in to a system, or it could be a nurse registering a new device. Discuss where a tag could be placed if it should be used to track a device?
3. Have a look at: <http://www.phidgets.com/products.php?category=14> and discuss what the different types of tags could be used for. Are there any other tags you would like to use?

d) Exercises with the Phidget SBC

The PhidgetSBC is a Single Board Computer with an integrated PhidgetInterfaceKit 8/8/8. At its most basic, it can be thought of as a Phidget that you connect using a network cable instead of USB. The PhidgetSBC also provides six full-speed ports that allow you to use normal USB Phidgets over its network connection. This can extend the effective range of a Phidget from USB's maximum of 15 feet, to anywhere that your network reaches. This is achieved using a web service interface.



The PhidgetSBC exposes an easy to use interface for setting up and running custom applications on-board.

This allows the PhidgetSBC to operate autonomously, without the need for a graphical interface or a remote connection at all times.

An integrated PhidgetInterfaceKit 8/8/8 allows you to connect devices to any of 8 analog inputs, 8 digital inputs and 8 digital outputs.

For more advanced users, the PhidgetSBC is an embedded computer that runs Debian GNU/Linux. Full shell access is provided via a built-in SSH server, access to the full Debian package repository. This means you can run Java or C++ applications in the Linux environment. PhidgetSBC2 can also be expanded to run C# and more.

1. Start with reading the guidelines for Phidget SBC installation and configuration:
http://www.phidgets.com/products.php?product_id=1073 (Use <http://www.phidgets.com/documentation/Phidgets/1072.pdf> for PhidgetSBC2)
2. Please note – you need a DNS enabled network that will allow the PhidgetSBC to connect and obtain an IP address. Usually this means that you will need a router unless your computer. Phidget SBC also supports zero configuration networking with Bonjour – but then you will need to have such a device available.
3. Also please note – you may have to do a firmware update on the Phidget SBC to get it running with the current version of the API.
4. Install the PhidgetSBC drivers and connect to it using the Phidget Control Panel
5. Connect the Phidget SBC to your computer with an Ethernet cable
6. Try out the example applications available from Phidgets (either C#, Java, C++ as you please)
7. Try out a range of the available sensors using the Phidgets software API's.
8. Try attaching the WiFi USB modem including configuring using the Web configuration interface
9. Try attaching the camera and / or audio devices, and access these from a PC application over the internet. See if you can use the camera to monitor a patient. E.g. when a movement sensor notices movement, use the web cam to grab a picture.

10. Try creating a Java or C++ application to upload to the SBC for non-PC interaction. Discuss when you would not want to use the SBC with a PC
11. Try creating an application for your Android phone or tablet
12. Try connecting an RFID board and a Sound Level Board to the PhidgetSBC ports. What happens?
13. Discuss with your group when you would want to use the Phidget Interface kits 2/2/2, 8/8/8 and when to use the PhidgetSBC boards?

e) Exercises with the Phidgets Interface Kit 0/0/4 (Relays)

The PhidgetInterfaceKit 0/0/4 provides a convenient way to interface your PC with various higher-voltage devices such as incandescent bulbs, high-power relays, and motors. This includes turning the lights on – or the television for instance. The kit contains 4 Relay Outputs for switching AC or DC power.

1. Start with reading the product manual http://www.phidgets.com/products.php?category=9&product_id=1014_2 and download drivers from: http://www.phidgets.com/docs/Operating_System_Support (find your OS and follow the instructions)
2. Next, install the Phidget Interface Kit drivers and connect to them using the Phidget Control Panel
3. Try turning the lights on and off
4. Build an application where you turn the lights on when there is no longer pressure on the pressure pad. This could be used in the bed of a user, or when someone steps on a doormat or similar.
5. Consider how you may use this to turn the television on when a user is standing in front of it ready to do exercises.

f) Exercises with the PhidgetBridge 4-Input (Load Cells)

