IoT Development Lab 2

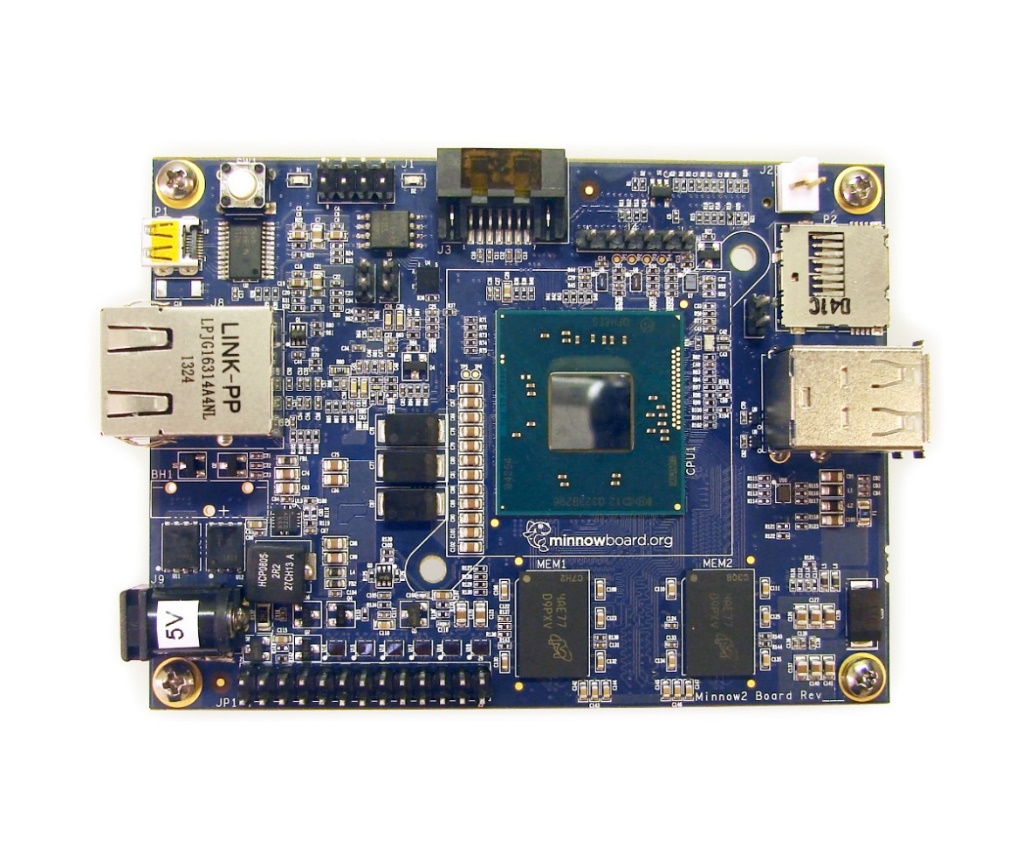
AllJoyn DSB GPIO

***Abstract*:** This is a fictional lab, intended to show the use of the template. The various styles in the WinHEC HOL template are used throughout. The lab does not imply HOL best practices nor appropriate length. The copyright page should be used intact in your lab, however. Also, do not move, resize, write text over or otherwise edit the logos on the cover page.

***Prerequisites*:** This lab is a hands-on look at the creation of the Universe. The lab assumes a working knowledge of Alchemy or general Windows magic. No prior knowledge of AllJoyn is required.

***Install and configure*:** You will need:

* A desktop PC running on Windows 10 RTM 10240
* A MinnowBoard Max development board running on Windows 10 IoT Core
* A keyboard, a mouse and a monitor for the MinnowBoard Max.



5V Power Adapter 🡪

Ethernet Cable 🡪

🡨 USB port x2

Micro-HDMI Cable 🡪

🡨 Micro-SD card

Power Button

↓

* An Ethernet cable to connect the desktop PC and MinnowBoard Max
* A DuPont Cable



Host  
(Development PC)

Ethernet Cable

Target  
(MinnowBoard Max)

The following table shows configuration information for these computers

|  |  |
| --- | --- |
| Development Computer Admin Account | Admin |
| Development Computer Admin Password |  |

|  |  |
| --- | --- |
| MinnowBoard Max Computer Name | minwinpc |
| MinnowBoard Max Admin Account | Administrator |
| MinnowBoard Max Admin Password | p@ssw0rd |

Copyright (c) 2015 Microsoft. All rights reserved.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

Contents

[Lab objective 4](#_Toc433744386)

[Exercise 1: Verify Hardware Setup 4](#_Toc433744387)

[Section 1: MinnowBoard Max Setting 4](#_Toc433744388)

[Section 2: Development PC and Software Settings 5](#_Toc433744389)

[Exercise 2: Create an Empty AllJoyn DSB App from a Template 5](#_Toc433744390)

