Throughout this course we have created two 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”***

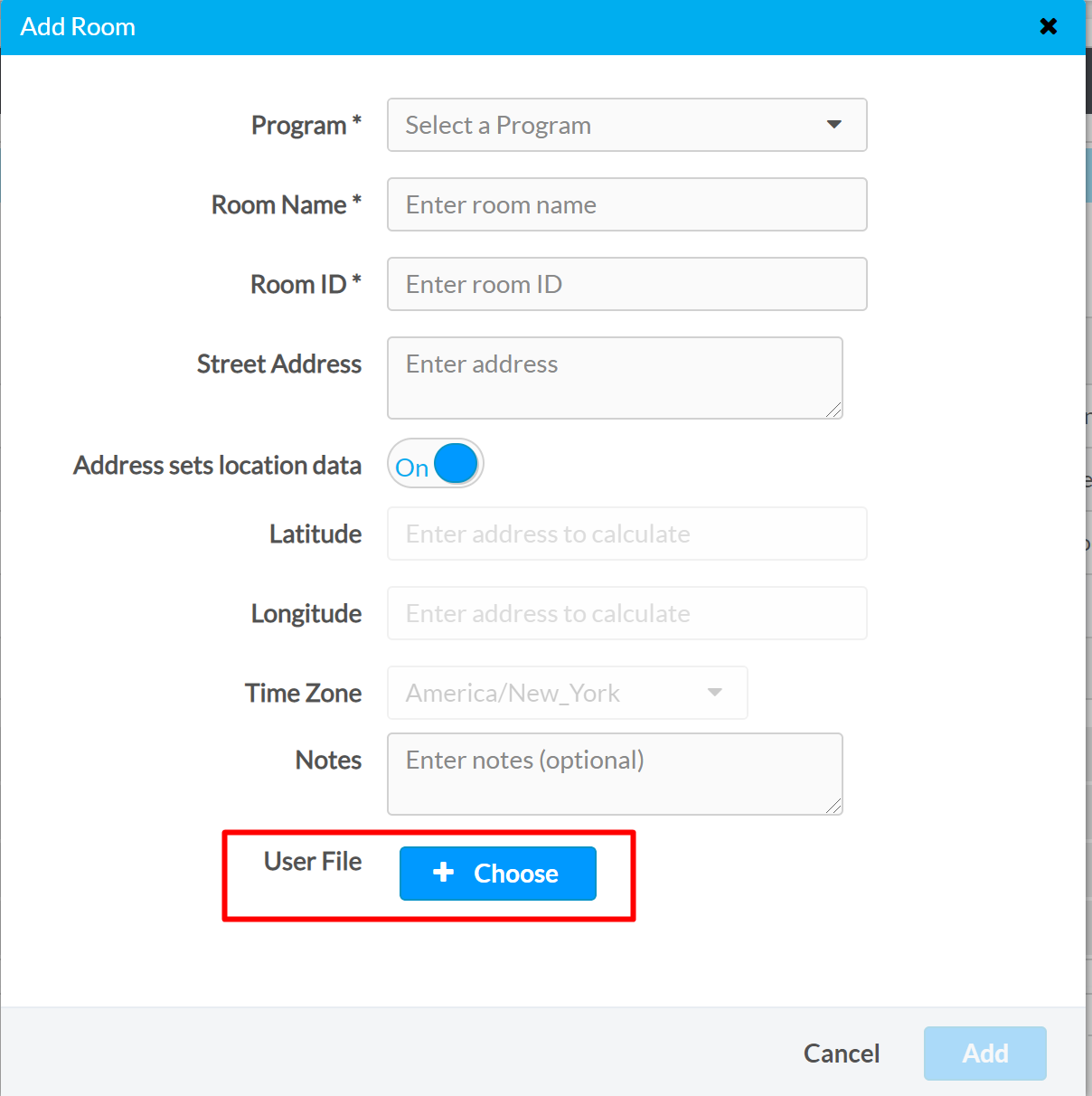
# Exercise 1 – Dynamic Registration

This first Lab will focus around the dynamic registration and configuration of devices.

