Throughout this course we have created multiple levels for each exercise. This should allow everybody, from a starter to a more seasoned programmer, to find a challenge in these exercises.

We will provide an existing Visual Studio 2019 solution that can be used during this course that will have snippets of code combined with a “task list” to work on.

However, if you prefer a clean solution you can start a new solution following the guidelines in the supplied document “***4-Series and VC-4 C# Development Instructions”***

# Exercise 1 – Device Registration

This first Lab session starts with one of the most important aspect of programming for a Crestron control system: device registration.

For the sake of time, we are not going to see examples for all different types of devices. Instead, as a reference, we will list important specifics for different device types below.

* All devices require registration before you can use them.

The exceptions to this rule are:

* + System Monitor
  + Internal EX Gateway
* To register a port on the control system you will have to register the individual port.

The exception to this rule is:

* + InfraRed. You register the full slot and then use individual ports.
* InfinetEX
  + To register an InfinetEX device on an internal gateway, you register each device individually
  + To register an InfinetEX device on an external gateway, you add the device to the gateway and then register the gateway
* For a DigitalMedia switcher and DMPS3, you add all cards and endpoints to the DM switcher, and then register the switcher itself.
  + For endpoints, use the constructor with IPID and DMInput
* DMPS3-4K-x50/350-C and DM-CPU3 systems
  + Register endpoint without IPID

dmTx201 = new DmTx201C(SwitcherInputs[9] as DMInput);

* For a standalone DM endpoint, register the endpoint with IPID and CrestronControlSystem

A best practice before registering a device is to check if the control system supports that specific form of communication (if (this.SupportsEthernet), if (this.SupportsCresnet), etc) before attempting to register.

Also, always check registration results (eDeviceRegistrationUnRegistrationResponse.Success) and act accordingly.

After registering a device, you can interact with them through the joins that you are familiar with in SIMPL Windows.

However, these joins are differently named, to match the correct type in C#. For instance, let’s look at a Touchpanel device.

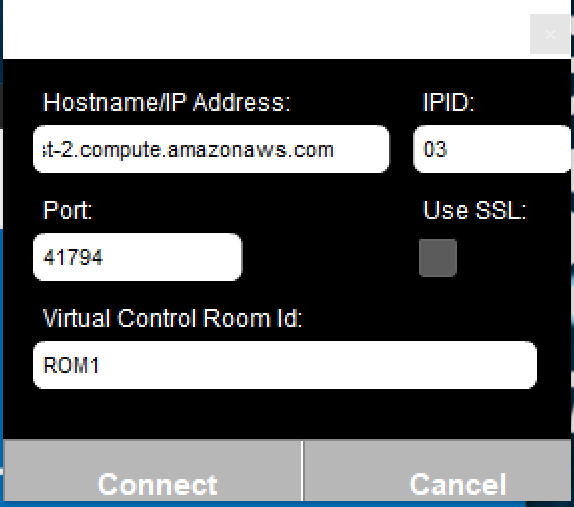
|  |  |
| --- | --- |
| **SIMPL Windows** | **C#** |
| Digital In/Out | BooleanInput / BooleanOutput |
| Analog In/Out | UShortInput / UShortOutput |
| Serial In/Out | StringInput / StringOutput |

Because a device can have multiple inputs/outputs, all of these signals are in their own separate **collection**. You will also have to reference the correct SigType.

So, to trigger Digital Join 1 to be 1/high on a Touchpanel device in C#, you could do something like this:

tp01.BooleanInput[1].BoolValue = true;

When using the provided Xpanel to test functionality, make sure you point to the hostname or IP address of the supplied VC-4 instance in combination with the ROOMID by selecting “Options” 🡪 “Host Settings”, or pressing Ctrl+H



For this exercise we are going to register an Xpanel that we will be using through the rest of this course.

As mentioned, there are 3 different levels. We expect everyone to finish Level 1, if you have additional time (and you still feel like it) you can continue with Level 2 and 3.

**Level 1**

For this level, we want you to create an Xpanel, register and use an Xpanel.

Register the necessary event handlers (at least SigChange and OnlineStatusChange), and lastly register the device on IPID 0x03.

In the OnlineStatusChange event handler, we would like you to add functionality to determine if the panel is online or offline.

In the SigChange event handler, we have already programmed an empty switch/case that you will have to complete.

Functionality we are looking for:

* Feedback of **Digital Join 11** needs to be high (true) when the Xpanel is online.
* When **Digital Join 12** is pressed, send the text “Hello World” to **Serial join 11** on the Xpanel
* When **Digital Join 12** is released, clear the text (send an empty string “”)

Hint: Remember the explanation regarding joins from the previous page? Use that for this exercise.

**Level 2**

After finishing the exercise for Level 1, you may continue with Level 2.