[Section 1: Install the AllJoyn Device System Bridge Template 5](#_Toc433744391)

[Section 2: Create an AllJoyn DSB App Project 6](#_Toc433744392)

[Exercise 3: Create an AllJoyn DSB Adapter to a GPIO Pin 6](#_Toc433744393)

[Section 1: Add a reference to AdapterLib project to use the Windows IoT Extension SDK 6](#_Toc433744394)

[Section 2: Expose a GPIO Pin to the AllJoyn Bus 7](#_Toc433744395)

[Section 3: Deploy the AllJoyn DSB App onto MinnowBoard Max 10](#_Toc433744396)

[Exercise 4: Utilizing AllJoyn Explorer (AJX) 11](#_Toc433744397)

[Section 1: Run the AllJoyn Explorer Application 11](#_Toc433744398)

[[Optional] Exercise 5: Push Notification to AllJoyn Bus when the GPIO Pin Signal Changes 14](#_Toc433744399)

[Section 1: Add Signal Support to the Adapter 14](#_Toc433744400)

[About Signals 15](#_Toc433744401)

[Section 2: Observe the Change of GPIO Signal from AllJoyn Explorer 16](#_Toc433744402)

**Lab objective**

Participants will learn about how to develop an AllJoyn Device System Bridge (DSB) to a GPIO pin using a template, use AllJoyn Explorer (AJX) to interact with existing AllJoyn device in the network, to signal notification to AllJoyn consumer application when device signal changes. The tutorials for this lab will be presented as hands-on tasks that have been grouped into the following sections:

* Verify the hardware setup
* Create and deploy an AllJoyn Device System Bridge (DSB) app from a template
* Create an AllJoyn Device System Bridge Adapter to the GPIO pin
* Utilize AllJoyn Explore to Interact with AllJoyn Devices
* Add Support to Signal GPIO Pin Value Changes

**Exercise 1: Verify Hardware Setup**

In this exercise, you will make sure that your Desktop computer and the development board (MinnowBoard Max) have been set up with the files and software tools that you need for this hands-on lab.

Section 1: MinnowBoard Max Setting

In this section you will verify that your development board (MinnowBoard Max) is set up correctly for this lab.

**On MinnowBoard Max**

1. Make sure that the MinnowBoard Max is powered up and running, and you can see the Windows 10 IoT Core Default IoT App displayed on the monitor.
2. Check that there is an Ethernet cable connecting your MinnowBoard Max to your Dev PC.
3. Record the IP Address on the Default IoT App.



MinnowBoard Max IP Address = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

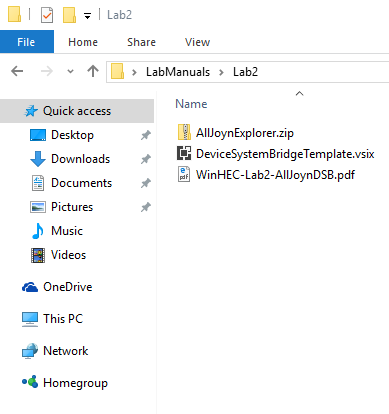
1. There is no wiring on the MinnowBoard Max. We will use the DuPont Cable at the last exercise (optional). Please remove the cable from the dev board for now.

Section 2: Development PC and Software Settings

In this section you will verify if your development PC is ready to create the AllJoyn Device System Bridge app.

**On Development Computer**

1. Make sure that your Dev PC is booted up and running on Window 10.
2. Launch **WindowsIoTCoreWatcher**, verify that your IoT device’s IP Address in the watcher matches the IP Address displayed on the Default IoT App.
3. On the desktop, double-click to open the **LabManuals\Lab2** folder, then verify that it contains files as shown in the following image.



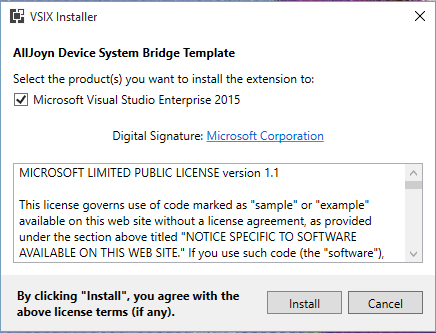
1. Close the folder.

**Exercise 2: Create an Empty AllJoyn DSB App from a Template**

From now, the rest of exercises will be performed **on the Development PC** unless stated otherwise. In this exercise, you will create an empty AllJoyn DSB App from the AllJoyn DSB App Project template.