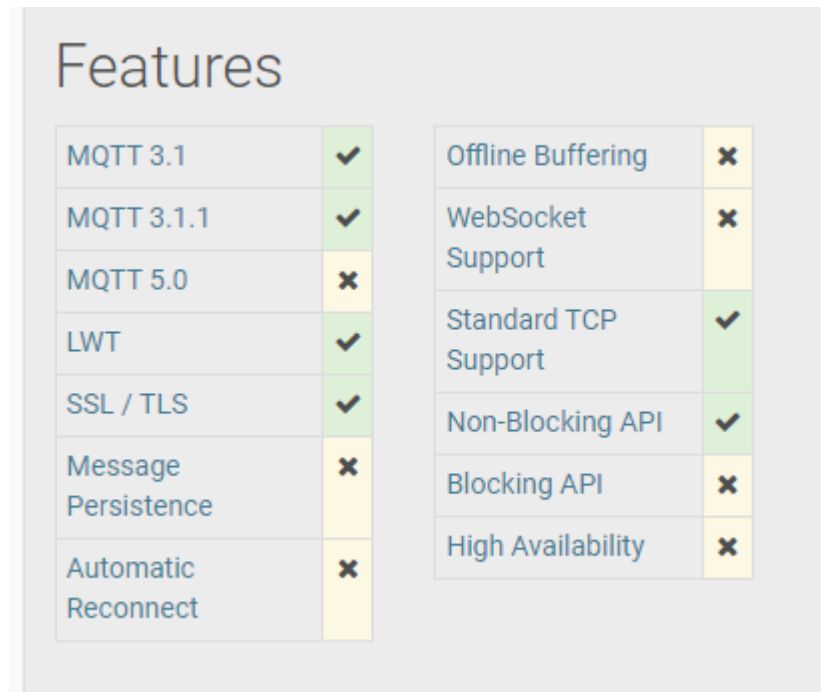
The PhidgetBridge is the interface board needed to measure the output from a load cell. You can connect up to four load cells, strain gauges, or wheatstone bridge sensors.

1. Start with reading the product manual http://www.phidgets.com/products.php?category=34&product_id=1046_0 and download drivers from: http://www.phidgets.com/docs/Operating_System_Support (find your OS and follow the instructions)
2. Next, install the Phidget drivers and connect to them using the Phidget Control Panel
3. Try attaching four sensor to e.g. a chair and see if you can get an accurate weight measurement out of it.

g) Sending Phidget events to MQTT server using Paho M2MQTT

In this step – you will be using the Paho M2MQTT C# .NET client to distribute an event.

M2Mqtt is a MQTT client available for all .Net platforms (.Net Framework, .Net Compact Framework and .Net Micro Framework) and WinRT platforms (Windows 8.1 and Windows Phone 8.1).



MQTT 3.1	✓	Offline Buffering	✗
MQTT 3.1.1	✓	WebSocket Support	✗
MQTT 5.0	✗	Standard TCP Support	✓
LWT	✓	Non-Blocking API	✓
SSL / TLS	✓	Blocking API	✗
Message Persistence	✗	High Availability	✗
Automatic Reconnect	✗		

Read more on Paho here: <https://www.eclipse.org/paho/index.php?page=clients/dotnet/index.php>

You can either download the Nuget assembeblies – or build them from source.

Download

The M2Mqtt client assemblies for using as references in your Visual Studio projects can be downloaded from <https://www.nuget.org/packages/M2Mqtt/> for .NET framework (“classic” Windows .NET) and .NET Core (for Linux and Mac)

Building from source

The project can be installed from the repository as well. To do this:

git clone <https://github.com/eclipse/paho.mqtt.m2mqtt.git>

You can open one of the available solutions for Visual Studio in the "org.eclipse.paho.mqtt.m2mqtt" folder

Documentation

Full client documentation is available on the official M2Mqtt project web site [here](#).

Step 1) Make sure you have Visual Studio 2019 or newer

Step 2) We assume you have installed Phidgets drivers already

Step 3) Consider to use the Sample Builder on Phidgets website

https://www.phidgets.com/?view=code_samples&lang=CSharp

The screenshot shows a web browser window with the URL https://www.phidgets.com/?view=code_samples&lang=CSharp. The page title is "Code Samples". Below it, the section "Device-Specific Code Samples" is visible. The configuration is set for "C#" and "1011_0 - PhidgetInterfaceKit 2/2/2". There is a "View API for this device" link. The "Serial Number" field is optional. Under "Channels", the "Voltage Ratio Input" is selected. Under "Example Options", "Attach/Detach Handlers", "Separate Event Handlers", "Exception Handling", and "Press Enter" are selected. A "DOWNLOAD EXAMPLE" button is at the bottom.