The additional exercises for this level are:

* Change the behavior of **Digital Join 21** to resemble a Toggle.

When you press the button, enable feedback and keep it high. Also send “Hello World” to **Serial join 21**.

When you press the button again, disable feedback and clear the text on **Serial join 21**.

* After you have finished the toggle, change the behavior of **Digital Join 22, 23 and 24** to resemble an Interlock.

When you press **Digital Join 22**, send “Hello World” to **Serial join 21**. (Overwriting the text from the previous exercise)

When you press **Digital Join 23**, send “Hallo Wereld!” to **Serial join 21**.

When you press **Digital Join 24**, send “Hola Mundo” to **Serial join 21**.

Throughout these presses, only the button feedback of the button you pressed should be high (true)

Pressing **Digital Join 25** should clear the Interlock functionality, also clearing the feedback on **Digital Join 22, 23 and 24** (false) and clearing the text on **Serial join 21** (empty string “”)

**Level 3**

The last level for this exercise involves adding functionality for a slider to your code.

When dragging the slider with **Analog join 31** back and forth, you will have to display the value of the slider in percentage through **Analog join 32.**

We have also added a multi-mode button (**Analog join 33**). This button has 4 states:

* Black (default state)
* Green
* Yellow
* Red

When the slider with **Analog feedback join 31** has a value of 0%, the multi-mode button should show the default state. Between 1% and 33%, the button should show the green state.

When the slider has a value between 33% and 66%, the button should show the yellow state.

Lastly, when the slider has a value between 66% and 100%, the button should show the red state.

Additionally, with a press of **Digital Join 31** you should register another XPanel on a different IPID with all the same functionality as the first one. This involves making clever use of the available information in the SigChange event handler.

Pressing button 24 again should unregister that new XPanel from your code.

# Exercise 2 – Crestron Web Scripting & File handling

The second Lab session involves two topics:

* Crestron Web Scripting (CWS)
* File operations

We are going to use CWS to receive data from a 3rd party source through HTTP requests.

If these requests are **GET** requests, you can use your browser to try them and see the results in your browser.

When you want to execute anything other than a **GET** request (POST/PUT/etc) you will need a tool to do this. A good one is Postman: (<https://www.postman.com/>)

For the first exercise in this Lab session we will get data through CWS, write a response to the client and store received data in a local text file.

If you want to start with Level 2 or 3, please read the description of Level 1 below. It has some pointers to get the right CWS URL.

For writing to / reading from the log file, you can use *Directory.GetApplicationRootDirectory()* to get to the correct folder on your VC-4 instance.

Keep in mind that there is no SSH access, so there is no way to get to this file. If you want to read the contents of the file, follow the exercise in **Level 3**.

**Level 1**

For this level, we are registering an HTTP route called helloworld

When somebody posts a GET request to that route, we want to:

* Receive the data on the following HTTP route:

[http://[IpAddress]/VirtualControl/Rooms/[RoomID]/cws/helloworld/[YourText](http://[IpAddress]/VirtualControl/Rooms/%5bRoomID%5d/cws/helloworld/%5bYourText)]

[IpAddress] = IP address of your VC-4 instance

[RoomID] = ID of your room instance in VC-4

[YourText] = Your free-form text field

* Send the text from [YourText] to **Serial Join 11** on the Xpanel.
* Append the text from [YourText] to a local text file **User/logfile.txt** on a newline.
* Return the received text as a response to the HTTP request.

**Level 2**

Additionally, for the 2nd level, work on the following:

* Register a new **POST** route called holamundo

This request will receive a JSON payload in the body of the request.

*{"text": "Hello world"}*

* Parse the data in your preferred way and show only the value of the “text” key.

From the previous example, this would be *“Hello world”*

* Append the received text to **User/logfile.txt**
* If the state of **Digital Join 21** is high (true), send a JSON formatted response with payload

*{"button": true}*

* If the state of **Digital Join 21** is low (false), send a JSON formatted response with payload

*{"button": false}*

* Register an additional **GET** route called interlockstatus

When this request is triggered, there should be a response with this JSON payload

*{*

*"status": [{*

*"button": true*

*},*

*{*

*"button": false*

*},*

*{*

*"button": false*

*}*

*]*

*}*

Of course, the values need to match the current state of the Interlock

**Level 3**

The 3rd level for this exercise involves the slider on the Xpanel

* Register a **GET** route called getslider

This should return the current value of **Analog join 31** as a percentage.

{"value": "65%"}

* Register a **POST** route called postslider

This should take in a JSON payload that sets the value of **Analog join 31**

{"value": 50}

Also append this value to the local text file **logfile.txt**

* Register a **GET** route called log

Whenever this call is being triggered, the full contents of **logfile.txt** should be returned.