AllJoyn Device System Bridge – GPIO Device Sample

This tutorial is prepared to show how a ***GPIO Device*** is simply exposed to the ***AllJoyn Bus*** using the ***AllJoyn Device System Bridge***.

Step 1: Hardware Setup

The sample uses a Raspberry Pi 2 that one of its GPIO pins is connected to a photo resistor as shown in the schematic below. If another device is sues the pin number in the code has to be changed to match the HW setup.

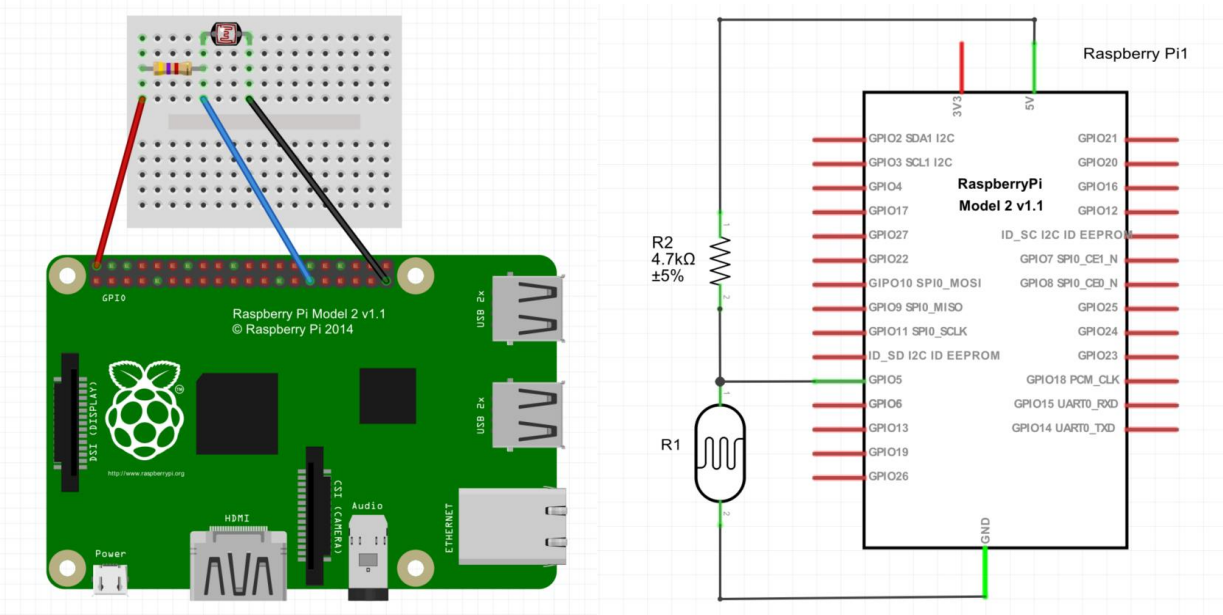

Figure 1 Sensor hook-up to Raspberry Pi2

Step 2: Download and Install the AllJoyn Device System Bridge Template

The ***AllJoyn Device System Bridge Template*** is a Visual Studio extension that enables developers to create an ***AllJoyn Device System Bridge App*** project.

Please go to the Products and Extensions for Visual Studio and search for **"AllJoyn DSB"** and download the template.

After the download, double-click on the **DeviceSystemBridgeTemplate.vsix** file to install the extension.

Step 3: Create an AllJoyn Device System Bridge App Project

In the Visual Studio, choose **File > New > Project** which opens the **New Project** dialog box. In the opening dialog box, create a new ***AllJoyn Device System Bridge App*** project as given in the following:

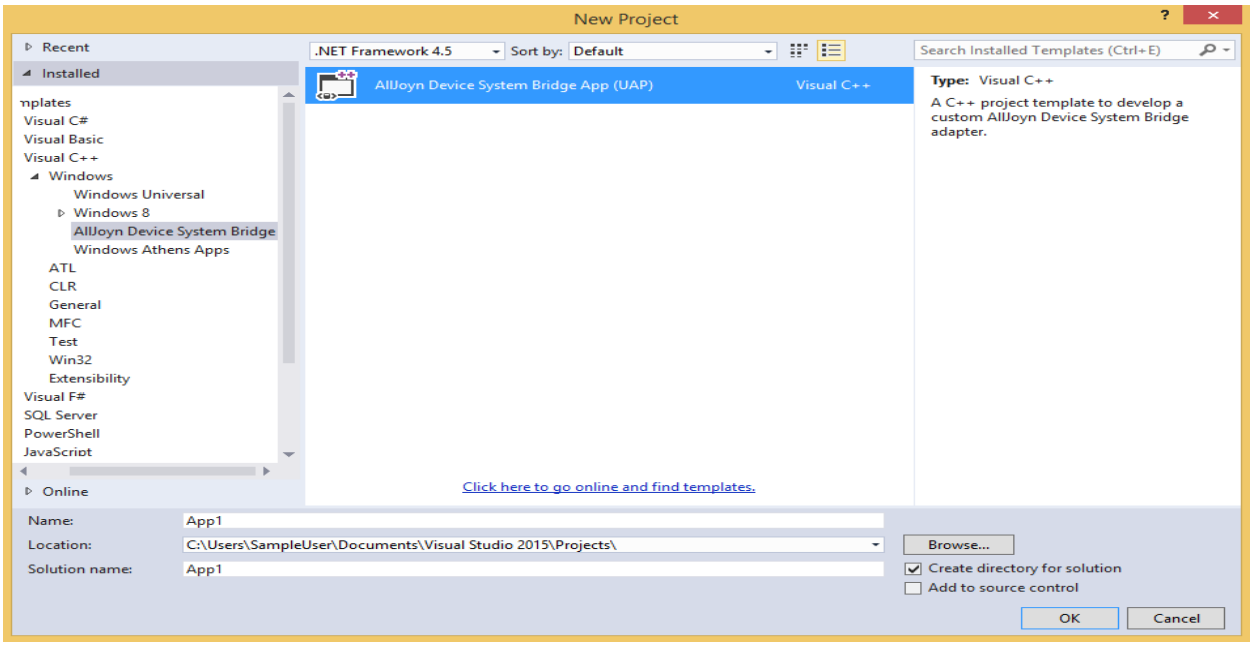

Figure 2: AllJoyn Device System Bridge App Project Creation

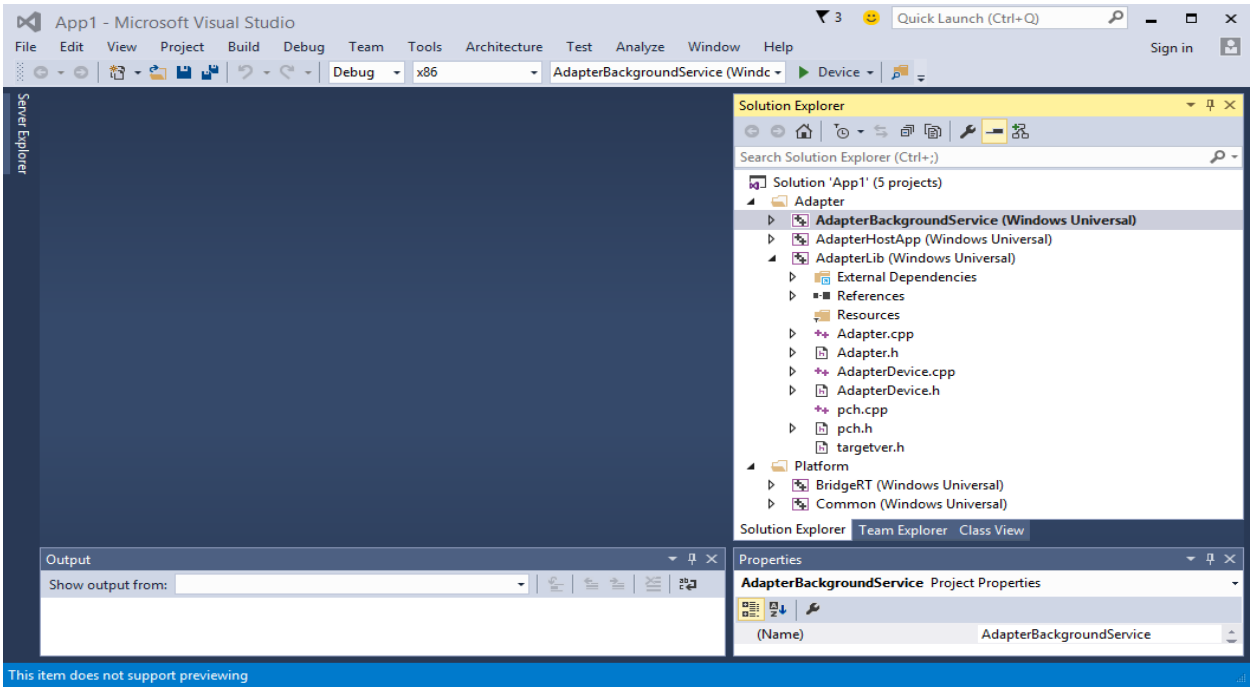

Figure 3: AllJoyn Device System Bridge App Sample Project

Step 4: Expose a GPIO Pin to the AllJoyn Bus

Add the following to the beginning of **AdapterLib's Adapter.cpp**.

```
using namespace Windows::Devices::Gpio;

// GPIO Device
String^ deviceName          = "Custom_GPIO_Device";
String^ vendorName          = "Custom_Vendor";
String^ modelName           = "Custom_Model";
String^ version             = "1.0.0.0";
String^ serialNumber        = "1111111111111";
String^ description         = "A Custom GPIO Device";

// GPIO Device Pin-5 Property
const int PIN_NUMBER        = 5;
String^ pinName             = "Pin-5";

// Pin-5 Property Attribute
String^ pinValueName        = "PinValue";
int     pinValueData        = -1;

GpioController^             controller;
GpioPin^                    pin;
```

Make sure to add **Windows IoT Extension SDK**, which is required to use the **Windows::Devices::Gpio** API, as a **Reference** to the **AdapterLib project**:

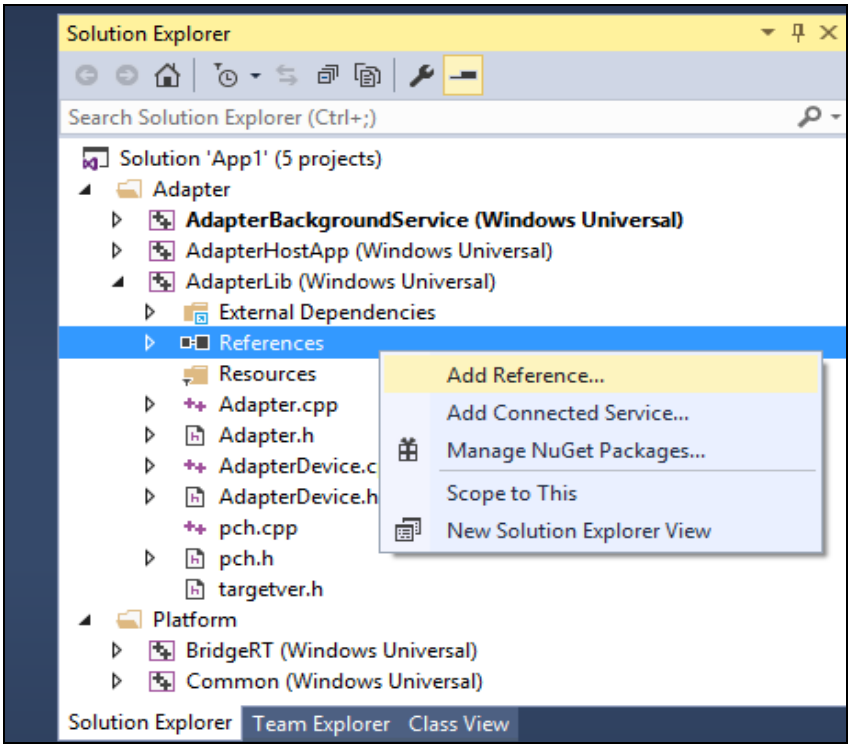

Figure 4: Adding reference to the AdapterLib

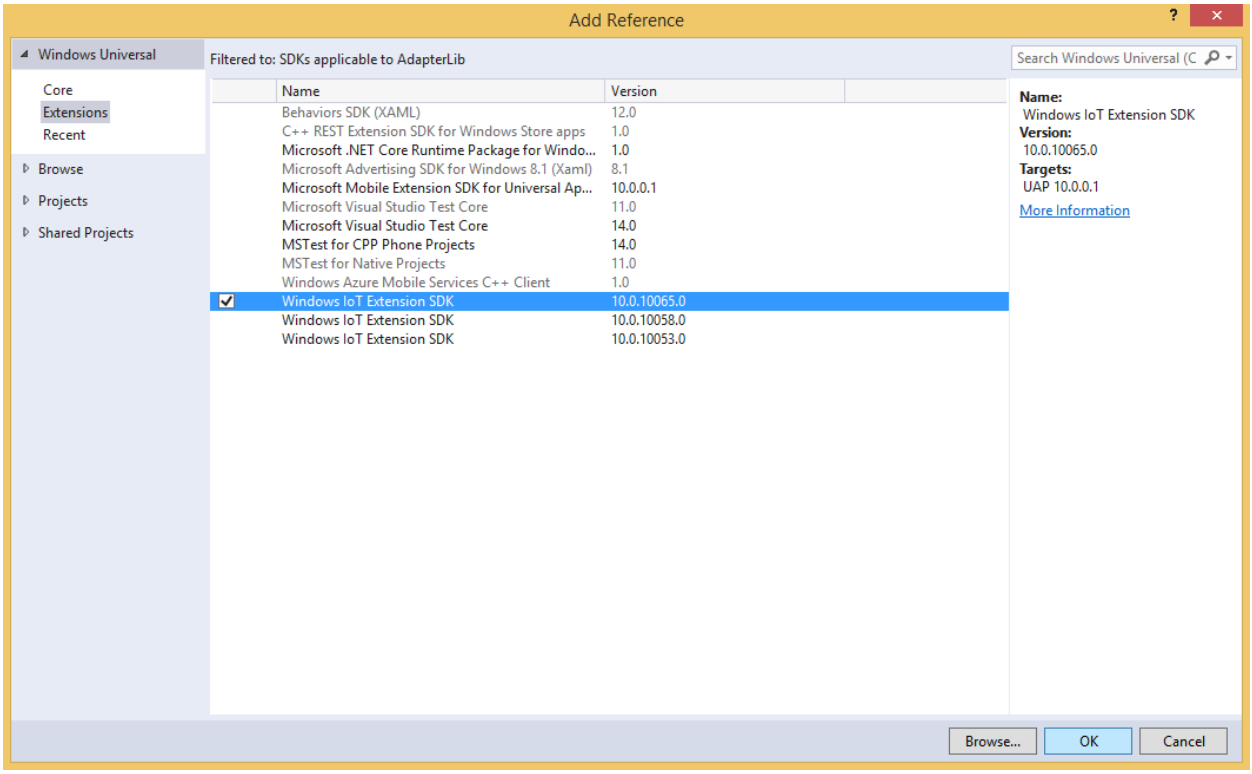

Figure 5: Adding the Windows IoT Extension SDK to the AdapterLib as a reference

In order to expose the ***GPIO Device*** to the ***AllJoyn Bus***, we need to create a corresponding ***Bridge Device (IAdapterDevice)*** instance. Modify the ***constructor*** of the **AdapterLib's Adapter.cpp** as follows:

```
Adapter::Adapter()
{
    controller = GpioController::GetDefault();
    pin = controller->OpenPin(PIN_NUMBER);           // Open GPIO 5
    pin->SetDriveMode(GpioPinDriveMode::Input);      // Set the IO direction as output

    this->vendor = L"Custom Vendor";
    this->adapterName = L"Custom Adapter";
    this->version = L"1.0.0.1";
}
```