Code Samples

Device-Specific Code Samples

C# 1011_0 - PhidgetInterfaceKit 2/2/2 View API for this device

Serial Number:

Channels:

Voltage Input ☐ 0 ☐ 1

Voltage Ratio Input ☒ 0 ☐ 1

Digital Input ☐ 0 ☐ 1

Digital Output ☐ 0 ☐ 1

Example Options

☐ Show Comments ☒ Exception Handling ☐ Remote (Network)

☒ Attach/Detach Handlers ☐ Error Event Handler ☒ Press Enter

☒ Separate Event Handlers ☐ Diagnostic Logging

Downloads

DOWNLOAD EXAMPLE

Make sure to generate to the correct controller you are using

```

using System;
using Phidget22;

namespace ConsoleApplication
{
    class Program
    {
        private static void VoltageRatioInput0_VoltageRatioChange(object sender, Phidget22.Events.VoltageRatioInput
        {
            Console.WriteLine("VoltageRatio: " + e.VoltageRatio);
        }

        private static void VoltageRatioInput0_Attach(object sender, Phidget22.Events.AttachEventArgs e)
        {
            Console.WriteLine("Attach!");
        }

        private static void VoltageRatioInput0_Detach(object sender, Phidget22.Events.DetachEventArgs e)
        {
            Console.WriteLine("Detach!");
        }

        static void Main(string[] args)
        {
            VoltageRatioInput voltageRatioInput0 = new VoltageRatioInput();

            voltageRatioInput0.VoltageRatioChange += VoltageRatioInput0_VoltageRatioChange;
            voltageRatioInput0.Attach += VoltageRatioInput0_Attach;
            voltageRatioInput0.Detach += VoltageRatioInput0_Detach;

            try
            {
                voltageRatioInput0.Open(5000);

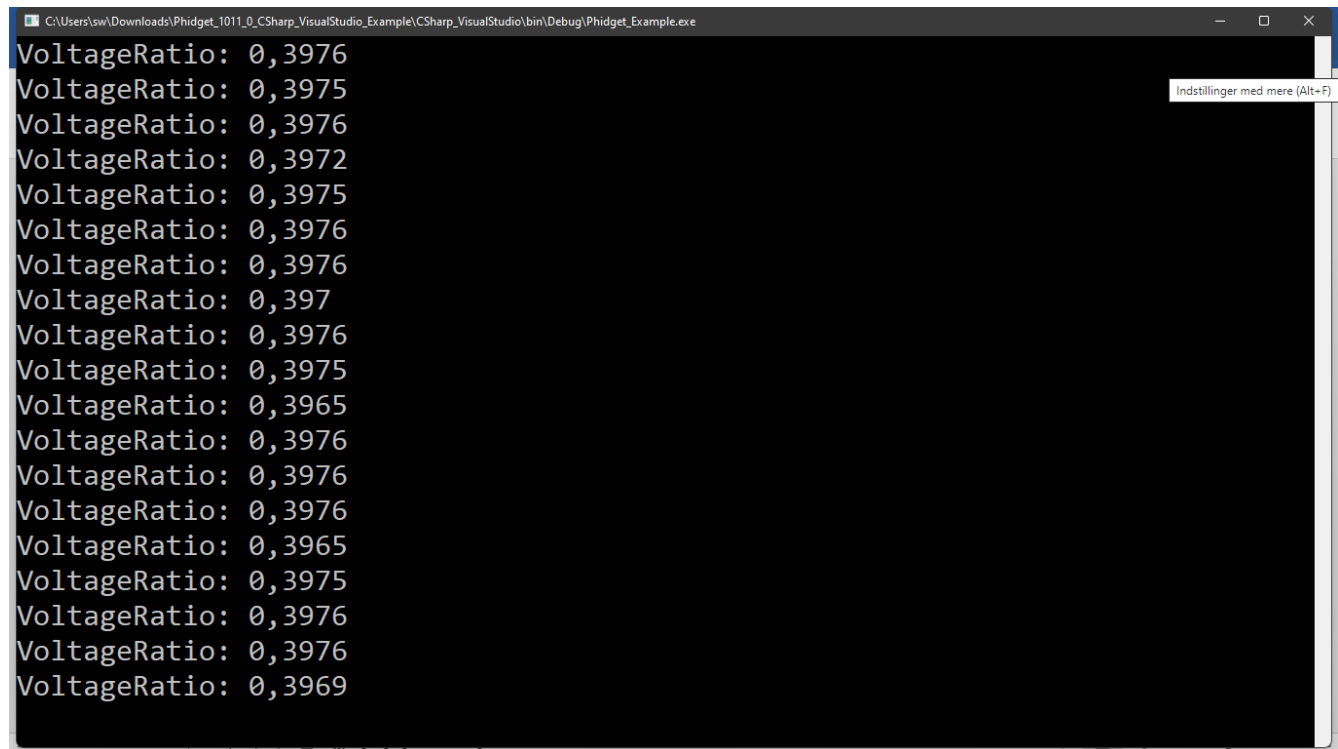
                //Wait until Enter has been pressed before exiting
                Console.ReadLine();

                voltageRatioInput0.Close();
            }
            catch (PhidgetException ex)
            {
                Console.WriteLine(ex.ToString());
                Console.WriteLine("");
                Console.WriteLine("PhidgetException " + ex.ErrorCode + " (" + ex.Description + "): " + ex.Detail);
            }
        }
    }
}

