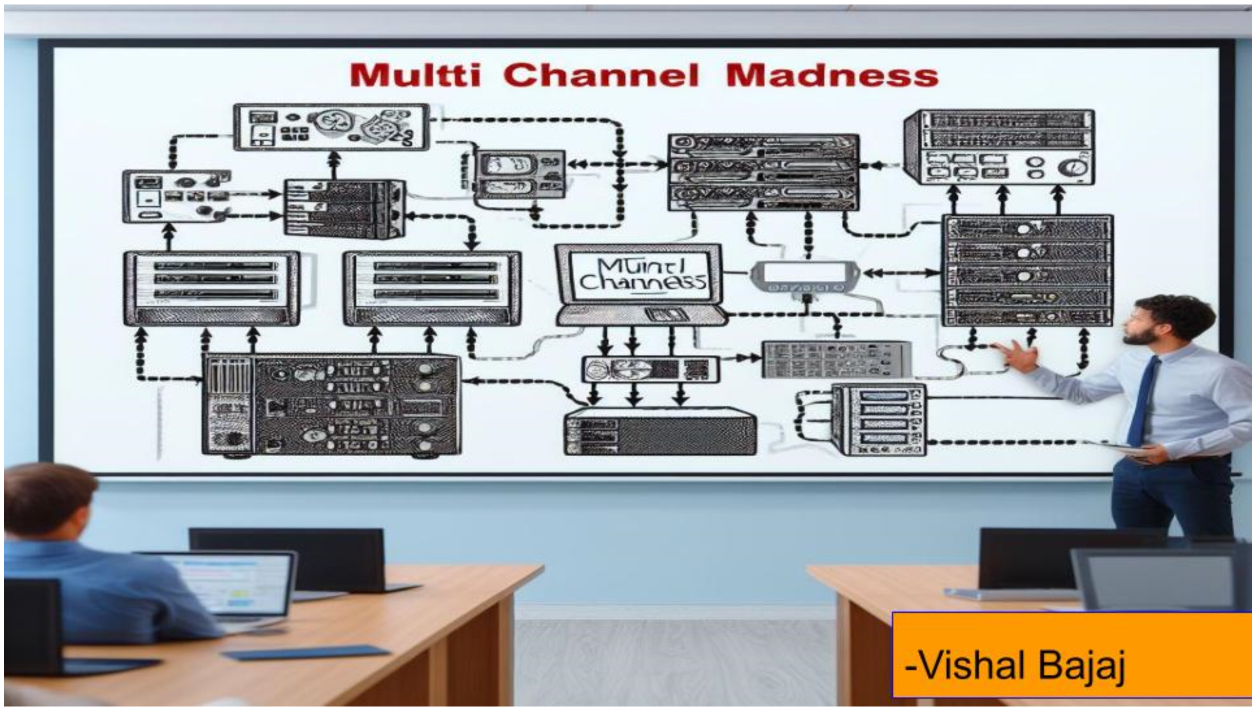


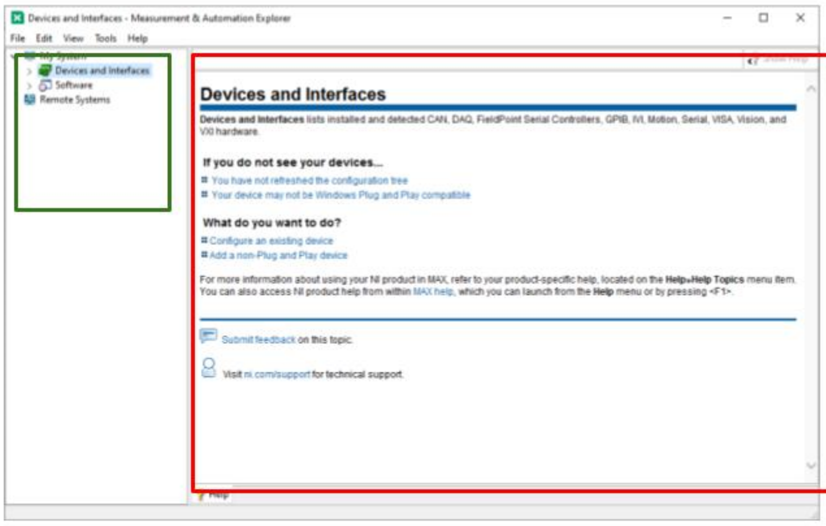
Often when building medium to large size systems we need to build a Configuration page to enable the user to configure multiple channels individually or as a group. Countless hours can be spent painstakingly navigating through configuration files and building the appropriate UI LabVIEW's Configuration Editor Framework (CEF) is here to change that. CEF is a powerful tool designed to simplify the process of creating and managing configuration settings for applications, particularly those involving multichannel or complex configurations.