Now, modify the ***Initialize*** function as given in the following:

```
    uint32
    Adapter::Initialize()
    {
       // GPIO Device Descriptor: Static data for our device
       DEVICE_DESCRIPTOR gpioDeviceDesc;
       gpioDeviceDesc.Name         = deviceName;
       gpioDeviceDesc.VendorName   = vendorName;
       gpioDeviceDesc.Model        = modelName;
       gpioDeviceDesc.Version      = version;
       gpioDeviceDesc.SerialNumer  = serialNumber;
       gpioDeviceDesc.Description  = description;

       // Define GPIO Pin-5 as device property. Device contains properties
       AdapterProperty^ gpioPin_Property = ref new AdapterProperty(pinName);

       // Define and set GPIO Pin-5 value. Device contains properties that have one or
more values.
       pinValueData = static_cast<int>(pin->Read());
       AdapterValue^ valueAttr_Value = ref new AdapterValue(pinValueName, pinValueData);
       gpioPin_Property += valueAttr_Value;

       // Finally, put it all into a new device
       AdapterDevice^ gpioDevice = ref new AdapterDevice(&gpioDeviceDesc);
       gpioDevice->AddProperty(gpioPin_Property);
       devices.push_back(std::move(gpioDevice));

       return ERROR_SUCCESS;
    }
```

At this point when this application starts, the GPIO device will be created. However, it will not be on the AllJoyn bus yet. For that, we need to expose it from the adapter by returning the available devices and supporting the property query. To complete the process of exposing the ***GPIO Device***, modify the ***EnumDevices*** and the ***GetPropertyValue*** functions as given in the following:

```
     _Use_decl_annotations_
    uint32
    Adapter::EnumDevices(
        ENUM_DEVICES_OPTIONS Options,
        IAdapterDeviceVector^* DeviceListPtr,
        IAdapterIoRequest^* RequestPtr
        )
    {
        UNREFERENCED_PARAMETER(Options);
        UNREFERENCED_PARAMETER(RequestPtr);

        *DeviceListPtr = ref new BridgeRT:: AdapterDeviceVector(this->devices);

        return ERROR_SUCCESS;
    }
```

```
    _Use_decl_annotations_
    uint32
    Adapter::GetPropertyValue(
        IAdapterProperty^ Property,
        String^ AttributeName,
        IAdapterValue^* ValuePtr,
        IAdapterIoRequest^* RequestPtr
        )
    {
        UNREFERENCED_PARAMETER(RequestPtr);
        UNREFERENCED_PARAMETER(AttributeName);

        pinValueData = static_cast<int>(pin->Read());

        AdapterProperty^ adapterProperty = dynamic_cast<AdapterProperty^>(Property);
        AdapterValue^ attribute = adapterProperty->Attributes->GetAt(0);
        attribute->Data = pinValueData;

        *ValuePtr = attribute;

        return ERROR_SUCCESS;
    }
```

That is all for a basic GPIO pin device. At this point when this application runs, the GPIO pin will be seen on the AllJoyn bus. Whenever any AllJoyn Client Application polls the value of the pin, our ***AllJoyn Device System Bridge Application*** will read the value from the physical GPIO pin on the Raspberry Pi.

Step 5: Run the AllJoyn Explorer Application

When you run the ***AllJoyn Explorer Application*** in the same subnet with the ***AllJoyn Device System Bridge***, you should be able to see that the ***GPIO Device*** is discovered. It should be something like this:

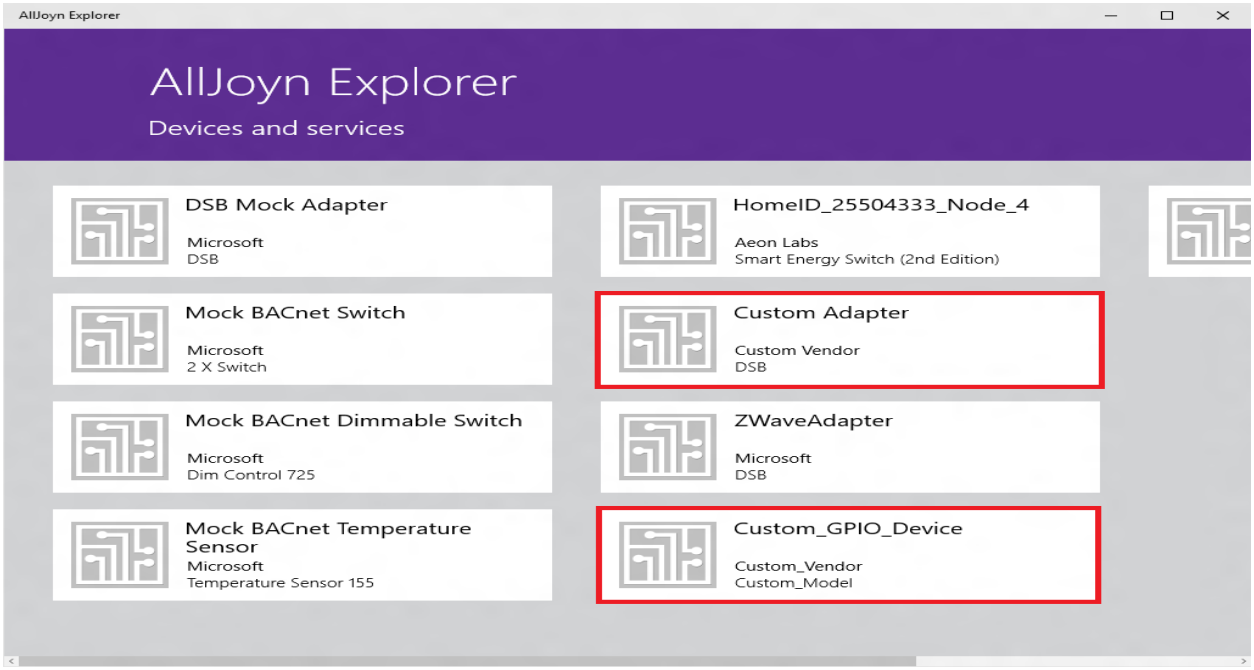

Figure 6: All devices and adapters discovered by the AllJoyn Explorer

Figure 7: The GPIO pin that is exposed to the AllJoyn bus as a custom device

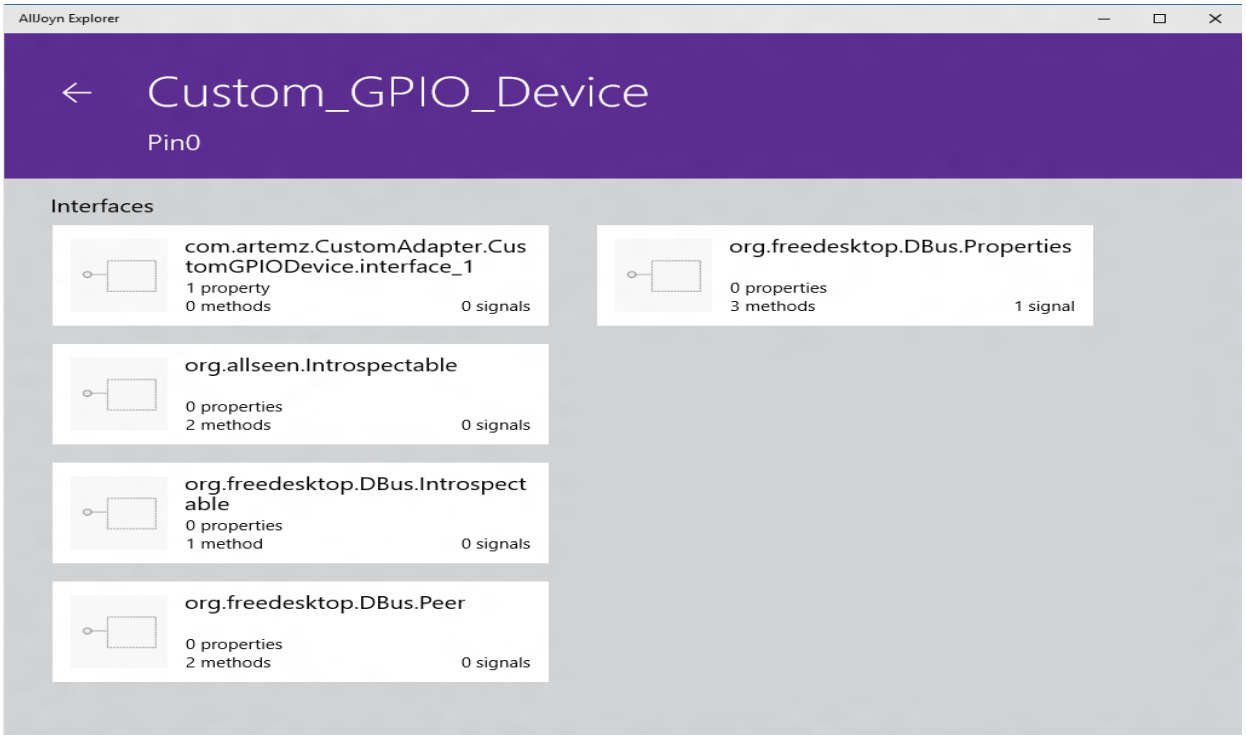

Figure 8: All the interfaces announced for the GPIO pin

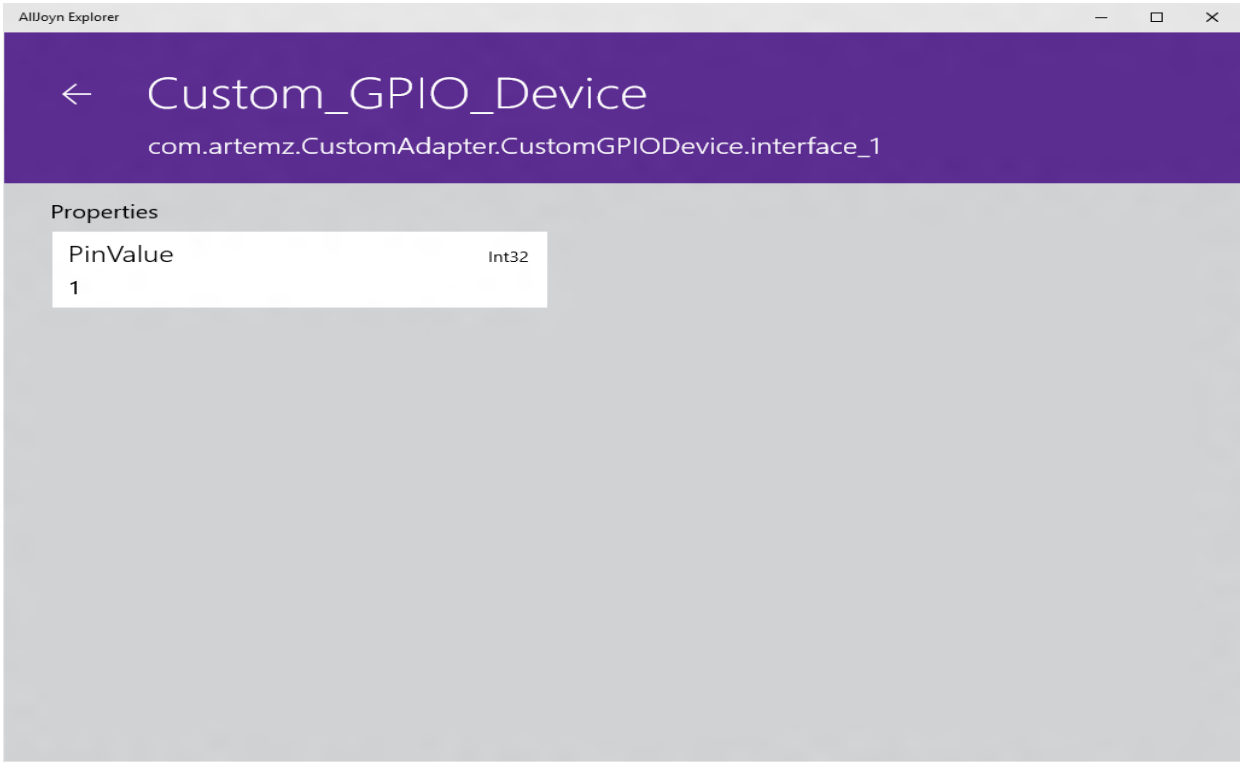

Figure 9: Current value read from the GPIO pin

EXTRA CREDIT: Signaling when the GPIO pin value changes

Suppose the applications on the AllJoyn bus do not want to poll the value of the GPIO pin but just to be notified when the GPIO pin value changes. For that, we need to add support for signals in our adapter. The contents below contain all the pieces needed to make the ***GPIO Device Sample*** notify the AllJoyn Consumer Applications.

