IVI Instrument Driver Programming Guide (LabVIEW Edition)

June 2012 Revision 2.1

1- Overview

1-1 Recommendation Of IVI-C Driver

LabVIEW has a capability to import IVI-C instrument drivers. Although it is possible to use IVI-COM instrument drivers directly, it is much easier to program using IVI-C instrument drivers if they are supported. (Kikusui IVI instrument drivers contain both IVI-COM and IVI-C instrument drivers.)

Therefore this guidebook recommends using IVI-C instrument drivers.

Notes:

- This guidebook shows examples that use KikusuiPwx IVI instrument driver (KIKUSUI PWX series DC Power Supply). You can also use IVI drivers for other vendors and other models in the same manner.
- This guidebook describes how to create 32bit (x86) programs that run under Windows7 (x64), using LabVIEW 2011 (32bit edition).

1-2 IVI Instrument Class Interfaces

When using an IVI instrument driver, there are two approaches – using specific interfaces and using class interfaces. The former is to use interfaces that are specific to an instrument driver and you can utilize the most of features of the instrument. The later is to utilize instrument class interfaces that are defined in the IVI specifications allowing to utilize interchangeability features, but instrument specific features are restricted.

Notes:

- The instrument class to which the instrument driver belongs is documented in Readme.txt for each of drivers. The Readme document can be viewed from Start button➔All Programs➔Kikusui➔KikusuiPwx menu.
- If the instrument driver does not belong to any instrument classes, you can't utilize class interfaces. This means that you cannot develop applications that utilize interchangeability features.

1-3 Installing LabVIEW Instrument Driver Import Wizard

In order to writes program codes that calls IVI-C instrument driver in LabVIEW, you need generate a set of LabVIEW VI libraries (LabVIEW IVI-C wrapper) by using **LabVIEW Instrument Driver Import Wizard** that imports an IVI-C driver. The Import Wizard is not a part of LabVIEW, you need download from the following site and then install it.

https://lumen.ni.com/nicif/us/infolvinstdriver/content.xhtml

1-4 Installing IVI Compliance Package

In order to make the LabVIEW IVI-C wrapper generated with the Import Wizard function , **NI IVI Compliance Package** is required as a runtime environment. IVI Compliance

Package is not included in LabVIEW, so you need separately download from the following site then install it.

http://joule.ni.com/nidu/cds/view/p/id/2589

2- Importing IVI-C Driver .fp File

Here we describe how to import the IVI-C driver so that it can be used with LabVIEW environment . The driver import is required when using specific interfaces described in the next chapter. When using class interfaces described in the more next chapter does not require it.

2-1 Instrument Driver Import Wizard

Because an IVI-C instrument driver is provided as the same formats with legacy VXI Plug&Play instrument drivers (.fp, .h, .sub, and .dll), it cannot be used directly from LabVIEW environment. Therefore, it is required to import the driver interfaces converting to the LabVIEW-compatible formats (.vi or.llb).

To import the instrument driver, use the above-mentioned LabVIEW Instrument Driver Import Wizard.

After launching LabVIEW with the condition that the Instrument Driver Import Wizard is being installed, a new menu item - **Tools | Instrumentation | Import LabWindows/CVI Instrument Driver...** is already added so select it. Then, at the first screen you will see **Basic/Advanced** choice, so select **Advanced**.

Figure 2-1 Import Wizard - Welcome screen

Click **Next** button and you will be asked to enter a .fp (Function Panel) file, so select kipwx.fp that is placed in **C:/Program Files (x86)/IVI Foundation/IVI/Drivers/kipwx** directory.

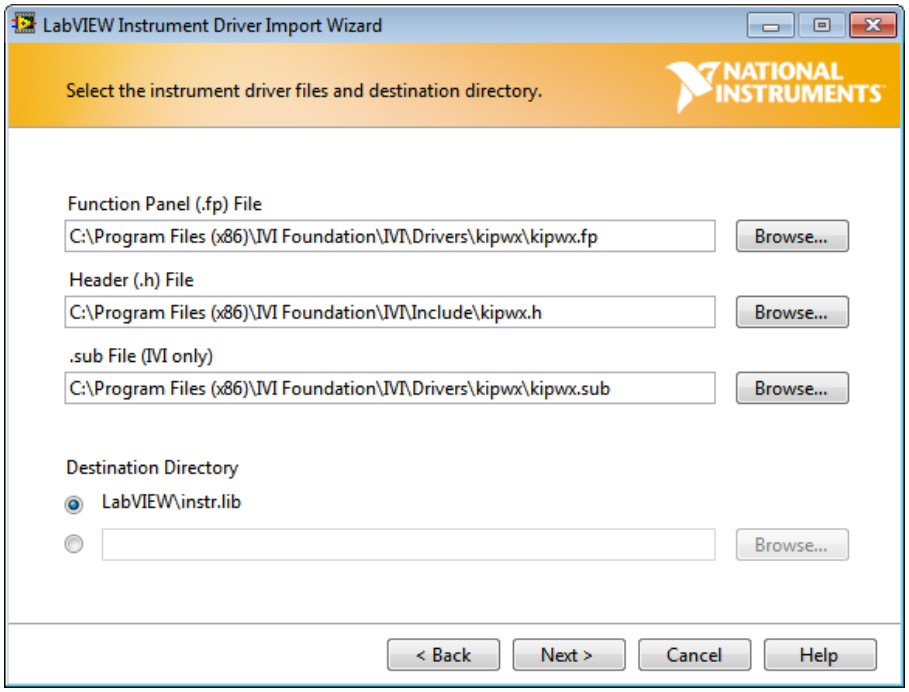

Figure 2-2 Import Wizard - Select Function Panel (.fp) File

The .h file (C language header file) and.sub file (attribute info file) will be automatically set. Click **Next** button, then you will be asked for the file type to be generated. Here, just confirm that every item is selected.

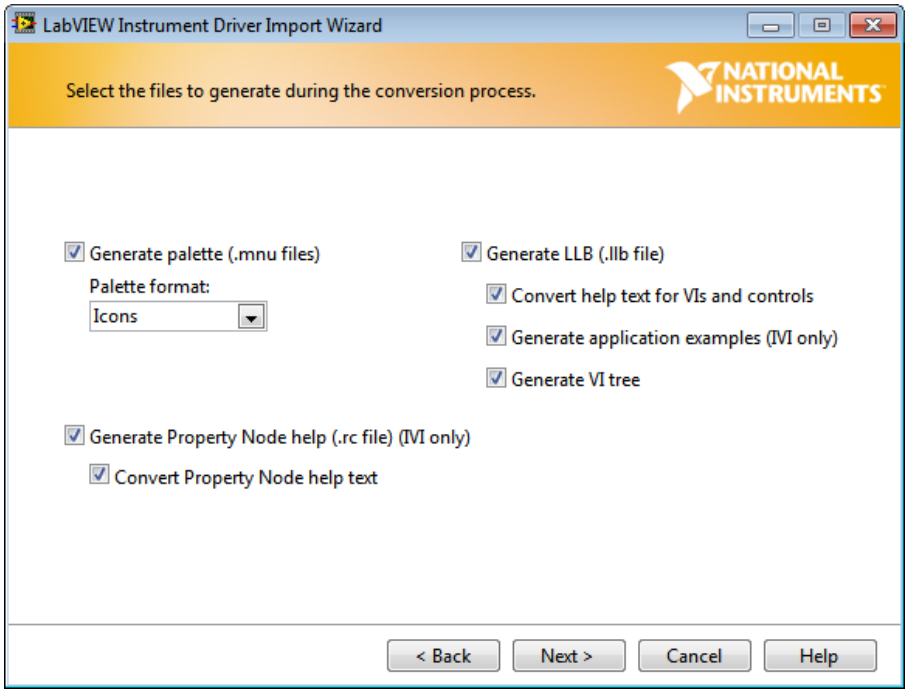

Figure 2-3 Import Wizard - Select Files To Generate

Click **Next** button more, the you will see a screen for settings review, then confirm each item - Driver Group, Driver Prefix, Shared Library DLL.