## Section 1: Install the AllJoyn Device System Bridge Template

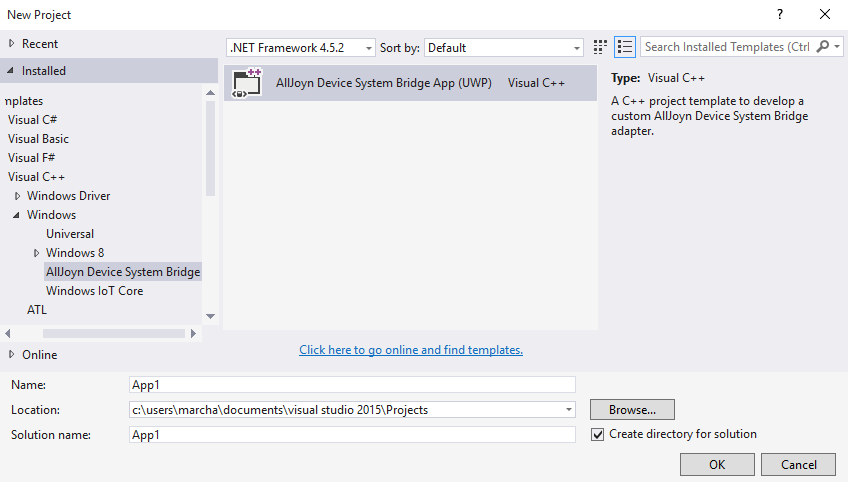
In this section, you will install AllJoyn DSB App Project template, a Visual Studio extension that enables developers to create an AllJoyn Device System Bridge App project.

1. Navigate to **LabManuals\Lab2** folder on Desktop.
2. Double-click on the **DeviceSystemBridgeTemplate.vsix** file to install the extension.
3. Click **Install** to install the extension. 

Section 2: Create an AllJoyn DSB App Project

In this section, you will create a new AllJoyn DSB App Project from the template.

1. Launch **Visual Studio 2015**, click **File** > **New** > **Project**.
2. In the New Project dialog box, click **Installed** > **Visual C++** > **Windows** > **AllJoyn Device System Bridge** to create a new AllJoyn Device System Bridge App project as given in the following:



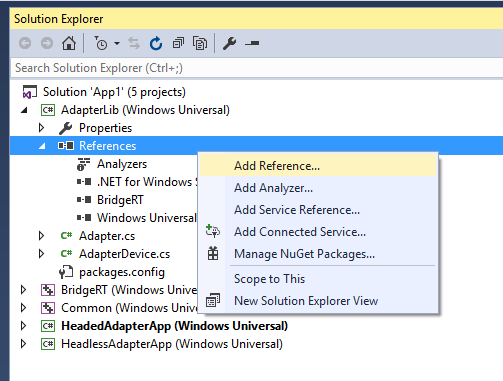
# Exercise 3: Create an AllJoyn DSB Adapter to a GPIO Pin

In this exercise, you will add the necessary code to the AllJoyn Device System Bridge (DSB) Adapter to GPIO Pin 5 by exposing the GPIO pin to the AllJoyn Bus.

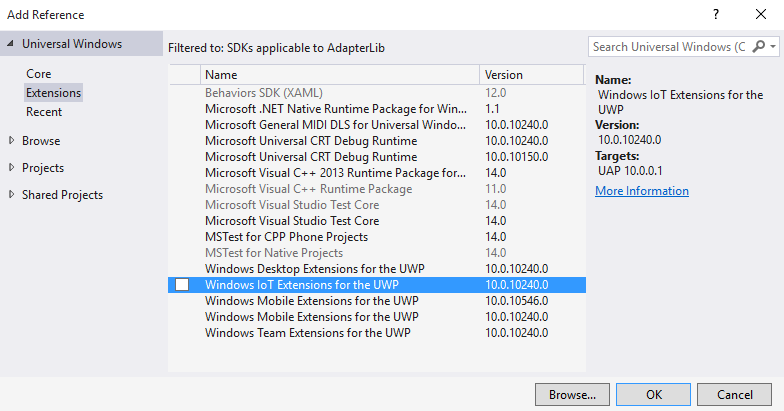
## Section 1: Add a reference to AdapterLib project to use the Windows IoT Extension SDK

In this section you will add a reference to the AdapterLib project to use the Windows IoT Extension SDK, which is required to use the Windows::Devices::Gpio API.

1. In the **Solution explorer**, locate the Adapter Lib project. Expand this project.
2. Right click “References” and select “Add Reference…”



1. On the left-hand side of the Reference Manager window, select “Extensions” under Windows Universal. Locate the latest version of the **Windows IoT Extension SDK** in the list and check the box its left to select this SDK.

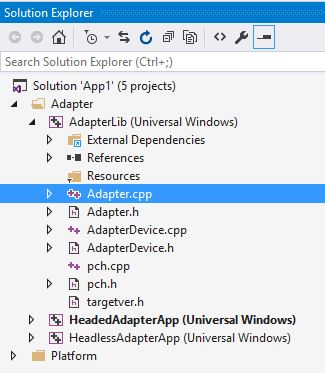


1. Click “OK”.

## Section 2: Expose a GPIO Pin to the AllJoyn Bus

In this section you will add some code to expose a GPIO Pin to the AllJoyn Bus.