Additions to **Adapter.h**:

```
    private:
        // GPIO pin value change event handler
        void pinValueChangedEventHandler(
        _In_ Windows::Devices::Gpio::GpioPin^ gpioPin,
        _In_ Windows::Devices::Gpio::GpioPinValueChangedEventArgs^ eventArgs);
```

Additions to **Adapter.cpp**:

```
    uint32
    Adapter::Initialize()
    {
        // GPIO Device Descriptor: Static data for our device
        DEVICE_DESCRIPTOR gpioDeviceDesc;
        gpioDeviceDesc.Name        = deviceName;
        gpioDeviceDesc.VendorName  = vendorName;
        gpioDeviceDesc.Model       = modelName;
        gpioDeviceDesc.Version     = version;
        gpioDeviceDesc.SerialNumer = serialNumber;
        gpioDeviceDesc.Description = description;

        // Define GPIO Pin-5 as device property. Device contains properties
        AdapterProperty^ gpioPin_Property = ref new AdapterProperty(pinName);

        // Define the GPIO Pin-5 value. Device has properties with attributes.
        pinValueData = static_cast<int>(pin->Read());
        AdapterValue^ valueAttr_Value = ref new AdapterValue(pinValueName, pinValueData);
        gpioPin_Property += valueAttr_Value;

        // Create Change_Of_Value signal for the Value Attribute
        AdapterSignal^ signal = ref new AdapterSignal(CHANGE_OF_VALUE_SIGNAL);
        signal += ref new AdapterValue(COV__PROPERTY_HANDLE, gpioPin_Property);
        signal += ref new AdapterValue(COV__ATTRIBUTE_HANDLE, valueAttr_Value);


        // Finally, put it all into a new device
        AdapterDevice^ gpioDevice = ref new AdapterDevice(&gpioDeviceDesc);
        gpioDevice->AddProperty(gpioPin_Property);
        gpioDevice->AddSignal(signal);
        devices.push_back(std::move(gpioDevice));

        // Pin Value Changed Event Handler
        pin->ValueChanged += ref new Windows::Foundation::TypedEventHandler<GpioPin ^,
GpioPinValueChangedEventArgs ^>(this, &Adapter::pinValueChangedEventHandler);

        return ERROR_SUCCESS;
    }
```

```
    _Use_decl_annotations_
    uint32
    Adapter::RegisterSignalListener(
        IAdapterSignal^ Signal,
        IAdapterSignalListener^ Listener,
        Object^ ListenerContext
        )
    {
        AutoLock sync(&lock, true);

        signalListeners.insert({ Signal->GetHashCode(),
                            SIGNAL_LISTENER_ENTRY(Signal, Listener, ListenerContext) });

        return ERROR_SUCCESS;
    }
```

```
    _Use_decl_annotations_
    uint32
    Adapter::NotifySignalListener(
        IAdapterSignal^ Signal
        )
    {
        AutoLock sync(&lock, true);

        auto handlerRange = signalListeners.equal_range(Signal->GetHashCode());

        std::vector<std::pair<int, SIGNAL_LISTENER_ENTRY>> handlers(handlerRange.first,
handlerRange.second);

        for (auto iter = handlers.begin(); iter != handlers.end(); ++iter)
        {
            IAdapterSignalListener^ listener = iter->second.Listener;
            Object^ listenerContext = iter->second.Context;
            listener->AdapterSignalHandler(Signal, listenerContext);
        }

        return ERROR_SUCCESS;
    }
```

```
    _Use_decl_annotations_
    void Adapter::pinValueChangedEventHandler(
        GpioPin^ gpioPin,
        GpioPinValueChangedEventArgs^ eventArgs)
    {
        AutoLock sync(&lock, true);

        // Notify registered listeners only when pin value actually changes.
        IAdapterSignal^ covSignal = devices.at(0)->Signals->GetAt(0);
        for (auto param : covSignal->Params)
        {
            if (param->Name == COV__ATTRIBUTE_HANDLE)
            {
                pinValueData = static_cast<int>(pin->Read());
                IAdapterValue^ valueAttr_Value = (AdapterValue^)param->Data;

                int previousPinValue = (int) valueAttr_Value->Data;
```

```
                if (previousPinValue != pinValueData)
                {
                    valueAttr_Value->Data = pinValueData;
                    NotifySignalListener(covSignal);
                }
            }
        }
    }
```