Presenter Introduction

- Started with LV as a part of my final year thesis project controlling motion stages.
- 2+ years LV development experience.
- Worked on a multitude of small and medium size test system projects ranging from ground up development to cleaning up legacy code.

Hi everyone. My name’s Vishal. I've been working with LabVIEW for the last 2 + years now. I was introduced to LabVIEW through my final year thesis project where I was trying aimlessly to control 2 very expensive multi stage axes trying to align 2 optical fibers. Then during my second year I was formally introduced to LabVIEW through my first job, and am currently working for a semiconductor company in Brisbane.

Configuration Editor Framework (CEF)


Devices and Interfaces - Measurement & Automation Explorer
File Edit View Tools Help
My System
Devices and Interfaces
Software
Remote Systems
Help

Devices and Interfaces

Devices and Interfaces lists installed and detected CAN, DAQ, FieldPoint Serial Controllers, GPIB, IVI, Motion, Serial, VISA, Vision, and VXI hardware.

If you do not see your devices...

- You have not refreshed the configuration tree
- Your device may not be Windows Plug and Play compatible

What do you want to do?

- Configure an existing device
- Add a non-Plug and Play device

For more information about using your NI product in MAX, refer to your product-specific help, located on the **Help»Help Topics** menu item. You can also access NI product help from within MAX help, which you can launch from the **Help** menu or by pressing <F1>.

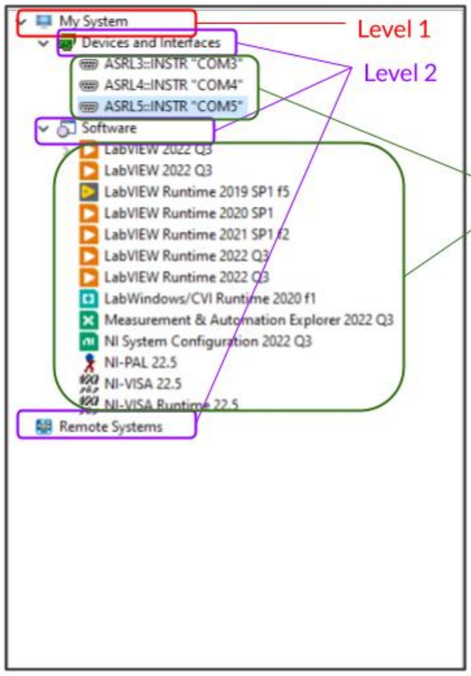
Submit feedback on this topic.

Visit ni.com/support for technical support.

After that brief introduction now let’s get into the meat of it. By show of hands can anyone tell me if they’ve used/heard of the Configuration Editor Framework before??
Configuration Editor Framework can be downloaded from VIPM. It is shipped with 2 templates. We’ll look at one of them later.

Now on my screen you can see the very familiar NI Max screen.
This is a Configuration Editor wherein we have a subpanel where the user Interface is inserted and we have a tree structure next to it where by pressing different tree Nodes we can insert a new user Interface into our subpanel. Each User Interface will have settings pertaining to its tree node.

The Configuration editor then stores the User input settings in its local memory called a “Repository” and ever so often it would then write these changes to a File, (the process of writing to file can also be prompted by the user). So now the next time the user opens the configuration Editor they can reload their saved configuration file and make edits to it.