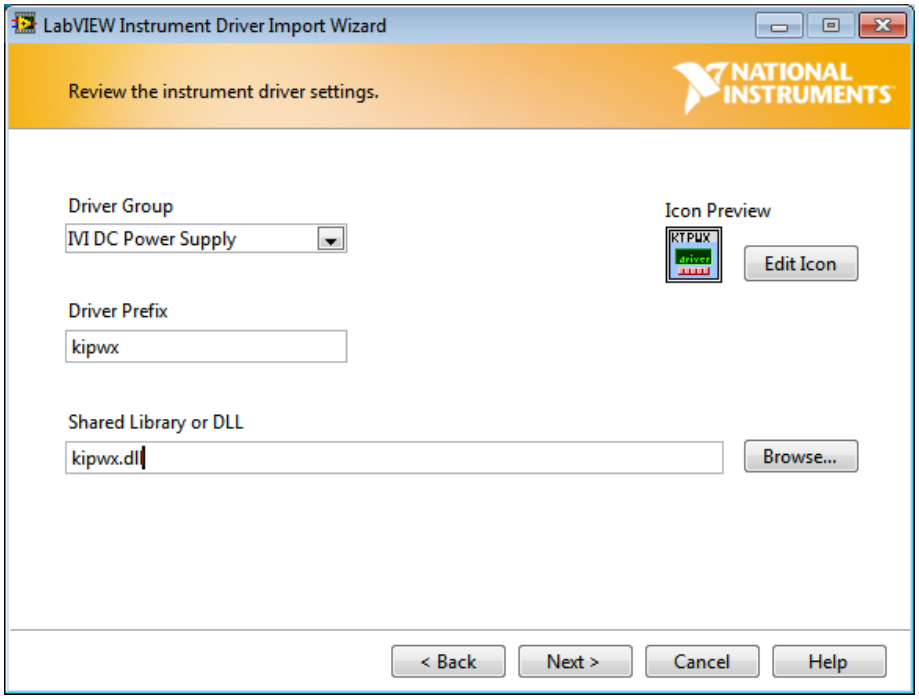

Figure 2-4 Import Wizard - Review Instrument Driver Settings

Here, there are a couple of important items (techniques for avoiding potential issues Import Wizard has) to take care.

Driver Group

Specifies an instrument class that the IVI instrument driver belongs to. KikusuiPwx (kipwx) IVI instrument driver is **IviDCPwr** class -compliant (IVI DC Power Supply), so choose the one.

Notes:

- When importing an IVI driver created with Nimbus (IVI-COM/IVI-C instrument driver development tool), the default choice of this item may be VXI Plug&Play. In this case, you need select the correct instrument class by hands by according to the instrument driver.

Driver Prefix

Confirm that it has the same base file name as the .fp that was specified at first.

Shared Library or DLL

Specifies the driver DLL file name. As default, a wildcard expression such as kipwx_32.* (or kipwx_64.* for 64bit driver). But here, click **Browse...** button to explicitly specify the correct DLL file name. The DLL is placed in **C:/Program Files (x86)/IVI Foundation/IVI/BIN** directory.

Notes:

- As for IVI instrument drivers created with Nimbus, the 64bit driver name is like <prefix>_64.dll, but the 32bit driver name is like <prefix>.dll having no _32 suffix. Therefore, leaving the default wildcard such as <prefix>_32.* will not match with the actual DLL file, resulting to generate the wrapper that will not work normally.

Click **Next** button and you will see a screen where you select VIs to be generated. If you see the following warning message, the above mentioned DLL file name may be incorrect. In this case close the warning message with the upper-right [X] button, then retry to specify the DLL file or confirm the installation condition of the IVI driver.

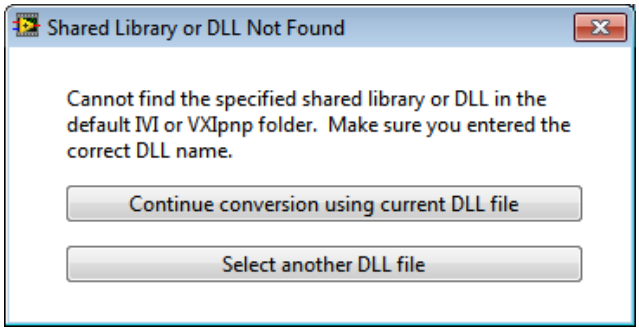

Figure 2-5 Import Wizard - Shared Library or DLL Not Found

If DLL file is correctly specified, functions will be extracted, and you will see the function tree screen shown below.

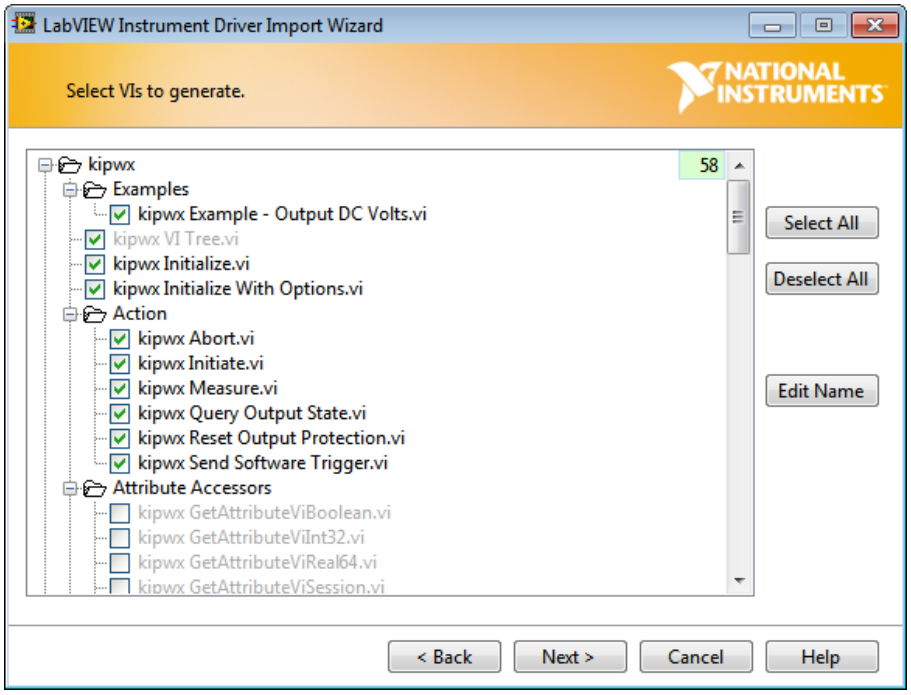

Figure 2-6 Import Wizard - Select VIs To Generate

All the functions are detected as default, click **Next** to proceed.