```



Step 4) Now run the code



```
C:\Users\sw\Downloads\Phidget_1011_0_CSharp_VisualStudio_Example\CSharp_VisualStudio\bin\Debug\Phidget_Example.exe
VoltageRatio: 0,3976
VoltageRatio: 0,3975
VoltageRatio: 0,3976
VoltageRatio: 0,3972
VoltageRatio: 0,3975
VoltageRatio: 0,3976
VoltageRatio: 0,3976
VoltageRatio: 0,397
VoltageRatio: 0,3976
VoltageRatio: 0,3975
VoltageRatio: 0,3965
VoltageRatio: 0,3976
VoltageRatio: 0,3976
VoltageRatio: 0,3976
VoltageRatio: 0,3965
VoltageRatio: 0,3975
VoltageRatio: 0,3976
VoltageRatio: 0,3976
VoltageRatio: 0,3969
```

Step 5)

Now – I have attached a Sharp Distance sensor. I want something useful

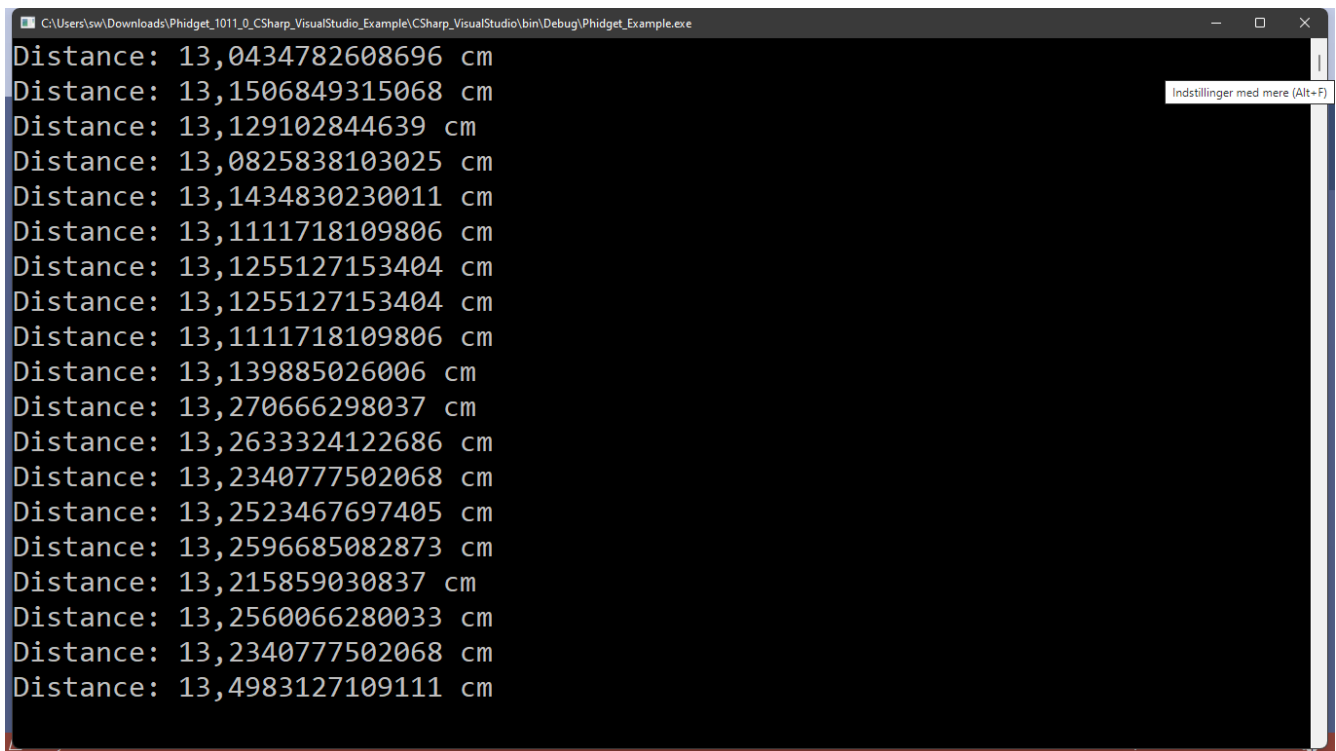
The formula to translate voltage into Distance for Sharp 10-80cm analog sensors is:

$$\text{Distance (cm)} = \frac{4.8}{\text{VoltageRatio} - 0.02}$$

1 reference

```
private static void VoltageRatioInput0_VoltageRatioChange(object sender)
{
    //Console.WriteLine("VoltageRatio: " + e.VoltageRatio);
    var dist = 4.8 / (e.VoltageRatio - 0.02);

    Console.WriteLine("Distance: " + dist + " cm");
}
```