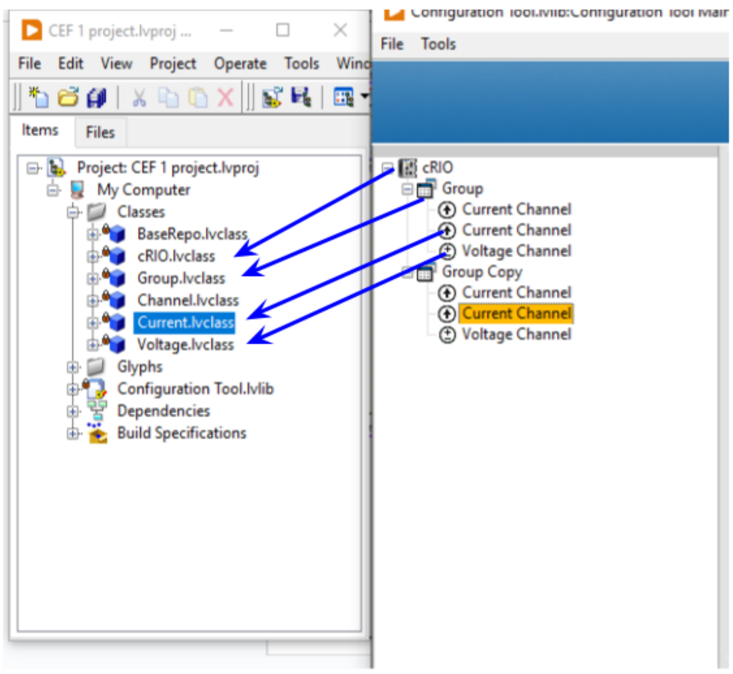


Tree Structures

Tree

Now let’s look at some features of Tree Structures and kind of get an understanding of what we want to achieve with a Configuration Editor. The first thing we can notice is that Tree Structures are hierarchical. In this example we have 3 levels of Nodes. Like so (click, click, click).
The tree structure will be the primary graphical interface we will use to navigate through the configuration.
Now if we look at the example program shipped with the CEF we can discover some more features of the Tree we are imagining. (Show Base example with Tree node features like

This is the shipped project template for the CEF. We start with a device called “the cRIO” .
Now I can right click and it gives me a shortcut menu which has an Add. Now I can add a subnode “Group ''. Then if I right click Group I get a different shortcut menu wherein I can “Add” a different set of subnodes “Current Channel” and a “Voltage Channel”. But I can also “Remove” this Node and duplicate this Node. Now let's quickly go ahead and try all those functions.
If I also quickly look at my file functions here. I can Save a configuration, Open a configuration and Exit. How cool is that all this functionality is pre shipped and we are only going to build further on it.



Now let's look at the project .

- We can see that each node in your tree corresponds to a Class.
- Each of these Classes are DVR reference Classes.

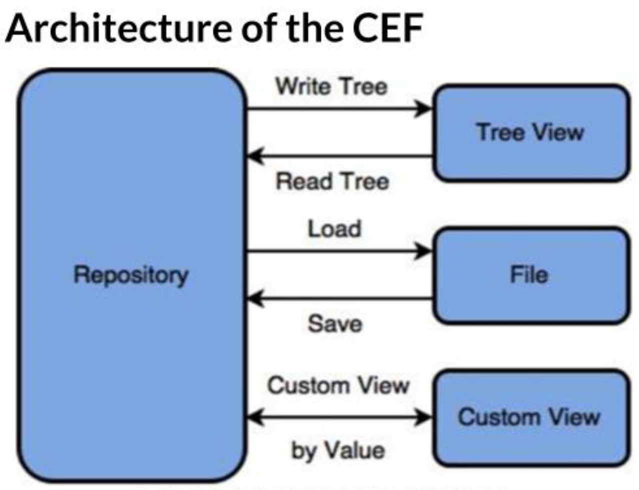
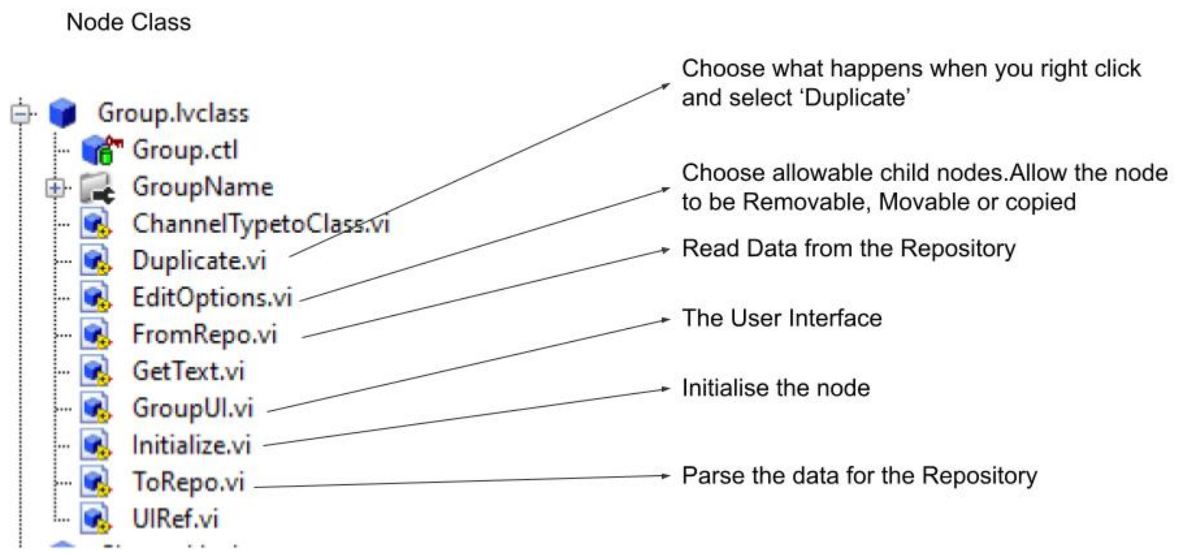