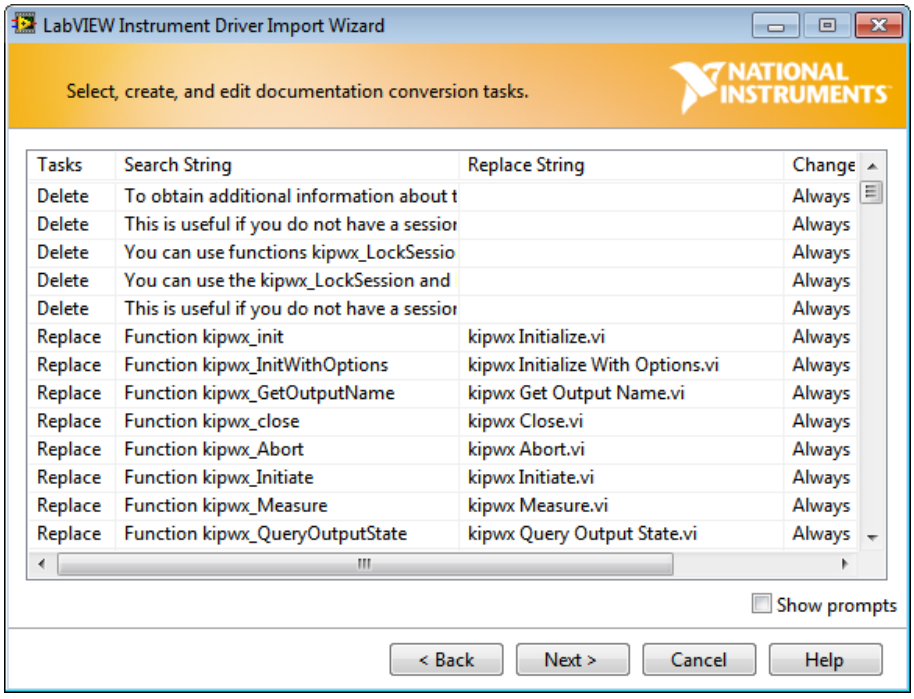

Figure 2-7 Import Wizard - Select, Create, & Edit Document Conversion Tasks

Here you specify the conversion condition for documents (mainly context help). Most of conversion are what replace "VI" with "function". It is not an important stuff and troublesome if being prompted for asking convert or not, so uncheck **Show Prompts** here.

By clicking **Next** more, the summary of conversion will be shown so proceed to convert (import).

After the conversion is complete, the kipwx sub-directory will be created under the LabVIEW's default instrument driver directory (normally **C:/Program Files (x86)/National Instruments/LabVIEW 2011/instr.lib)**. There, the VI library file (kipwx.llb), and multiple palette menu files(.mnu) are generated. The set of these files is the IVI-C wrapper that can be used directly from LabVIEW.

The generated kipwx IVI-C wrapper can be referenced through the Instrument I/O functions palette on the LabVIEW block diagrams.

Note:

- The generated .llb file is a wrapper module to the IVI-C driver, and not a real instance of the instrument driver. Therefore, the IVI instrument driver must be installed on the target machine when you run the completed application.
- When installing the IVI instrument driver to the target machine, use the driver's original installer as well as installing to the development machine. Just copying DLLs alone will not work correctly.
- To the target machine, IVI Compliance Package must be also installed as well as LabVIEW Runtime Engine.

3- 2- Example Using Specific Interfaces

Here we introduce an example using specific interfaces. By using specific interfaces, you can utilize the maximum feature (or model specific functions) provided by the driver but you have to spoil interchangeability.

3-1 2-2 Adding Controls and Functions

First, create a new application. Open the Front Panel window, place an **error in** cluster and an **error out** cluster.

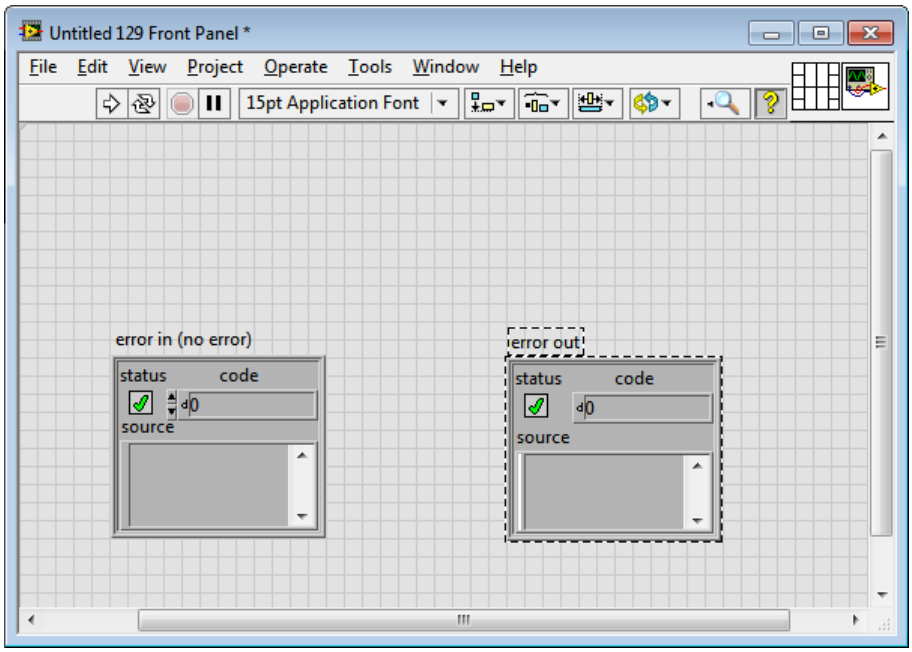

Figure 3-1 Front Panel

Next, open the Block Diagram window, then open the function palette for the ki4800 driver (wrapper). The function palette will be found through the context menu → **Instrument I/O** → **Instr Drivers** → **kipwx**.