1. Open the **Adapter.cpp** file in the **AdapterLib** project.



1. Modify **Adapter.cpp** by inserting the following block of text after the last “using” statement and before “namespace AdapterLib {“. This block of text declares the GPIO device properties.

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 pinValue = -1;

GpioController^ controller;

GpioPin^ pin;

1. In order to expose the GPIO Device to the AllJoyn Bus, we need to create a corresponding Bridge Device (IAdapterDevice) instance. Add the following three lines to the **Adapter()** constructor in the **AdapterLib** project’s **Adapter.cpp** file:

Adapter::Adapter()

{

...

controller = GpioController::GetDefault();

pin = controller->OpenPin(PIN\_NUMBER);

pin->SetDriveMode(GpioPinDriveMode::Input);

}

1. Modify the **Initialize ()** function of Bridge Device (IAdapterDevice) instance as shown below:

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.

AdapterProperty^ gpioPin\_Property = ref new AdapterProperty(pinName, "");

// Define and set GPIO Pin-5 value attribute.

pinValue = static\_cast<int>(pin->Read());

Platform::Object^ pinValueData = Windows::Foundation::PropertyValue::CreateInt32(pinValue);

AdapterAttribute^ gpioPin\_valueAttr = ref new AdapterAttribute(

pinValueName,

BridgeRT::E\_ACCESS\_TYPE::ACCESS\_READ,

pinValueData

);

gpioPin\_valueAttr->COVBehavior = BridgeRT::SignalBehavior::Always;

gpioPin\_Property += gpioPin\_valueAttr;

// 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;

}

1. Next, add highlighted code into the **GetPropertyValue()** function to read pin values:

\_Use\_decl\_annotations\_

uint32

Adapter::GetPropertyValue(

IAdapterProperty^ Property,

String^ AttributeName,

IAdapterValue^\* ValuePtr,

IAdapterIoRequest^\* RequestPtr

)

{

if (RequestPtr != nullptr)

{

\*RequestPtr = nullptr;

}

// sanity check

AdapterProperty^ tempProperty = dynamic\_cast<AdapterProperty^>(Property);

if (ValuePtr == nullptr ||

tempProperty == nullptr ||

tempProperty->Attributes == nullptr)

{

return ERROR\_INVALID\_PARAMETER;

}

// find corresponding attribute

\*ValuePtr = nullptr;

for (auto attribute : tempProperty->Attributes)

{

if (attribute->Value != nullptr &&

attribute->Value->Name == AttributeName)

{

// Read Pin Value

pinValue = static\_cast<int>(pin->Read());

attribute->Value->Data = Windows::Foundation::PropertyValue::CreateInt32(pinValue);

\*ValuePtr = attribute->Value;

return ERROR\_SUCCESS;

}

}

return ERROR\_INVALID\_HANDLE;

}

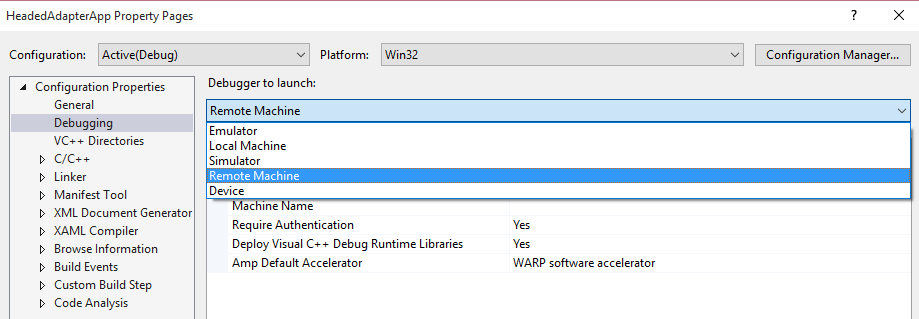
## Section 3: Deploy the AllJoyn DSB App onto MinnowBoard Max

In this section you will configure debug settings and deploy the AllJoyn DSB App onto Windows 10 IoT Core using Visual Studio 2015.

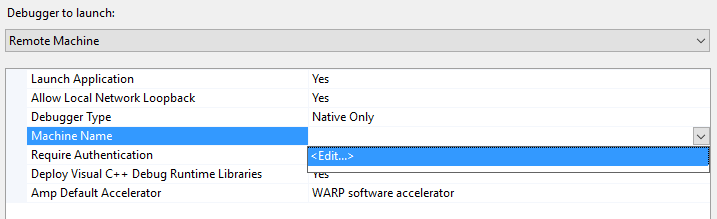
1. Check Solution Configuration is set to **Debug** and Platform is set to **x86** in Debugging toolbar.