Figure 3 CEF Repository and Views relationship

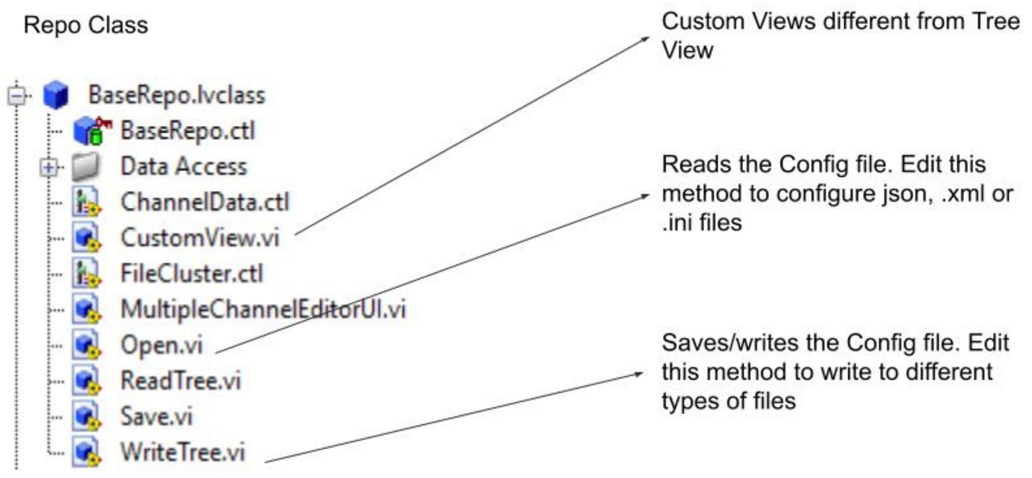
https://forums.ni.com/t5/Example-Code/Configuration-Editor-Framework-CEF/ta-p/3984276

Now let's look at the Architecture of the CEF. This image is taken from the NI CEF Documentation page. The link of which is given below the image.
The primary block we see here is the repository which is the in memory location where the

Configuration is stored. The Repo is a Class. This Configuration is then read and written by various entities.
First of which is our Tree View, then there is the File (this file maybe a binary, .ini, xml, json etc.
The last block labeled “Custom View” represents views other than the Tree View that can be created by the User.
The Configuration Editor Project Template comes with a custom view which is called the Multiple Channel Editor. This view enables the user to change multiple channels in one go. But here I must mention that from my experience a Custom View is an optional feature and sometimes you can’t really create one for your specific use case.


Node Class
Group.lvclass
Group.ctl
GroupName
ChannelTypetoClass.vi
Duplicate.vi
EditOptions.vi
FromRepo.vi
GetText.vi
GroupUI.vi
Initialize.vi
ToRepo.vi
UIRef.vi
Choose what happens when you right click and select ‘Duplicate’
Choose allowable child nodes.Allow the node to be Removable, Movable or copied
Read Data from the Repository
The User Interface
Initialise the node
Parse the data for the Repository

Each CEF project should have 1 Child of the Repo Class and several children of the Node Class. One of these Group’s is shown here.(Explain some methods of the Node and Repo Class) .
Now let's look at the most important method for our purposes, the From Repo and ToRepo functions.


Repo Class
BaseRepo.lvclass
BaseRepo.ctl
Data Access
ChannelData.ctl
CustomView.vi
FileCluster.ctl
MultipleChannelEditorUI.vi
Open.vi
ReadTree.vi
Save.vi
WriteTree.vi
Custom Views different from Tree View
Reads the Config file. Edit this method to configure json, .xml or .ini files
Saves/writes the Config file. Edit this method to write to different types of files

Now we look at the methods/functions of the Repo Class.