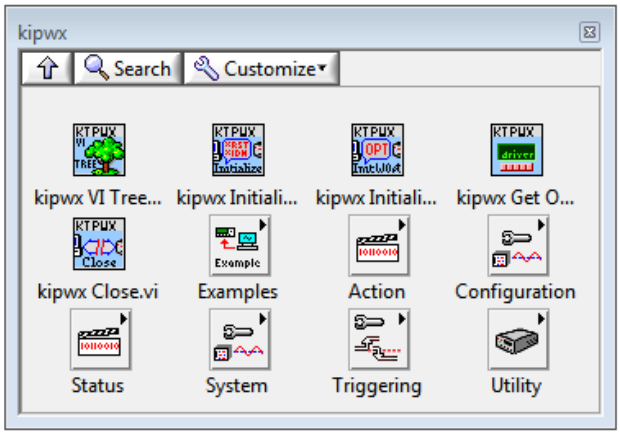

Figure 3-2 kipwx Function Palette

On the block diagram, place **`Initialize With Options.vi`** and **`Close.vi`**. Furthermore place **`Configure Voltage Level.vi`**, **`Configure Current Limit.vi`**, and **`Configure Output Enabled.vi`**, which will be found in **Configuration→Output** palette,

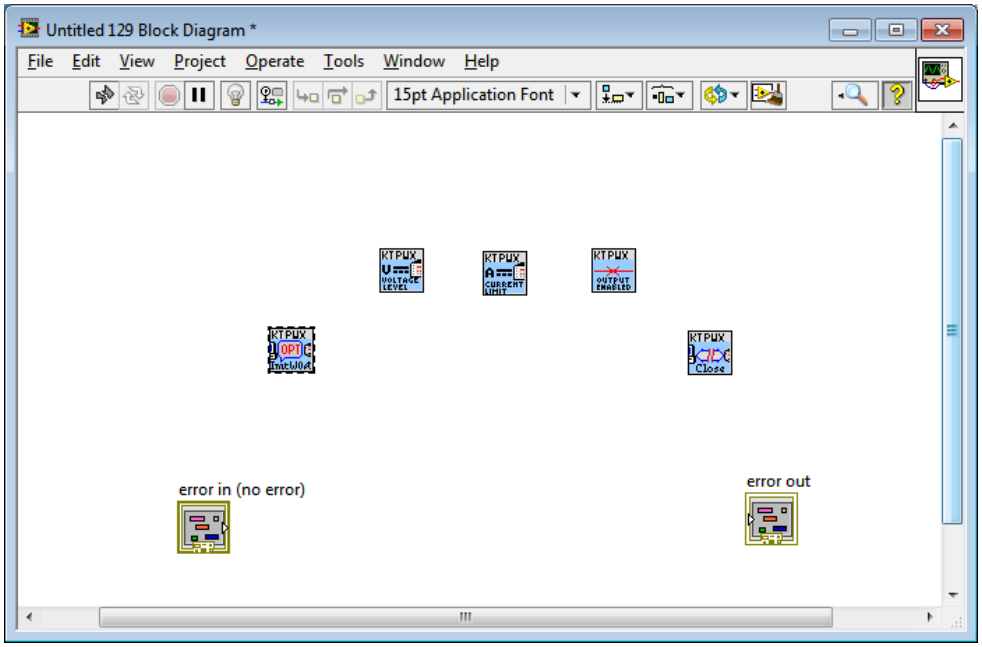

Figure 3-3 Block Diagram

3-2 2-3 Parameter Settings

Here, assuming that Kikusui PWX Series DC Power Supply on the network is configured as IP address 192.168.1.5, pass the parameters -- **`resource name`**, **`id query`**, and **`reset device`** to **`Initialize With Options.vi`**.

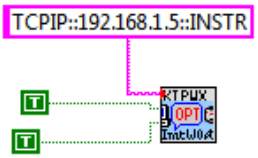

Figure 3-4 Params for Initialize With Options

Subsequently, add parameters that set voltage, current, and output. Here, we set 20V/2A and the output ON.

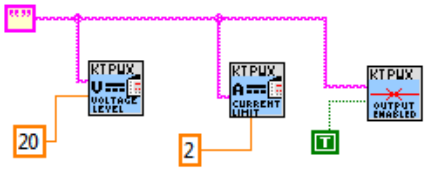

Figure 3-5 Params for Configure Functions

Here, mind the blank string commonly passed to the three VIs. This is the target channel name for the DC power supply to be controlled. Details are described later.

Finally, wire the controls/functions between **error in** and **error out** clusters as like the picture below. Not only connecting error ins/outs, but make sure to connect instrument session (handle) wires also.

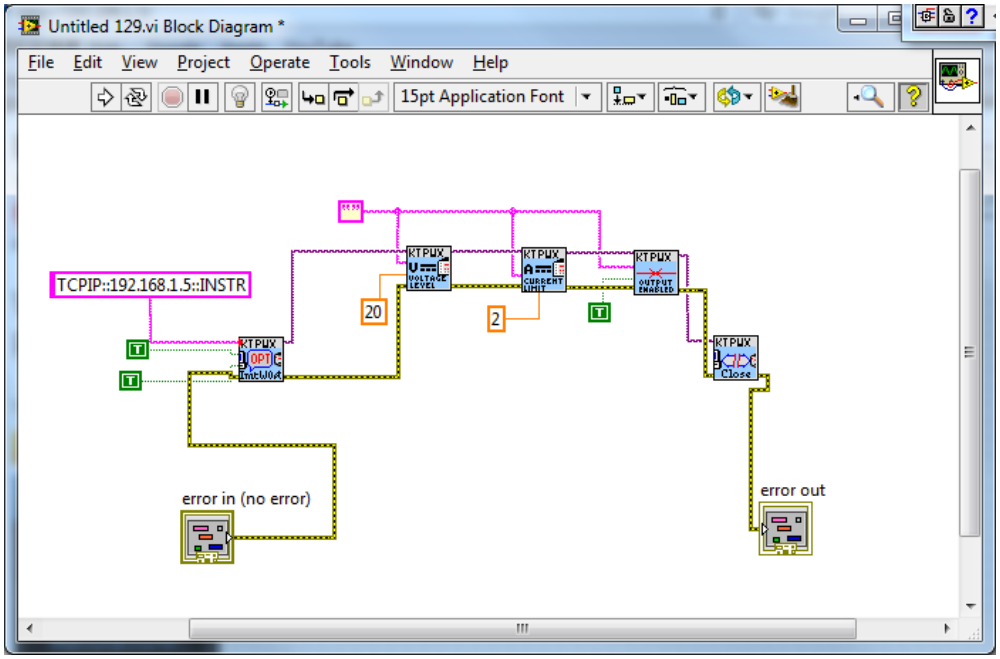

Figure 3-6 Open/Configure/Close

3-3 Program Execution

You can execute the previous codes for the time being. **`Initialize With Options.vi`** resets the instrument settings since the **`Reset Device`** parameter is set to `TRUE`. As you execute the program, I/O communications with the instrument immediately starts. If the instrument is actually connected with succeesing **`Initialize With Options`** and **`Close`** calls, the error code shown on the **error out** cluster will be 0. If a communication problem has occurred or the VISA library is not configured properly, an exception is generated and its information will be shown on the **error out** cluster. By selecting Context Menu ➔**Explain Error**, you can see the detail information on the **explain error** dialogue.