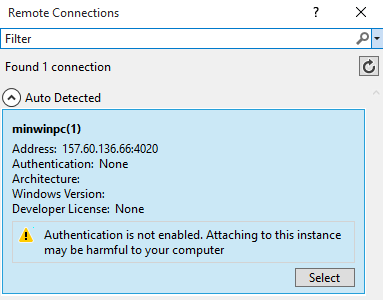
1. In **Solution Explorer** windows, right-click on the **HeadedAdapterApp** project file and select **Properties** > **Configuration Properties** > **Debugging**.
2. Under **Debugger to Launch**, select **Remote Machine** for Target device.



1. For **Machine Name**, Select **<Edit…>**. Remote Connections window would pop-up.



1. Select **minwinpc**. Check that the IP Address is correct. Click **Select** to close the window.



1. Select **Apply** then **OK** to close the properties window.
2. Press **F5** to deploy the App. You MBM screen should turn black, with numbers appear on the top corners.

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 MinnowBoard Max.

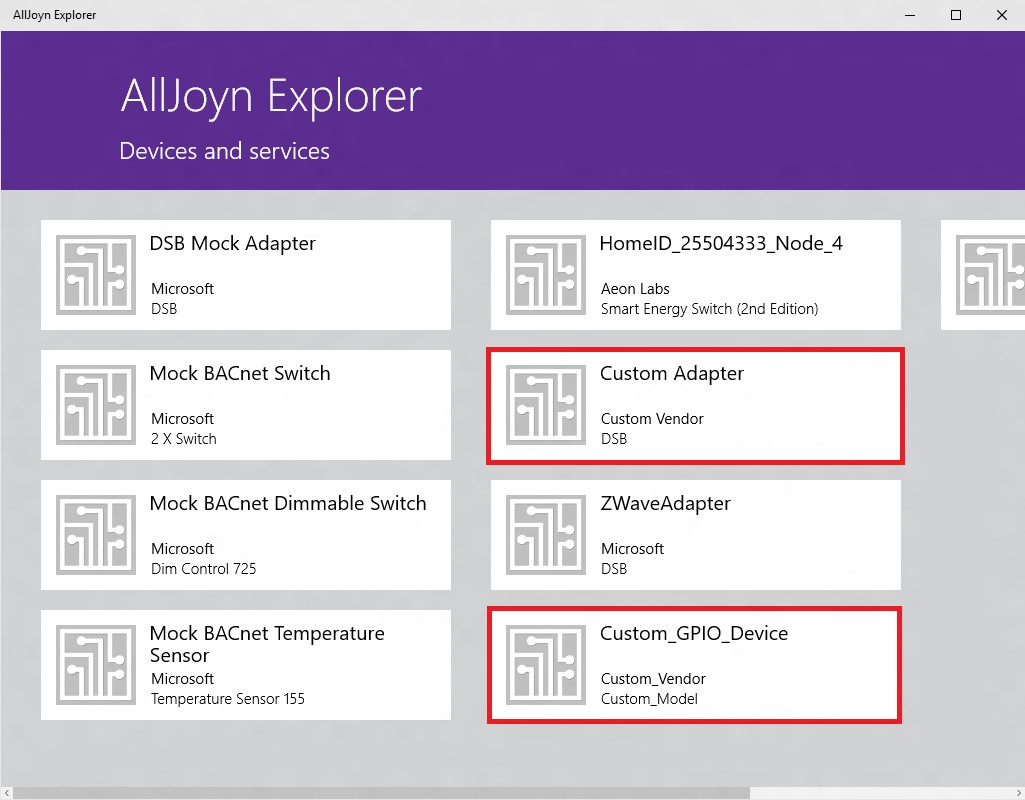
# Exercise 4: Utilizing AllJoyn Explorer (AJX)

In this exercise, you will use AllJoyn Explorer Application, a tool that can browse and interact with services exposed on the AllJoyn bus, to explore AllJoyn Device Service Bridge (DSB) we created for the GPIO pin in Section 1.