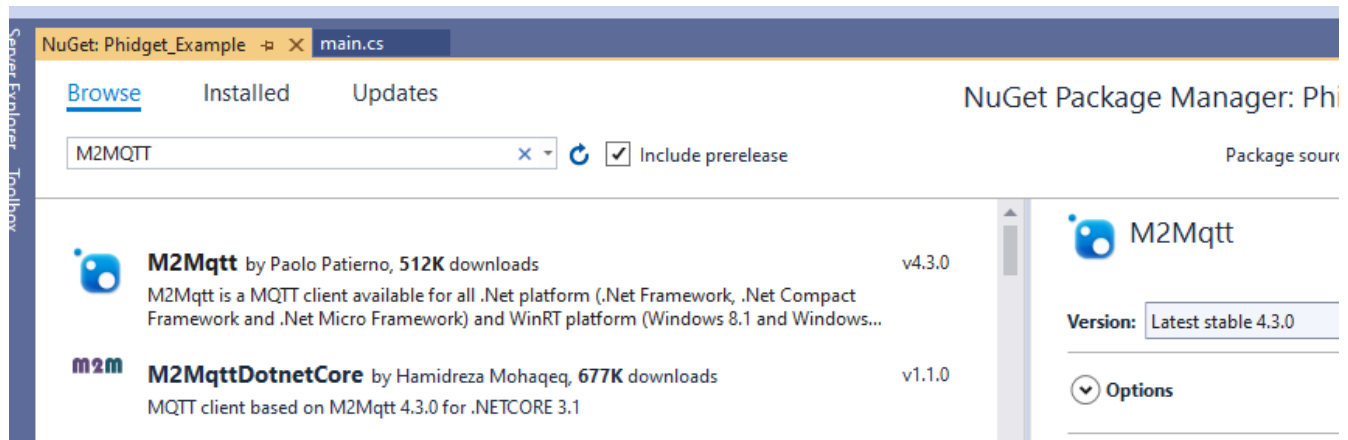
```
C:\Users\sw\Downloads\Phidget_1011_0_CSharp_VisualStudio_Example\CSharp_VisualStudio\bin\Debug\Phidget_Example.exe
Distance: 13,0434782608696 cm
Distance: 13,1506849315068 cm
Distance: 13,129102844639 cm
Distance: 13,0825838103025 cm
Distance: 13,1434830230011 cm
Distance: 13,1111718109806 cm
Distance: 13,1255127153404 cm
Distance: 13,1255127153404 cm
Distance: 13,1111718109806 cm
Distance: 13,139885026006 cm
Distance: 13,270666298037 cm
Distance: 13,2633324122686 cm
Distance: 13,2340777502068 cm
Distance: 13,2523467697405 cm
Distance: 13,2596685082873 cm
Distance: 13,215859030837 cm
Distance: 13,2560066280033 cm
Distance: 13,2340777502068 cm
Distance: 13,4983127109111 cm
```

Step 6)

Fine. Now we have a context aware system. But, it is not yet distributed. Let's distribute it with MQTT.

Ensure you have Mosquitto or other MQTT broker running.

Fetch M2MQTT Nuget package ...



Now to the code

```
using uPLibrary.Networking.M2Mqtt;  
using uPLibrary.Networking.M2Mqtt.Messages;
```

And in the main

```
static void Main(string[] args)  
{  
    //Now let's add the M2MQTT client  
    string BrokerAddress = "localhost";  
    //Use localhost or test.mosquitto.org if you want to use a distri  
    //Please not - standard port is not 1887 - but 1883  
    client = new MqttClient(BrokerAddress, 1887, false, null, null, M  
    // use a unique id as client id, each time we start the applicati  
    var clientId = Guid.NewGuid().ToString();  
    //Now connect. If you have NO user or password - just remove  
    client.Connect(clientId, "test", "test");  
}
```