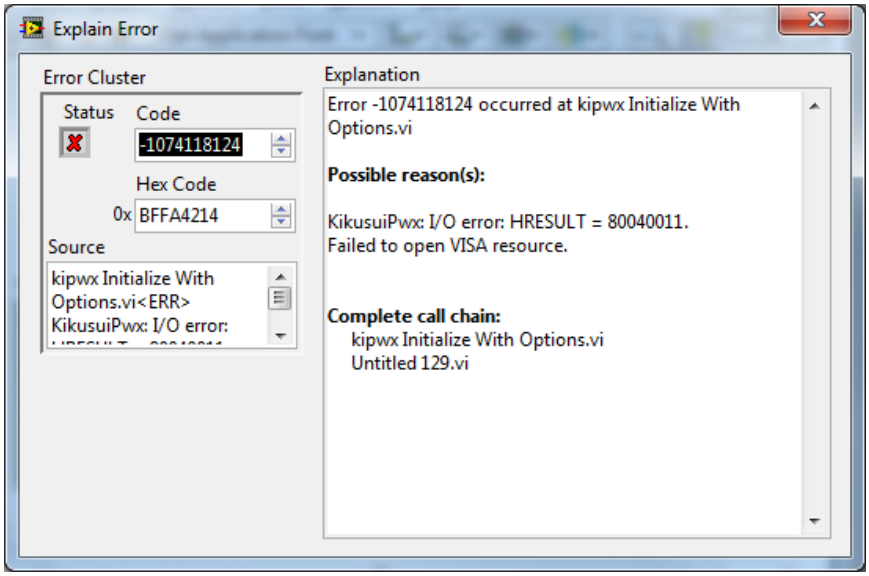

Figure 3-7 Runtime Error

4- Description

4-1 Opening Session

To open the driver session, the **kipwx Initialize With Options.vi** is used. Although the prefix **kipwx**, which is applied to the vi (function) is different depending for each instrument driver, this naming convention is applied for all the IVI-C instrument drivers

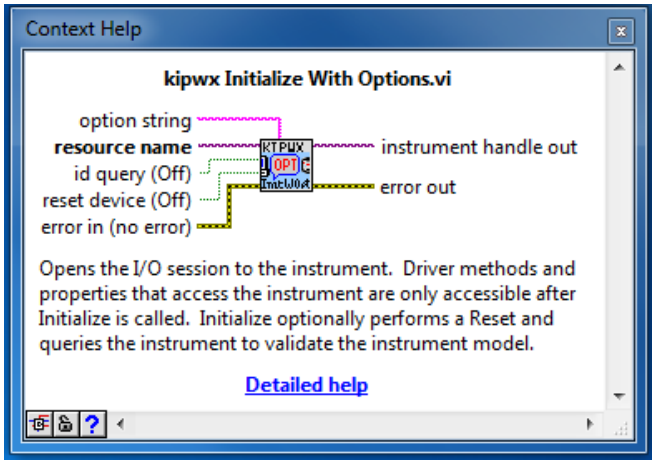

Figure 4-1 Initialize With Options.vi Help

Notes:

- As a terminology of IVI-C and VXI Plug&Play Instrument Driver, the term <prefix> is frequently used. This is an identifier name that is given for each instrument driver, and kipwx is the one for this guidebook. For example, a generic expression **<prefix> Initialize.vi** specifies **kipwx Initialize.vi** for the kipwx instrument driver.
- Every vi (driver function) other than **<prefix> Initialize.vi** and **<prefix> Initialize With Options.vi** has **instrument handle (in)** parameter at the upper-left corner of the VI.
- Every vi (driver function) other than **<prefix> Close.vi** has the output parameter **instrument handle out** at the upper-right corner of the VI. This shall be connected to the **instrument handle (in)** of the next vi.
- **<prefix> Initialize.vi** is remained for the compatibility with VXI Plug&Play drivers. This is equivalent to **<prefix> Initialize With Options.vi** with exception that **option string** cannot be specified.

Now let 's talk about parameters of the **kipwx Initialize With Options.vi**. Every IVI instrument driver has an **Initialize With Options.vi**, which is defined by the IVI specifications. This function has the following parameters.

Table 4-1 Initialize With Options のパラメータ

Parameter	Type	Description
Resource Name	String	VISA resource name string. This is decided according to the I/O interface and/or address through which the instrument is connected. For example, a LAN-based instrument having IP address 192.168.1.5 will be TCPIP::192.168.1.5::INSTR (when VXI-11case).
Id Query	Boolean	Specifying VI_TRUE performs ID query to the instrument.
Reset Device	Boolean	Specifying VI_TRUE resets the instrument settings.
Option String	String	Overrides the following settings instead of default: RangeCheck Cache Simulate QueryInstrStatus RecordCoercions Interchange Check Furthermore you can specify driver-specific options if the driver supports DriverSetup features.

Resource Name specifies a VISA address (resource name). If **Id Query** is VI_TRUE, the driver queries the instrument identities using a query command such as **"*IDN?"**. If **Reset Device** is VI_TRUE, the driver resets the instrument settings using a reset command such as **"*RST"**.

Option String has two features. One is what configures IVI-defined behaviours such as **RangeCheck**, **Cache**, **Simulate**, **QueryInstrStatus**, **RecordCoercions**, and **Interchange Check**. Another one is what specifies **DriverSetup** that may be differently defined by each of instrument drivers. Because the **Option String** is a string parameter, these settings must be written as like the following example:

```
QueryInstrStatus = TRUE , Cache = TRUE , DriverSetup=12345
```

Names and setting values for the features being set are case-insensitive. Since the setting values are ViBoolean type, you can use any of VI_TRUE, VI_FALSE, 1, and 0. Use commas for splitting multiple items. If an item is not explicitly specified in the **Option String** parameter, the IVI-defined default value is applied for the item. The IVI-defined default values are VI_TRUE for **RangeCheck** and **Cache**, and VI_FALSE for others.

Some instrument drivers may have special meanings for the **DriverSetup** parameter. It can specify items that are not defined by the IVI specifications when invoking the **InitializeWithOptions** function, and its purpose and syntax are driver-specific. Therefore, specifying the **DriverSetup** must be at the last part on the **Option String** parameter. Because the contents of **DriverSetup** are different depending on each driver, refer to driver's Readme document or online help.

4-2 Channel Access

When supporting power supply and/or oscilloscope instruments, the IVI instrument driver is generally designed assuming the instrument has multiple channels. Therefore, driver functions operating instrument panel settings often have the **channel name** parameter, which specifies the channel.

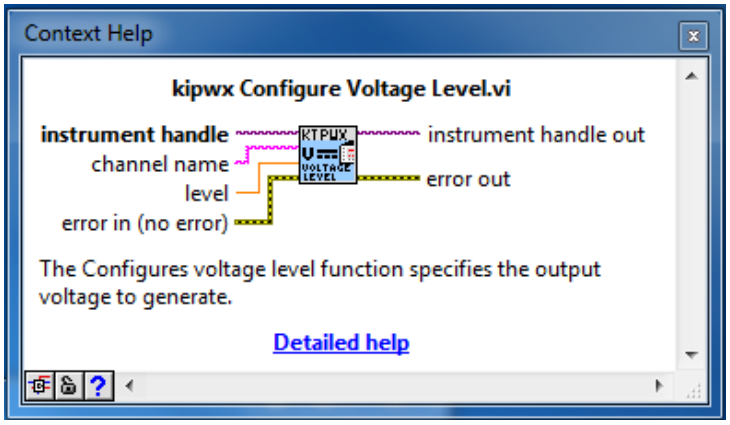

Figure 4-2 Configure Voltage Level.vi Help

As this guidebook uses the KikusuiPwx (kipwx) driver that operates the DC power supply, we use a channel name that specifies the channel name to controlled. Above example uses a blank string, however black is applicable to the case when there is only one channel. You need explicitly specify a channel name when multiple channel model, and the name will be normally such as **"Output1"**. (PWX assigns channel numbers from zero under Multi-Drop expanded operations, so the first channel name is **"Output0"**.) Detail about channel names that can be actually used, refer to the driver online help.