## Section 1: Run the AllJoyn Explorer Application

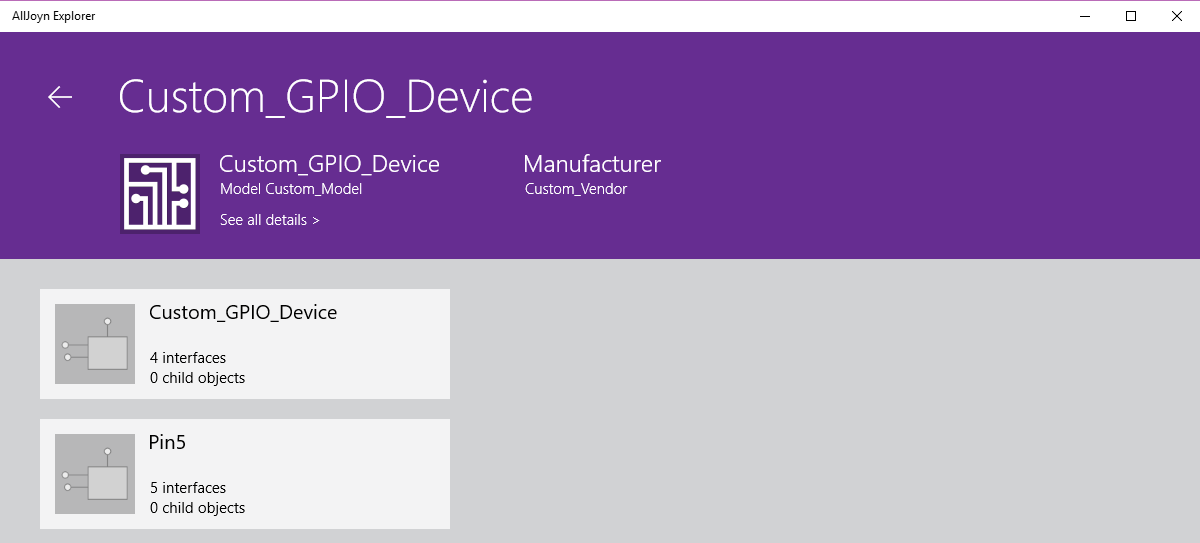
In this section you will run **AllJoyn Explorer** to discover the GPIO device.

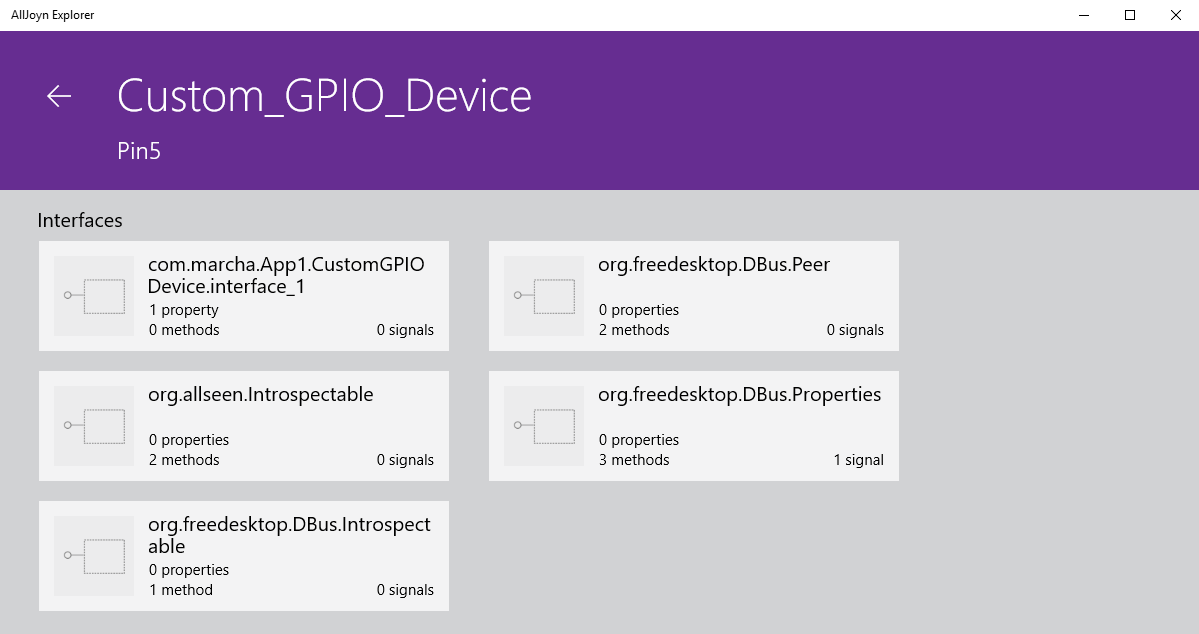
1. Launch **AllJoyn Explorer** by searching for “AllJoyn Explorer”in the search bar next to the Start Menu button at the button-left of the desktop.
2. Locate two nodes in the list - the “Custom Adapter” and “Custom\_GPIO\_Device” in the list of exposed Devices and Services.