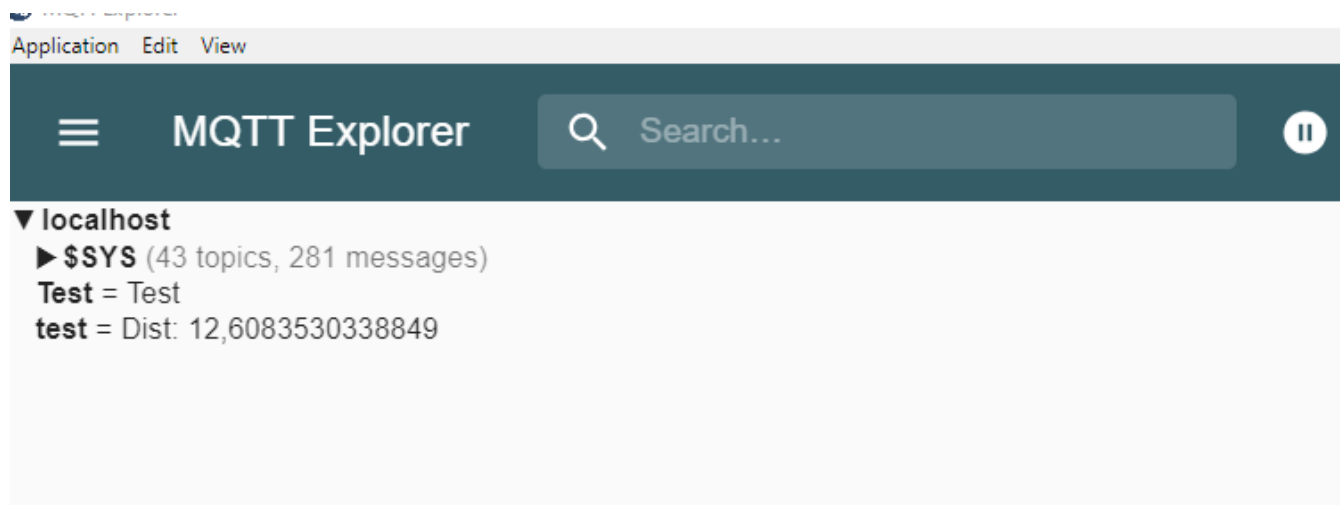
And to publish the distance

1 reference

```
private static void VoltageRatioInput0_VoltageRatioChange(object sender, EventArgs e)
{
    //Console.WriteLine("VoltageRatio: " + e.VoltageRatio);
    var dist = 4.8 / (e.VoltageRatio - 0.02);

    Console.WriteLine("Distance: " + dist + " cm");
    client.Publish("test", Encoding.UTF8.GetBytes(("Dist: " + dist)))
}
```

Now run it ... and see the following:



Step 7)

Now – let us also listen after “test” topic events (subscribe) – which is of course something we would want to do in our SERVICE

```

var clientId = Guid.NewGuid().ToString();
//Now connect. If you have NO user or password - just remove

// register a callback-function (we have to implement, see below) which
client.MqttMsgPublishReceived += client_MqttMsgPublishReceived;
//client.MqttMsgPublished += Client_MqttMsgPublished;
client.Connect(clientId, "test", "test");

if (client.IsConnected)
{
    client.Subscribe(new string[] { "test" }, new byte[] { MqttMsgBase.Q
}

```

Here – you can choose to operate at different semantics. At most once, at least once, exactly once.

MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE

```

// this code runs when a message was received
1 reference
static void client_MqttMsgPublishReceived(object sender, MqttMsgPubl
{
    string ReceivedMessage = Encoding.UTF8.GetString(e.Message);

    Console.WriteLine("I got this from broker: " + ReceivedMessage);
}

```

```
C:\Users\sw\Downloads\Phidget_1011_0_CSharp_VisualStudio_Example\CSharp_VisualStudio\bin\Debug\Phidget_Example.exe
I got this from broker: Dist: 13,0293159609121Distance: 13,0470236477304 cm
I got this from broker: Dist: 13,0470236477304Distance: 12,9835001352448 cm
I got this from broker: Dist: 12,9835001352448Distance: 13,039934800326 cm
I got this from broker: Dist: 13,039934800326Distance: 13,0222463374932 cm
I got this from broker: Dist: 13,0222463374932Distance: 13,0257801899593 cm
I got this from broker: Dist: 13,0257801899593Distance: 13,0612244897959 cm
I got this from broker: Dist: 13,0612244897959Distance: 13,0683365096651 cm
I got this from broker: Dist: 13,0683365096651Distance: 13,0046057978868 cm
I got this from broker: Dist: 13,0046057978868Distance: 13,2413793103448 cm
I got this from broker: Dist: 13,2413793103448Distance: 14,1509433962264 cm
I got this from broker: Dist: 14,1509433962264Distance: 17,5118569865013 cm
I got this from broker: Dist: 17,5118569865013Distance: 48,8301119023398 cm
I got this from broker: Dist: 48,8301119023398Distance: 57,9710144927536 cm
I got this from broker: Dist: 57,9710144927536Distance: 70,2781844802343 cm
I got this from broker: Dist: 70,2781844802343Distance: 90,0562851782364 cm
I got this from broker: Dist: 90,0562851782364Distance: 73,1707317073171 cm
I got this from broker: Dist: 73,1707317073171Distance: 66,6666666666667 cm
I got this from broker: Dist: 66,6666666666667Distance: 192,771084337349 cm
I got this from broker: Dist: 192,771084337349Distance: 65,7534246575343 cm
I got this from broker: Dist: 65,7534246575343
```