4-3 Closing Session

To close the instrument driver session, use the **kipwx Close.vi** function.

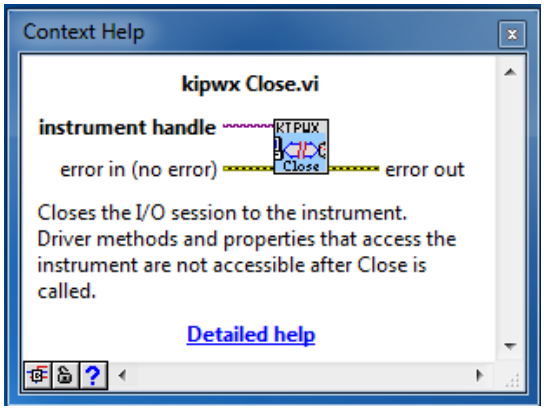

Figure 4-3 Close.vi Help

5- Example Using Class Drivers

Now we explain how to use instrument class drivers. By using instrument class drivers, you can swap the instruments without recompiling/relinking your application codes. In this case, however, IVI-C instrument drivers for both pre-swap and post-swap models must be provided, and these drivers both must belong to the same instrument class. There is no interchangeability available between different instrument classes.

5-1 Virtual Instrument

What you have to do before creating an application that utilizes interchangeability features is create a virtual instrument. To realise interchangeability features, you should not write codes that are very specific to a particular IVI-C instrument driver (e.g. direct call of **kipwx_init** function) and should not write a specific VISA address (resource name) such as " TCPIP::192.168.1.5::INSTR ". Writing them directly in the application spoils interchangeability.

Instead, the IVI specifications define methods to realise interchangeability by placing the external IVI Configuration Store. The application indirectly selects an instrument driver according to contents of the IVI Configuration Store, and accesses the indirectly loaded driver through the class driver that has no dependency to specific instrument models.

The IVI Configuration Store is normally **C:/ProgramData/IVI Foundation/IVI/IviConfigurationStore.XML** file and is accessed through the IVI Configuration Server DLL. This DLL is mainly used by IVI instrument drivers and some VISA/IVI configuration tools, not by end-user applications. When using LabWindows/CVI, the NI-MAX (NI Measurement and Automation Explorer) software provided by National Instruments allows you to perform IVI driver configurations.

Creating Driver Session

After launching NI-MAX, refer to the **IVI Drivers** node on the tree. Right-click on the **Driver Session** then select **Create New (case sensitive)...** menu to create a new Driver Session. Being asked for its name, give the name **`mySupply`**. Selecting **General** tab you will see the following screen.

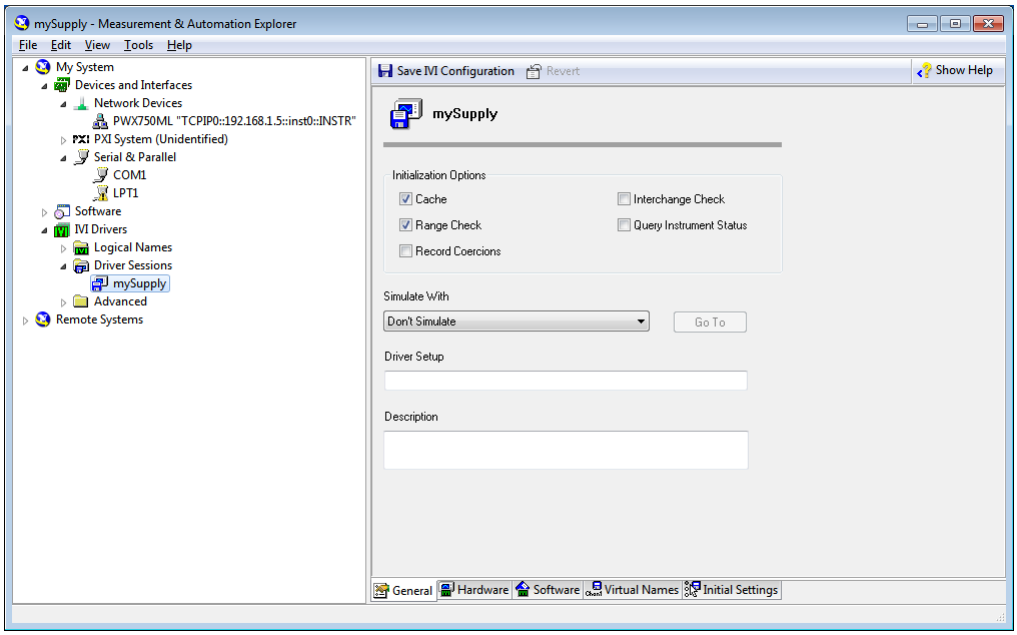

Figure 5-1 NI-MAX General Tab

Creating Hardware Asset

Subsequently select the **Hardware** tab to show the hardware asset management screen. The hardware asset specifies what interface route your actual instrument is connected through. Here you click the **Add** button to create a new Hardware Asset. Being asked for its name, give the name `mySupply` again, furthermore specify a valid VISA address (TCPIP::192.168.1.5::inst0::INSTR in this case) though which the actual instrument is connected, as **Resource Descriptor**.

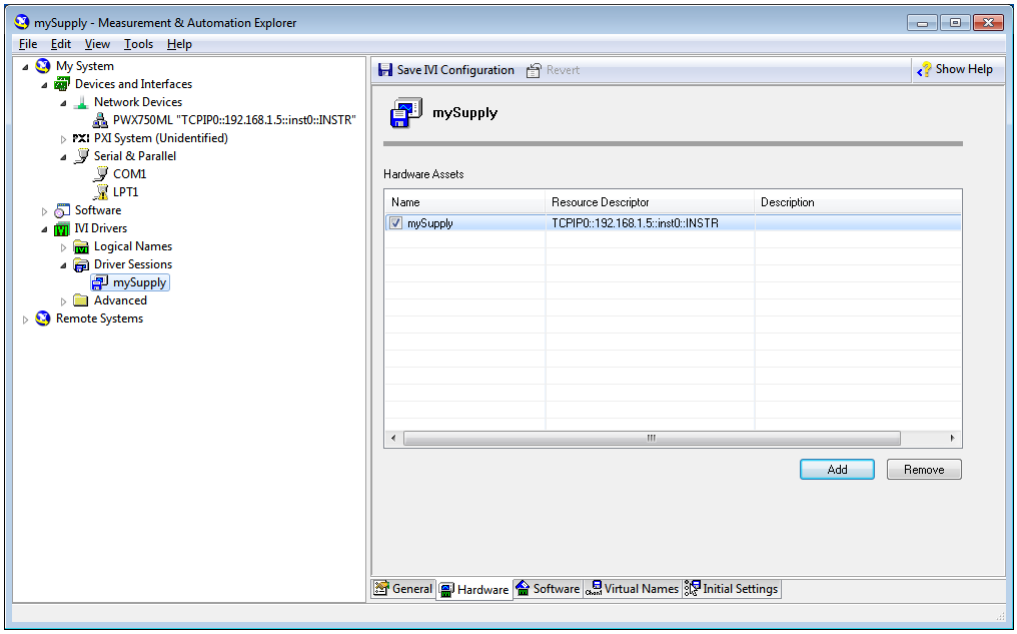

Figure 5-2 NI-MAX Hardware Tab

Setting Linkage for Software Module

Subsequently select the **Software** tab to show the software module management screen. The software module specifies the instrument driver module (DLL module). Here select **kipwx** from the **Software Module** list.