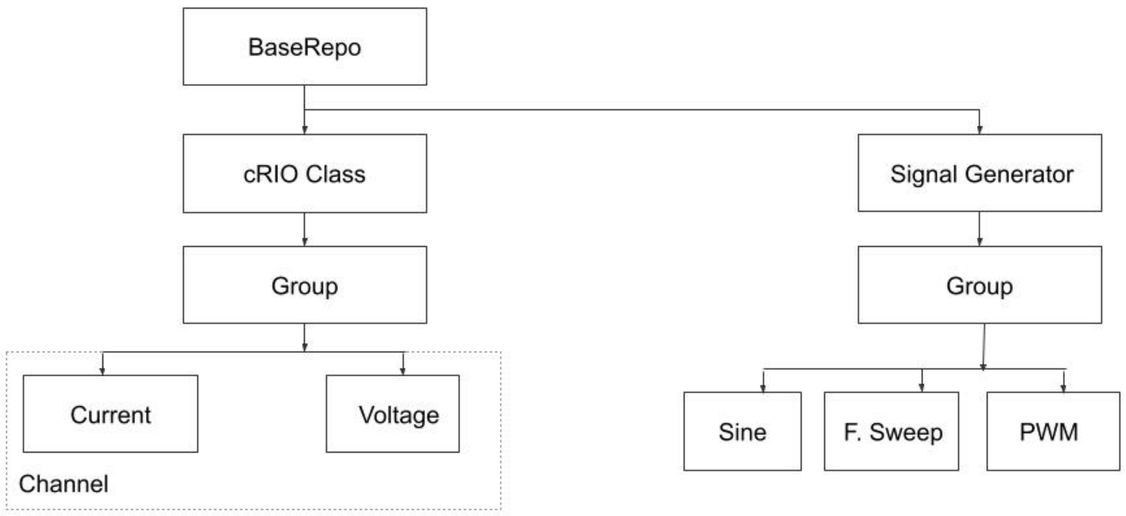
These are the main methods we need to edit to achieve multi device configurations. I would not touch the other methods like Write Tree or Read Tree
as their functionality does not need to be changed.

If you want to know more about some of those Node level functions please refer the CEF

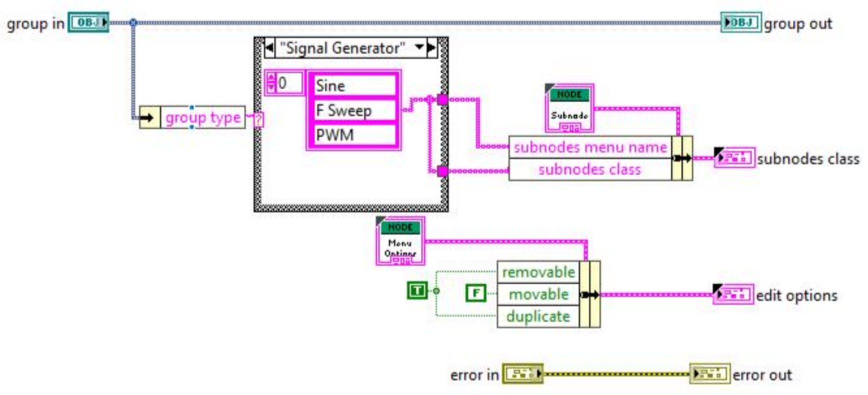
documentation or this very helpful video by Tom on youtube.
That's his Channel. Now I want to take this CEF a step further and bring in the multi channel and multi device element.

Also, if you can't understand a lot of what I say I would recommend you watch that video first and then re-watch this once it's published and it should make more sense.