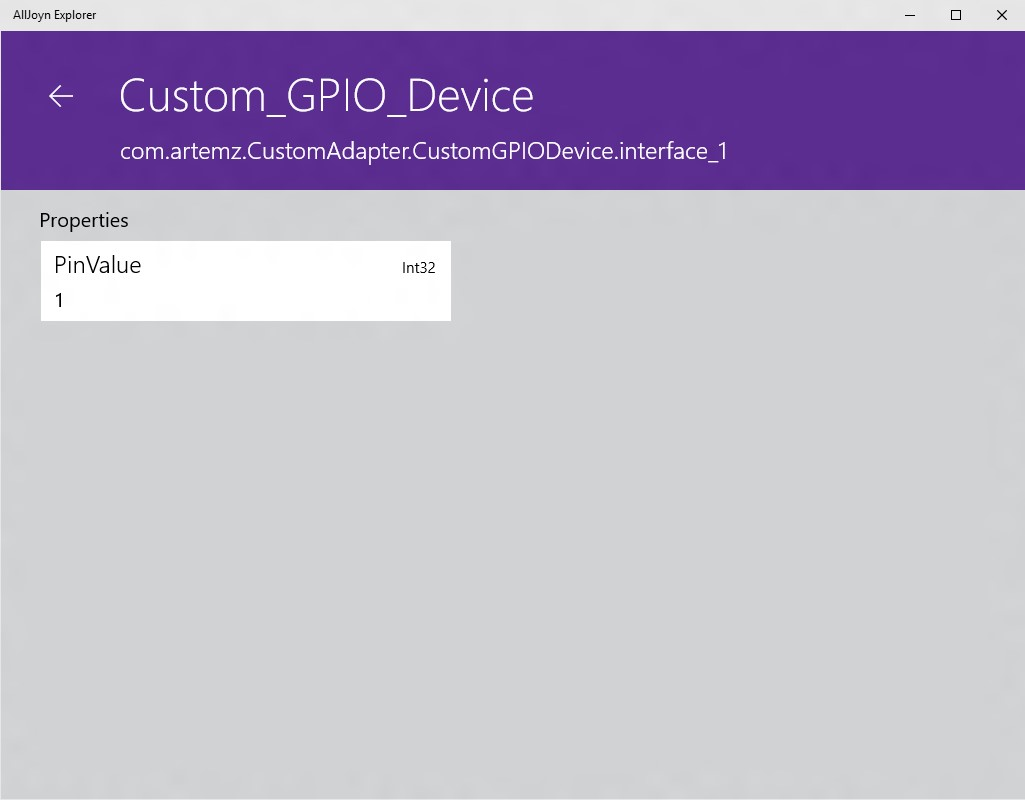
NOTE: Your adapter name may be different than the one shown below. By default, the adapter name is the name of the project you’ve created. For example, if your project name is DsbAdapter, the name of the adapter node will be “DsbAdapter” instead of “Custom Adapter”.

1. Find **Custom\_GPIO\_Device** from the list. Select **Pin5**. Observe the interfaces announced for this GPIO pin.





1. Click any interface to view its properties. 1.Selecting an interface allows you to view its properties; You should observe a single interface property, with PinValue equal to 1



# [Optional] Exercise 5: Push Notification to AllJoyn Bus when the GPIO Pin Signal Changes

In this exercise, you will see how to make the DSB Adapter notify the AllJoyn Consumer Applications to signal GPIO pin value changes. Sometimes 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.

## Section 1: Add Signal Support to the Adapter

In this section you will add support for signals to the adapter by adding an event handler.

1. Open **Adapter.h**
2. Add the following code to declare an event handler when GPIO pin value change occurs.

private:

// GPIO pin value change event handler

void pinValueChangedEventHandler(

\_In\_ Windows::Devices::Gpio::GpioPin^ gpioPin,

\_In\_ Windows::Devices::Gpio::GpioPinValueChangedEventArgs^ eventArgs);

1. Open **Adapter.cpp**
2. Add the definiation of the pinValueChangedEventHandler:

\_Use\_decl\_annotations\_

void Adapter::pinValueChangedEventHandler(

GpioPin^ gpioPin,

GpioPinValueChangedEventArgs^ eventArgs

)

{

AutoLock sync(&lock, true);

IAdapterSignal^ covSignal = devices.at(0)->Signals->GetAt(0);

for (auto param : covSignal->Params)

{

if (param->Name == Constants::COV\_\_ATTRIBUTE\_HANDLE)

{

pinValue = static\_cast<int>(pin->Read());

IAdapterValue^ valueAttr\_Value = (AdapterValue^)param->Data;

int previousPinValue = (int)valueAttr\_Value->Data;

// Notify registered listeners only when pin value actually changes.

if (previousPinValue != pinValue)

{

valueAttr\_Value->Data = pinValue;

NotifySignalListener(covSignal);

}

}

}

}

1. Modify the **Initialize()** function of the adapter by creating CHANGE OF VALUE signal for the Value Attribute and initialize the pin value change event handler. See “About Signals” below for a brief explanation of CHANGE OF VALUE signal.

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 attribute. Device contains properties that have one or more attributes.

pinValue = static\_cast<int>(pin->Read());

Platform::Object^ pinValueData = Windows::Foundation::PropertyValue::CreateInt32(pinValue);

AdapterAttribute^ gpioPin\_valueAttr = ref new AdapterAttribute(

pinValueName,

BridgeRT::E\_ACCESS\_TYPE::ACCESS\_READ,

pinValueData

);

gpioPin\_valueAttr->COVBehavior = BridgeRT::SignalBehavior::Always;

gpioPin\_Property += gpioPin\_valueAttr;

// Create Change\_Of\_Value signal for the Value Attribute

AdapterSignal^ signal = ref new AdapterSignal(Constants::CHANGE\_OF\_VALUE\_SIGNAL);

signal += ref new AdapterValue(Constants::COV\_\_PROPERTY\_HANDLE, gpioPin\_Property);

signal += ref new AdapterValue(Constants::COV\_\_ATTRIBUTE\_HANDLE, gpioPin\_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;

}

1. Press **F5** to deploy the App.

### **About Signals**

In the AllJoyn Device System Bridge, we have 3 predefined signals DEVICE ARRIVAL, DEVICE REMOVAL and CHANGE OF VALUE signals.