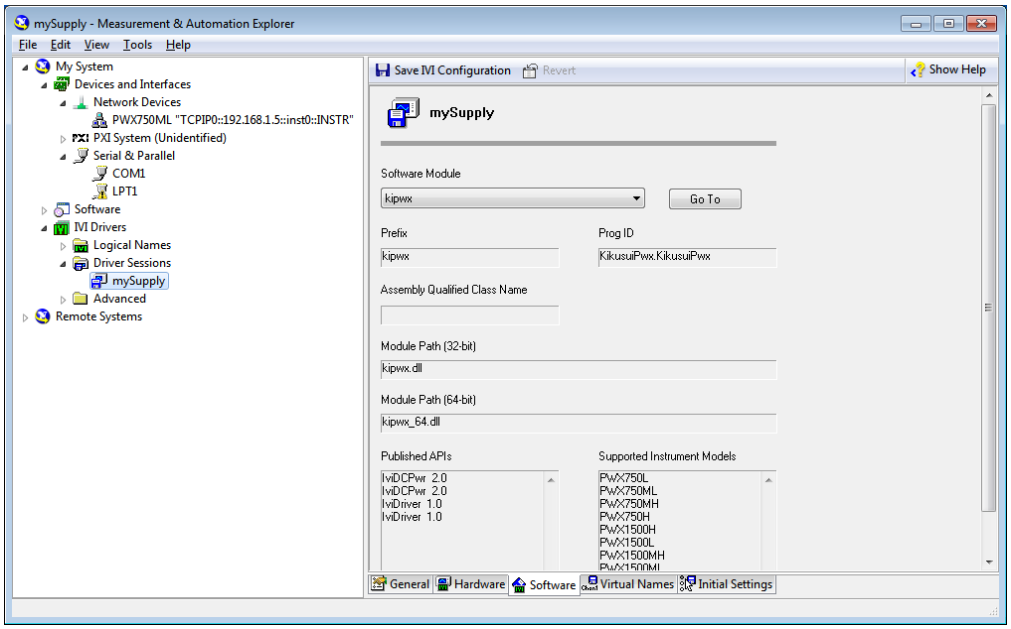

Figure 5-3 NI-MAX Software Tab

Creating Virtual Name

Subsequently select the **Virtual Names** tab to show the virtual name management screen. Normally, when channel names are related such as for power supply drivers, valid channel names are different depending on the drivers. Therefore, these channel names also have to be virtualized. Click the **Add** button to add a virtual name, then type "`Track_A`" for **Virtual Name**. Furthermore from **Physical Name** list, channels names that are working for the actual instrument are enumerated, On the list, **`IviDcpwrChannel!!Output0`** is only shown so select it, or simply type **`Output0`**.

Notes:

- Depending on driver's implementation or configuration of multi-channel power supplies, not all the channel names may be shown. As for valid channel names for each driver, refer to driver's Readme.txt or online help.

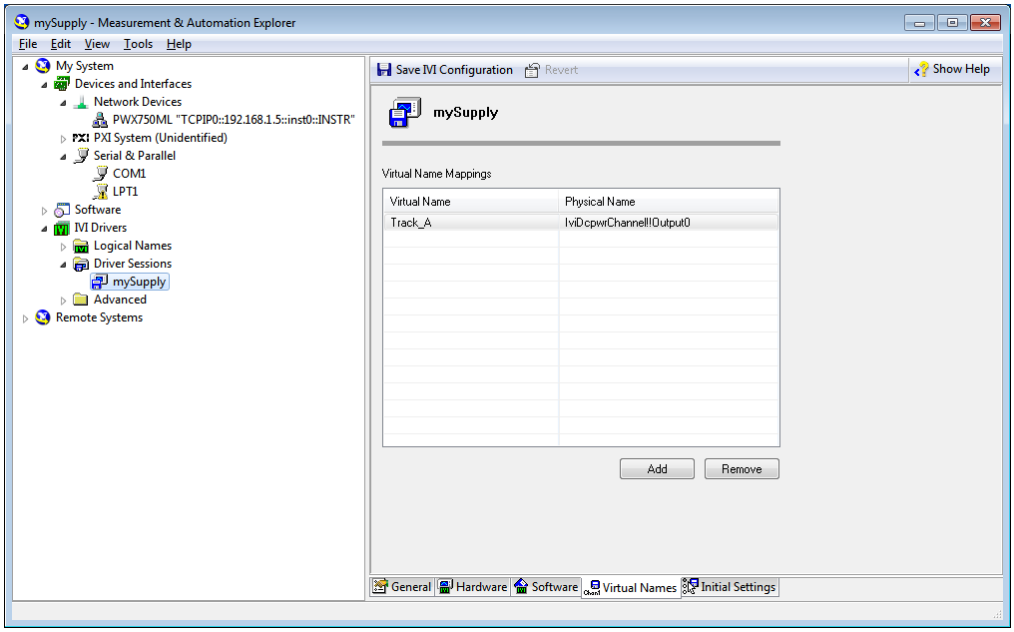

Figure 5-4 NI-MAX Virtual Names Tab

Creating Logical Name and linkage

Finally create a logical name. The logical name is equivalent to the name of virtual instrument configured with the NI-MAX. Refer to the **IVI Drivers** node on the tree. Right-click the **Logical Name** then select the **Create New (case-sensitive)** menu to create the new logical name. Being asked for its name, give the name "`MySupply`". Furthermore, select `mySupply` from the **Driver Session** list.