Step 8)

Now consider this (discuss in your groups):

- i) Now the service is available via all IP enabled networks incl. the internet. How can different programming language use e.g. a Distance Class as a common object? How can they communicate?
- ii) If we want to make an intelligent environment – how can we change the text format into something that other services could use?
- iii) How about security? Is this a secure solution?

h) Building the Pressalit Hygiene Sensor solution with Phidgets

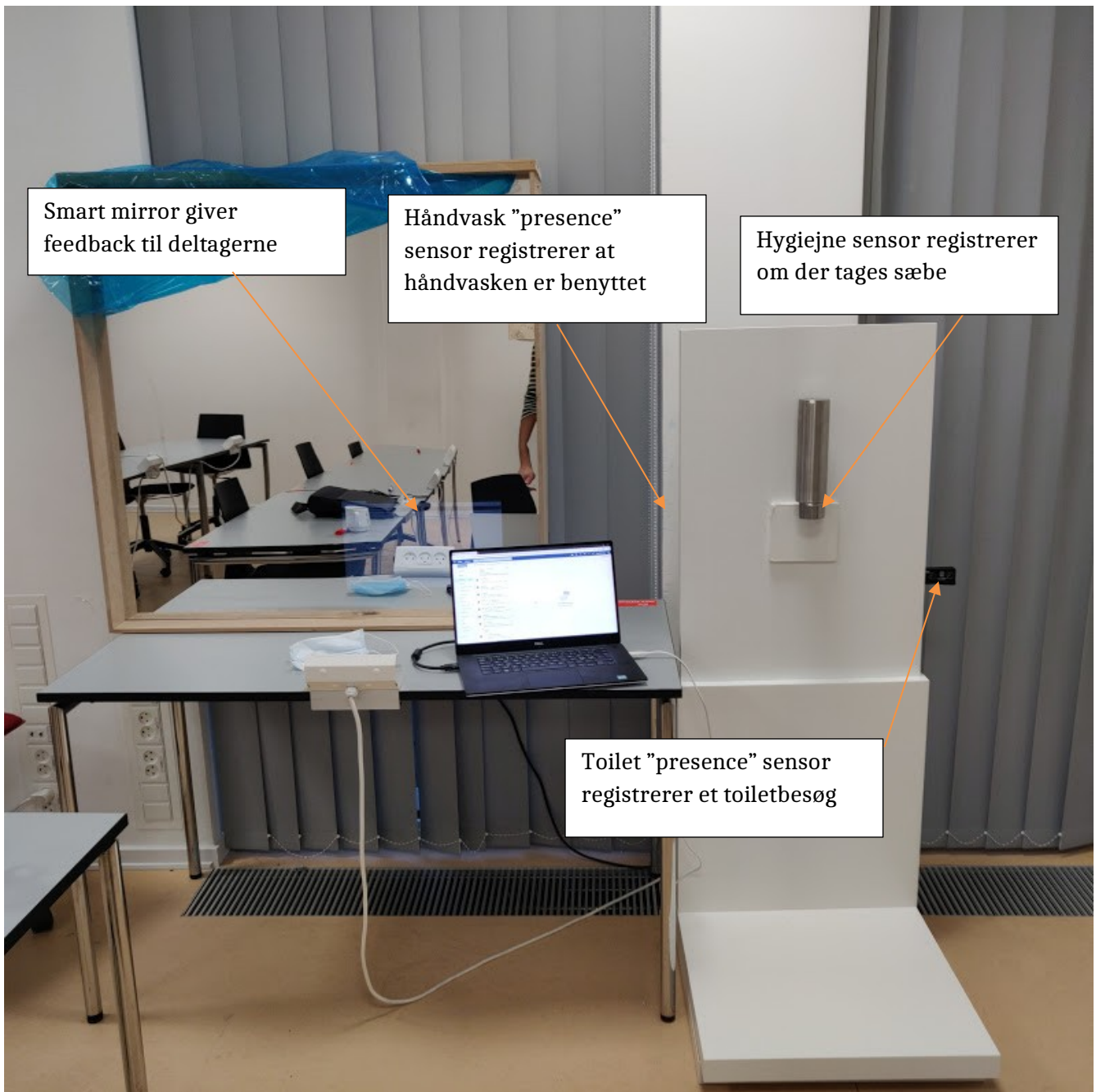
For this exercise – we are using a Phidgets VINT HUB AD converter along with a capacitive touch sensor to support sensing whether soap is being dispensed. Also, we use two ultrasound sensors to sense a toilet visit and handwashing respectively. Now, neither of these three sensors will guarantee that the action was performed properly, but it they will add to the likelihood of it being so.