From Repo function FlowChart

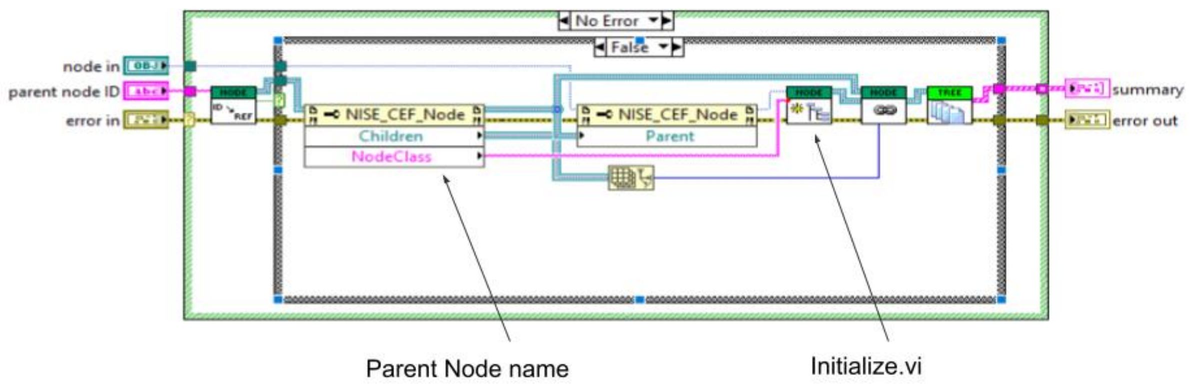


The first thing we need to do for this is to define the hierarchy of your nodes. As each node needs to know which node/nodes is/ are its "Children". In the cRIO example the hierarchy of nodes is simple as shown in the diagram. Now let's look at an example where we are adding another device.
Here I am adding a signal generator. Now, the generator's channels are divided into groups and each group can have 3 types of signals: a PWM, a sine and a frequency sweep.
Now some things to note here:
1. I want to reuse the Group Class for both a cRIO device and a Sig. Gen.
2. In the CRIO case the Current and Voltage Class Data is the same and that's the Parent "Channel" Class data. They don't have their own data whereas on the Signal Generation side I want to load in Classes with independent data. I.e Sine, PWM and Sweep don't have common data.
3. We have to create a bunch of new Classes. Here I like to use GDS to clone my existing Classes to create new ones. Also note I have renamed my cRIO Repo Class to BaseRepoClass.



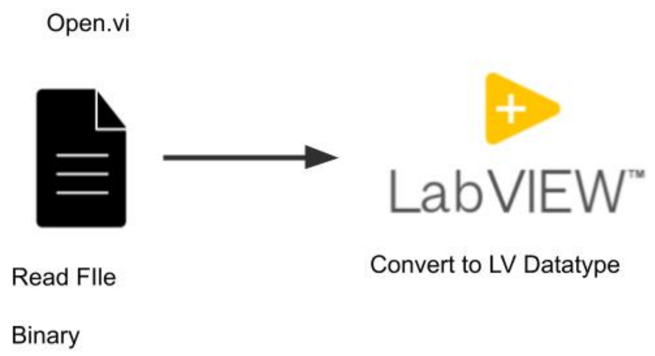
To reuse the Group Class I need to use a Case structure in its Edit options.vi which will select my shortcut menu options based on the type of parent Class Group has. cRIO or Signal Generator. So, I define a “Group type” element in the Group Class Data. Group type should be defined in the Initialise.vi