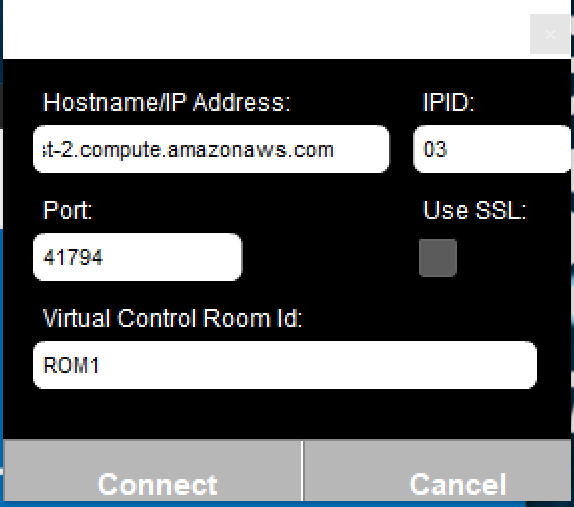
We have already set up a base solution the handles loading a configuration file. For anybody that has also followed the MSS-501/502 course the classes we have used to read in the config file will look very familiar.

The configuration file (*config.json*) is already provided in the **Documentation** folder. Please use this config file throughout the course.

Since we are dynamically loading this config file, don’t forget to add it as a **User File** when creating your room in VC-4:



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 exercise 1, we will be using a single “Management” class (SystemManager.cs) that will handle device registration as well as device configuration.

We have again provided an XPanel that can be used, that simply consists of two Dynamic List objects that we want to setup correctly based on the provided JSON config file.

**Level 1**

From the *SystemManager.cs* file, please implement the dynamic registration of the touchpanels as they have been defined in the configuration file.

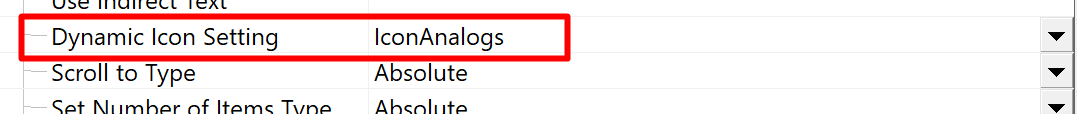
A lot of the heavy lifting is already done in the *TouchpanelUI.cs* file. You can create an instance of it in *SystemManager* to use it.

After you have registered the touchpanels, you will have to set them up.

Functionality we are looking for:

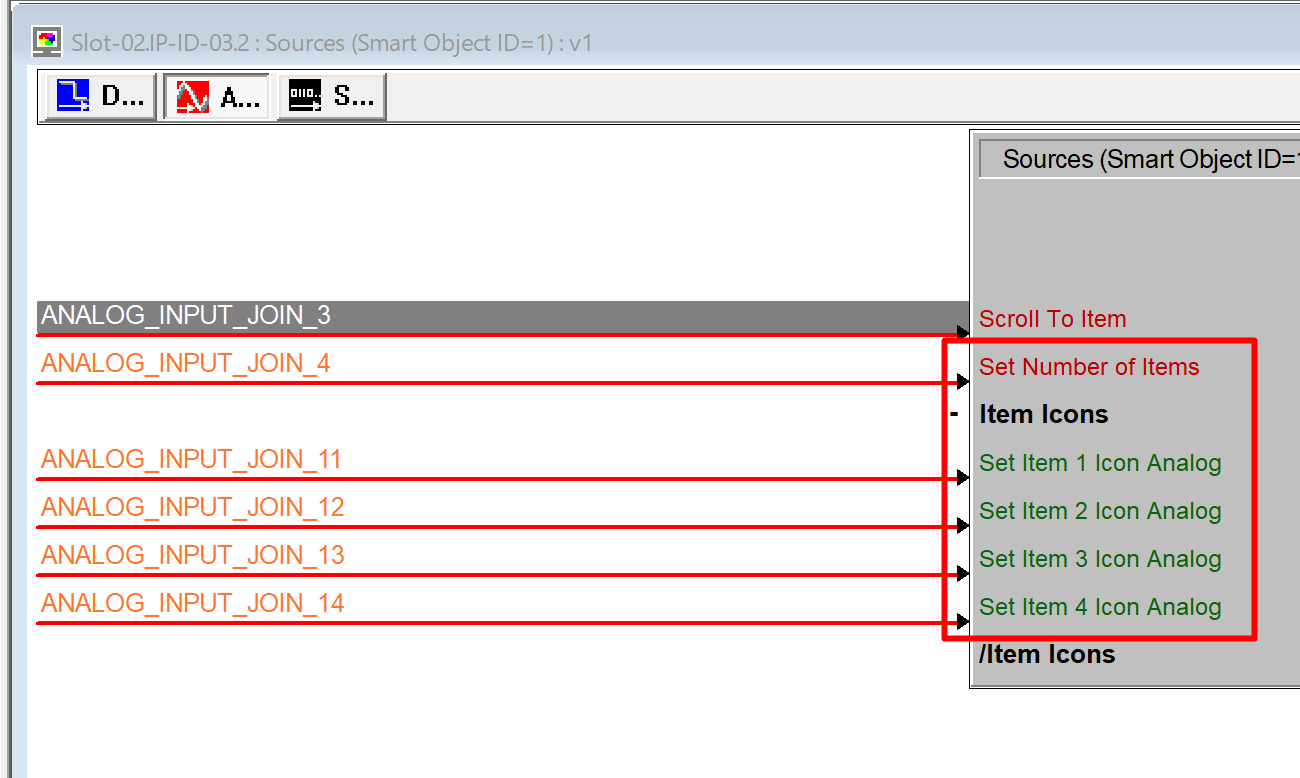
* There is currently one source defined in the config file. Expand the config file to add more sources.
* There is currently one destination defined in the config file. Expand the config file to add more destinations.
* Set up the two Smart Objects (Smart Object ID 1 & 2) to be dynamically configured using the information from the configuration file.
  + Change the size of the Dynamic List object to match the number of sources/destinations defined
  + Change the label for all sources/destinations to be dynamic *(you can use the “id” property from the configuration for this)*
  + Change the icon for all sources/destinations to be dynamic *(you can use the “id” property from the configuration for this)*