Smart mirror giver
feedback til deltagerne

Håndvask "presence"
sensor registrerer at
håndvasken er benyttet

Hygiejne sensor registrerer
om der tages sæbe

Toilet "presence" sensor
registrerer et toiletbesøg







You will need to connect to the VINT HUB via USB cable for the existing setup as shown below.

We use Input 0 for the capacitive touch – and 3 and 4 for the ultra sound sensors



Find the source code under “Source”.

The below snippet is only for showcasing what is needed to initialize the sensors

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Phidget22;
using uPLibrary.Networking.M2Mqtt;
using uPLibrary.Networking.M2Mqtt.Messages;
using OpenCare.EVODAY;
using Phidget_Example;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
using System.Text;

namespace ConsoleApplication
{
    class Program

        static void Main(string[] args)
        {
```

```

Log.Enable(LogLevel.Info, "phidgetlog.log");
DistanceSensor distanceSensor0 = new DistanceSensor();
distanceSensor0.HubPort = 3;
//distanceSensor0.DistanceChange += DistanceSensor0_DistanceChange;
distanceSensor0.Attach += DistanceSensor0_Attach;
distanceSensor0.Detach += DistanceSensor0_Detach;
distanceSensor0.Error += DistanceSensor0_Error;

DistanceSensor distanceSensor1 = new DistanceSensor();
distanceSensor1.HubPort = 4;
//distanceSensor0.DistanceChange += DistanceSensor0_DistanceChange;
distanceSensor1.Attach += DistanceSensor0_Attach;
distanceSensor1.Detach += DistanceSensor0_Detach;
distanceSensor1.Error += DistanceSensor0_Error;

//VoltageRatioSensorType voltageRatioSensor = new VoltageRatioSensorType();
VoltageRatioInput voltageRatioInput = new VoltageRatioInput();
voltageRatioInput.HubPort = 0;
voltageRatioInput.IsHubPortDevice = true;
voltageRatioInput.Channel = 0;

voltageRatioInput.VoltageRatioChange += VoltageRatioInput_VoltageRatioChange;

try
{
    InitizlizeMqttClient();

    distanceSensor0.Open(5000);

    distanceSensor0.DataInterval = 2000;
    distanceSensor0.SonarQuietMode = false;

    distanceSensor1.Open(5000);
    distanceSensor1.DataInterval = 2000;
    distanceSensor1.SonarQuietMode = false;

    voltageRatioInput.Open(5000);
    //voltageRatioInput.SensorType = VoltageRatioSensorType.PN_1129;

    voltageRatioInput.DataInterval = 100;
}

```

Next, we send the HEUCOD event.

```

private static void SendHandwashingEvent()
{
    var sensorData = new OpenCare.EVODAY.EDL.HandWashingEvent();
    sensorData.SensorId = "PressalitHandWash1" + Setting.Default.PatientID;
    sensorData.Value = 1;
    sensorData.PatientId = Setting.Default.PatientID;
    sensorData.MonitorId = Setting.Default.PatientID;
    sensorData.Description = "Pressalit hand wash tracker";
    sensorData.DeviceModel = "SoapDetect";
    sensorData.DeviceVendor = "Pressalit";
    sensorData.Room = "Bathroom";
    sensorData.Timestamp = Utils.ConvertToUnixTime(DateTime.UtcNow);
    sensorData.StartTime = Utils.ConvertToUnixTime(DateTime.UtcNow);
}

```

```

        sensorData.EndTime = Utils.ConvertToUnixTime(DateTime.UtcNow);

        var message = OpenCare.EVODAY.Serialize.ToJson((BasicEvent)sensorData);

        SendMQTT(message);
    }

```

Likewise, we use the capacitative touch sensor for the soap dispenser.

```

private static void VoltageRatioInput_VoltageRatioChange(object sender, Phidget22.
Events.VoltageRatioInputVoltageRatioChangeEventArgs e)
{
    if (lastSoapDispensingEventSent == null || (DateTime.Now -
(DateTime)lastSoapDispensingEventSent).TotalSeconds > 300)

    if (e.VoltageRatio > 0.9)
    {
        Console.WriteLine("Soap dispensed");
        lastSoapDispensingEventSent = DateTime.Now;
        lastHandWashEventSent = null;
        SendSoapDispensing();
    }
}

```