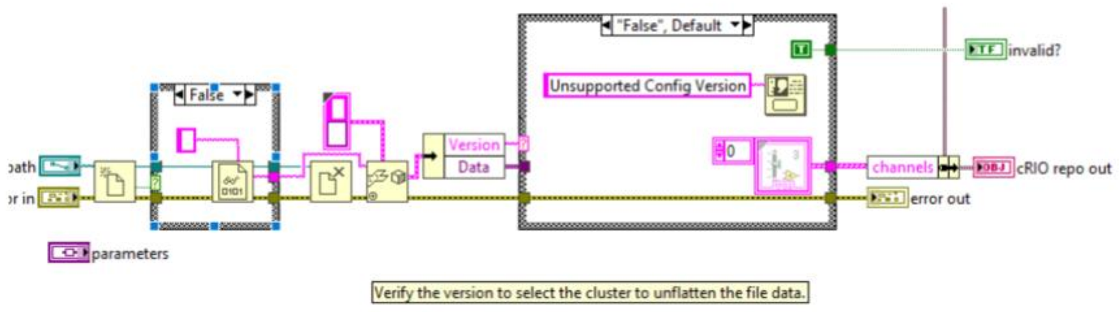
AddNode.vi

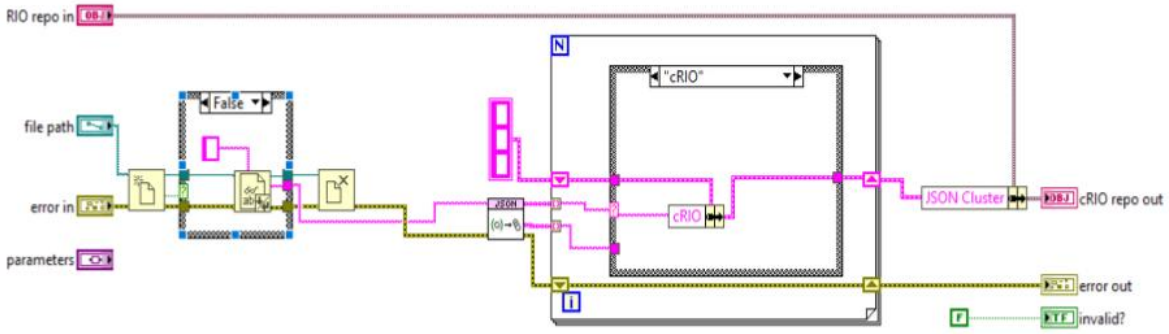


Now Initialize.vi is only called by AddNode.vi and here I wire the NodeClass Name to the

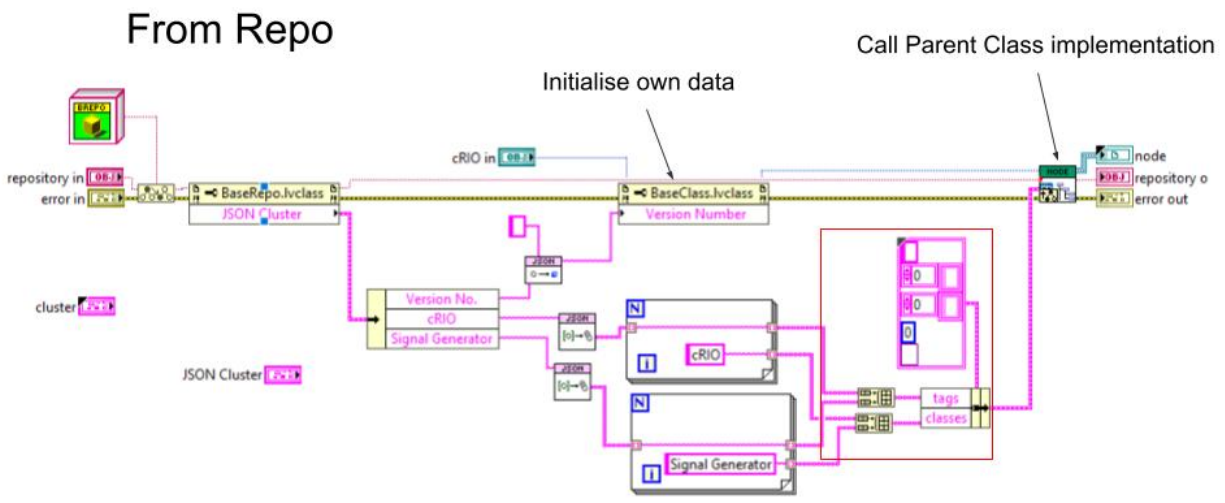
Parameter input of Initialize.vi So by doing this now Group can be reused for both the devices.




path
error in
parameters
False
Version
Data
"False", Default
Unsupported Config Version
0
invalid?
channels
cRIO repo out
error out
Verify the version to select the cluster to unflatten the file data.



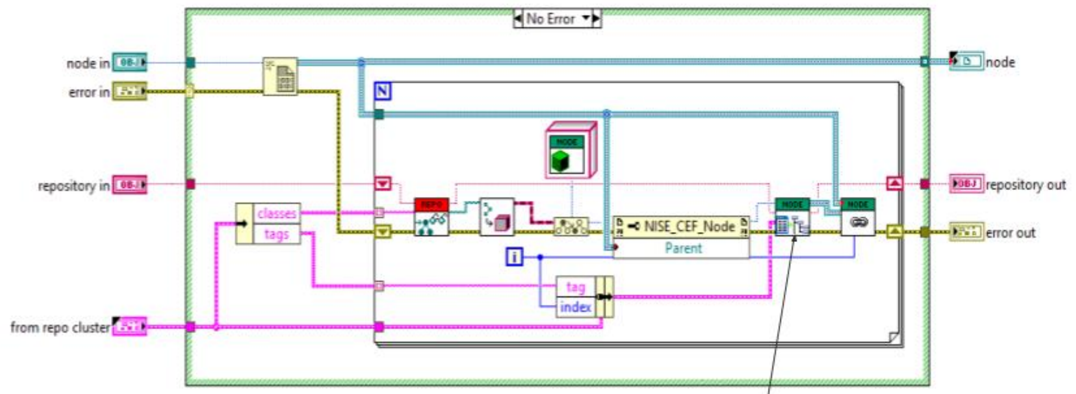
Now I wanna look at the functions which write to Repo and From Repo.
So in the Open.vi we Read from a Binary file and convert it to LV Data Type like so. In this case an array of Clusters. Called Channel Array. Now with multi device configurations I find it's very hard to do this because your multiple channels will not have the same data type but we can use Variants to get around that. (Or) if we are reading in a Json/XML file we can keep the configuration in its Json/XML format (which would just be a string datatype in LV and we pass those around. To our To Repo and From Repo functions. Something like this.