DEVICE ARRIVAL signal will be fired when a new device arrives to join to the AllJoyn network. To define the signal, you need to create an instance of IAdapterSignal with predefined constant signal name Constants::DEVICE\_ARRIVAL\_SIGNAL and a handle to the device (IAdapterDevice^) as signal parameter. Use predefined parameter name Constants::DEVICE\_ARRIVAL\_\_DEVICE\_HANDLE. This signal is associated with the Adapter.

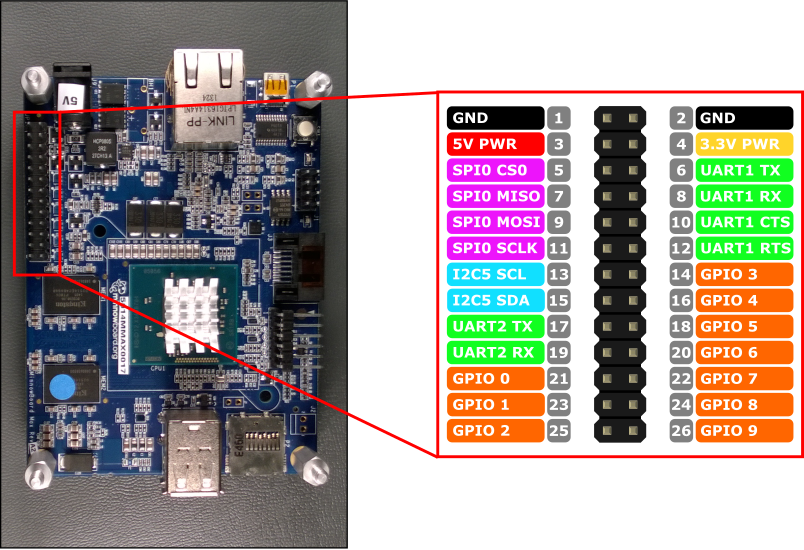
DEVICE REMOVAL signal will be fired when a device leaves the network. To define the signal, you need to create an instance of IAdapterSignal with predefined constant signal name Constants::DEVICE\_REMOVAL\_SIGNAL and a handle to the device (IAdapterDevice^) as signal parameter. Use predefined parameter name Constants::DEVICE\_REMOVAL\_\_DEVICE\_HANDLE. This signal is associated with the Adapter.

CHANGE OF VALUE signal will be fired when a property attribute value of a device changes. To define the signal, you need to create an instance of IAdapterSignal with predefined constant signal name Constants::CHANGE\_OF\_VALUE\_SIGNAL, a handle to the regarding property (IAdapterProperty^) as one of the signal parameters and a handle to the regarding property attribute (IAdapterValue^) as the other signal parameter. Use predefined parameter names Constants::COV\_\_PROPERTY\_HANDLE and Constants::COV\_\_ATTRIBUTE\_HANDLE. This signal is associated with the corresponding AdapterDevice.

Aside from the predefined ones, signals with custom name and parameters could be defined. Whenever these signals are fired, they will be sent to the AllJoyn Bus.

## Section 2: Observe the Change of GPIO Signal from AllJoyn Explorer

In this section you will physically change signals from GPIO Pin 5 and observe the change from AllJoyn Explorer. Refer to MinnowBoard Max Pin Mappping (below) for pin definitions.



**MinnowBoard Max Pin Mappings**

**On MinnowBoard Max**

1. Power off the MinnowBoard Max.
2. Set GPIO Pin 5 to 0: Use a DuPont cable to connect **Pin 18** to **Pin 2** (GPIO 5 to GND).
3. Power on the MinnowBoard Max
4. After MinnowBoard is power back on, run the AllJoyn DSB app. You may user web browser to start off the app, or use Visual Studio F5 to deploy the app.

**On Development Computer**

1. Launch **AllJoyn Explorer**.
2. Select **Custom\_GPIO\_Device** from the list.
3. Select **Pin5**.
4. View the value of Pin 5 by selecting **com.customverdor.CustomeAdapter.CustomeGPIODevice.interface\_1**. The PinValue property should be 0.

**On MinnowBoard Max**

1. Set GPIO Pin 5 to 1: Use a DuPont cable to connect **Pin 18** to **Pin 4** (GPIO 5 to 3.3V).

**On Development Computer**

1. In AllJoyn Explorer, the PinValue property should now be changed to 1.