**Hint**: The icons can be set up using “Dynamic Icon Setting”



We have provided you a document called **“*Icons\_State\_ID”*** in the Documentation folder.

**Hint:** In SIMPL Windows, these are the properties you would use:



**Level 2**

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

The additional exercise for this level is to add another *subsystem* to the configuration file that will allow you to dynamically register DM-NVX devices.

Functionality we are looking for:

* A way to define which NVX device you want to register:
  + DM-NVX350(-C)
  + DM-NVX351(-C)
* Being able to dynamically load and register these devices to your system.

# Exercise 2 – RabbitMQ

For the second exercise, we will be setting up publisher / consumer using RabbitMQ.

For more information regarding RabbitMQ, you can use the following resources which are applicable to this exercise.

<https://www.rabbitmq.com/tutorials/tutorial-one-dotnet.html>

<https://www.rabbitmq.com/tutorials/amqp-concepts.html>

Please use the new solution that was provided for this second exercise

**Level 1**

We want you to use RabbitMQ to dynamically publish and consume message.

All the implementations for RabbitMQ have been done already in both the *SystemManager.cs* file as well as the *DmNvx.cs* file.

We did this to allow you to focus on implementing the routing itself, in stead of worrying about the specifics of the RabbitMQ implementation. However, please take the time to examine **how** RabbitMQ was implemented and see how you can leverage it to create a dynamic system.

The goal of this exercise is to create a way to make a route using the previously registered DM-NVX units.

The objective is:

* Create your own “protocol” and publish it to the correct Exchange

Keep in mind that by publishing something, all the subscribers will receive it.

In our example, this means that all instances of the *DmNvx.cs* class will receive the message.

(We have already defined the Exchange for you as a const in *SystemManager.cs*)

* In *DmNvx.cs* you will have to parse the previously defined “protocol” and make sure that only a specific DmNvx unit makes the switch. It’s up to the programmer to figure out a way to make a message unique!

We will make use of **Subscription Based routing f**or NVX. You can use XioRouting.VideoOut on the NVX device for this.

Assume that the order of the subscriptions on NVX match the order of the sources as they are defined in the configuration file.

**Note**

For this exercise, we have given you a baseline solution. However, you can make changes to the existing class files, or even add additional class files if you feel this is necessary.

**Level 2**

For this exercise we want you to do something like the previous exercise, but this time we consume messages from another program. In this example, this program will be another room on our VC-4 instance.

Your task is to create a second program that will be controlling lighting presets.

On the XPanel, we have created an additional Smart Object (ID 3: Light Presets) to be used for this.

The objective is:

* Create a new exchange in your main program that will publish message specifically for the lighting subsystem.
* Create a new program that subscribes to this exchange.
* From the main program, publish a message whenever you press any of the buttons on **Smart Object 3**
* Parse these messages in your new program. Since we don’t have any additional hardware connected, you may just print the parsed command to the ErrorLog.

**Note**

For this exercise, we have given you a baseline solution. However, you can make changes to the existing class files, or even add additional class files if you feel this is necessary.

**Extra credit**

For anybody looking for an additional challenge, run multiple instances of the same program on VC-4 where you can share all the sources and destinations (from the configuration) between those programs using RabbitMQ.