Next I wanna talk about the From repo and To Repo functions and how they are implemented. It took me quite some time to wrap my head around this but hopefully I can tell you how it works in the next 5 mins. We can look at From Repo first. From Repo is a Dynamic Dispatch function and has a Parent class implementation and Child Class implementation. We call the Base Class implementation first which looks like this.

Here I'm taking the JSON Cluster and then deciding which Classes I need to initialize after this Base Node is created. I pass the names to those Classes as an Array. I also pass in some tags which in this case are the individual JSON strings for the cRIO and Signal Generator objects. The class also initializes its own data using that Property Node there.
After this I call the Parent Class implementation of this method here.

From Repo



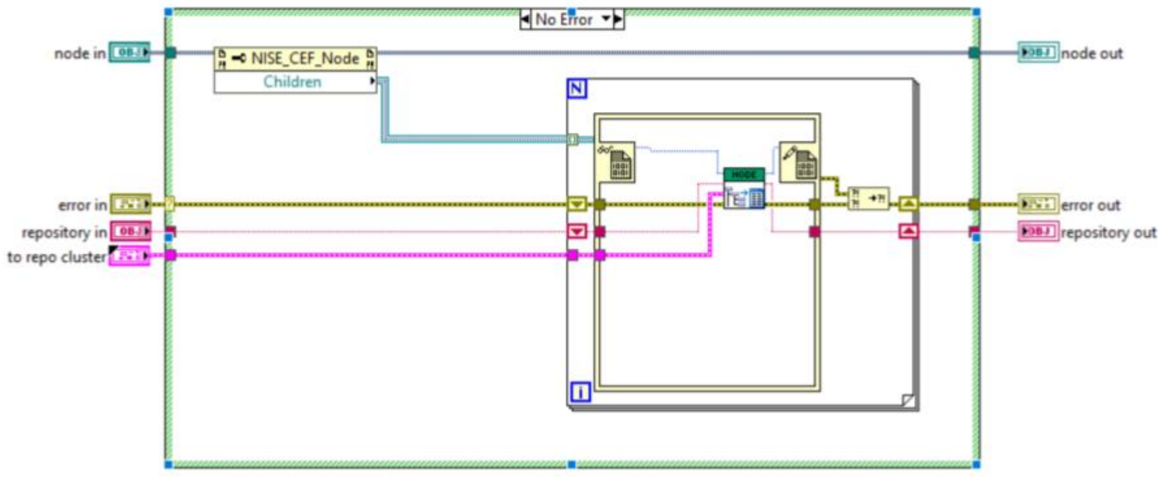
Calls From Repo method of newly created Class.

This is the Node (Base Class implementation of this function. Here we have a for loop that iterates through the Classes creating their DVR's and calling their From Repo method.
So this is kind of a recursive process and its important to understand this.
So therefore, Each from repo function initialises the data it needs for its Class and decides which Classes to create after and then passes this data on to its Parent Class (Node Class) which then creates the Class and its DVR and then calls that the new Classes From Repo function and this goes on until the last class.

The ToRepo function is implemented in the same way except that you don't need to put in logic to decide which Class to go to next.

I wanted to show my specific implementations of each of these.

To Repo



Now here I have the To Repo Parent Class function. You can see it's much simpler. In my case where I'm using Json I use my ToRepo methods to iteratively build up my configuration Json wherein each node Class adds its bit to the larger Configuration Json. It follows the same logic as our From Repo function.

So in summary each to repo function writes a part of the Json it has to the Repo through property nodes and finally the Save method of the Repo can consolidate these and write them to file.
Demo.

Q & A