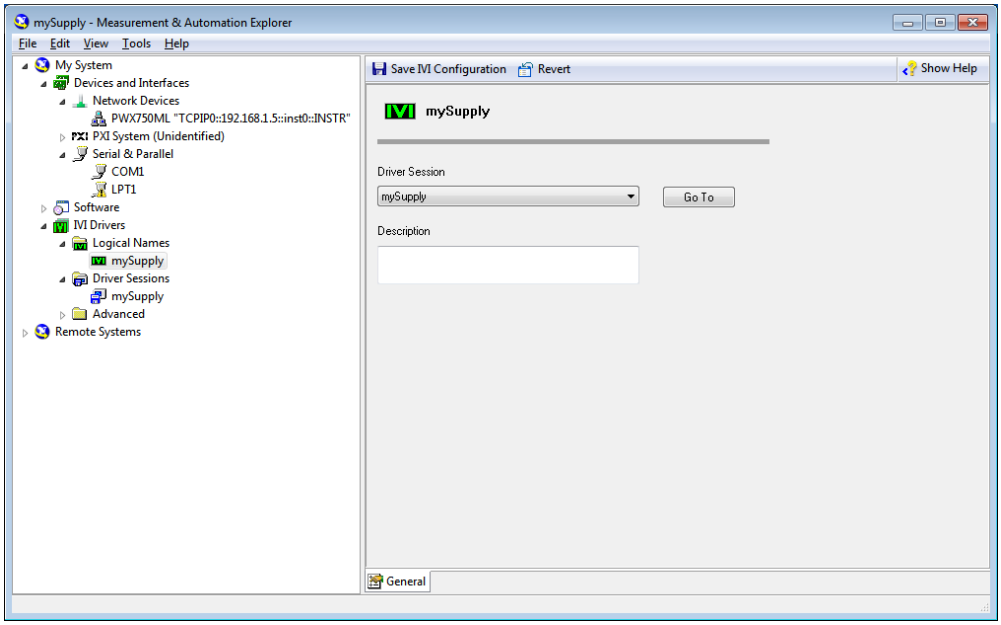

Figure 5-5 NI-MAX General Tab

Configuration for the virtual instrument is complete. Click the **Save IVI Configuration** button placed at the upper screen on the NI-MAX to save changes.

5-2 Adding Controls and Functions

First, create a new application. Open the Front Panel window, place an **error in** cluster and an **error out** cluster.

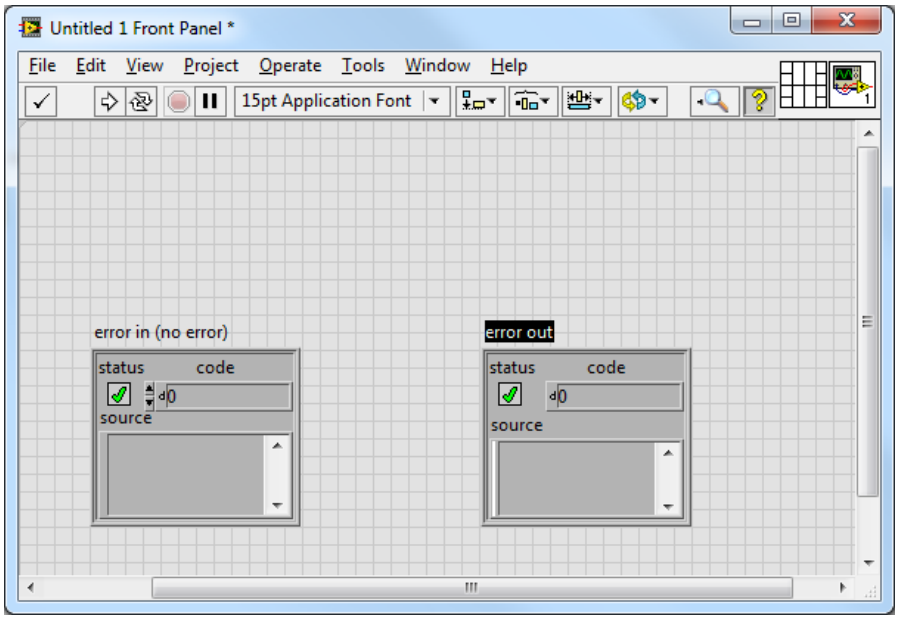

Figure 5-6 Front Panel

Next, open the Block Diagram window, then open the function palette for the IviDCPwr class driver (wrapper). The function palette will be found through the context menu → **Instrument I/O** → **IVI Class Drivers**→ **DC Power Supply**.

Notes:

- When IVI Class Drivers palette is not found, appropriate version of IVI Compliance Package may not be installed.

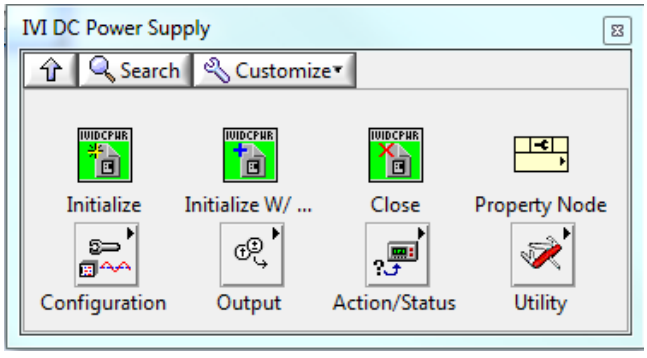

Figure 5-7 IviDCPwr Function Palette

Furthermore place **`Initialize With Options.vi`**, **`Close.vi`** on the block diagram. Furthermore add **`Configure Voltage Level.vi`**, **`Configure Current Limit.vi`**, **`Configure Output Enabled.vi`**, from **Configuration** palette.

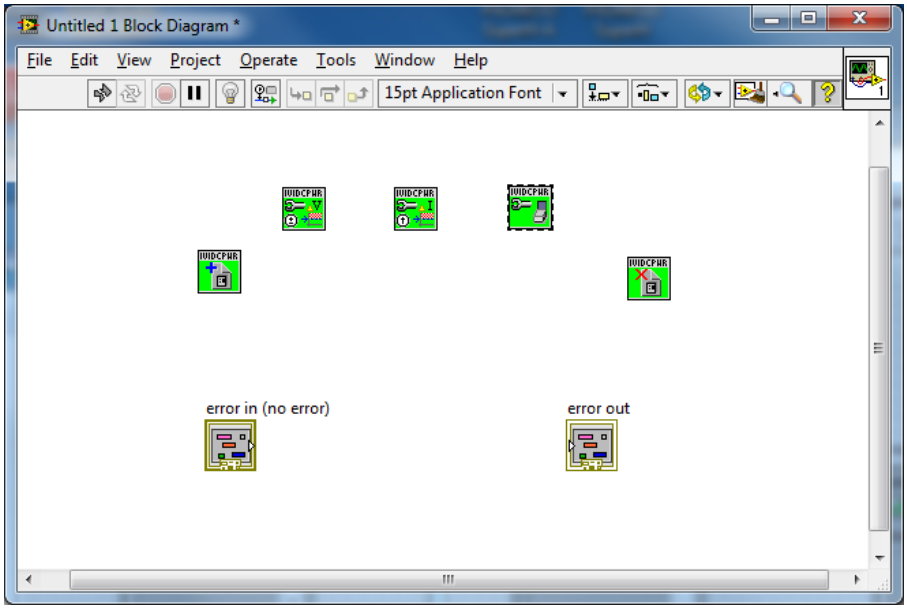

Figure 5-8 Block Diagram

At the class driver's initialization function, specify a virtual instrument (IVI Logical Name) instead of VISA resource name (VISA address). Here specify **`mySupply`** that was configured with NI-MAX.

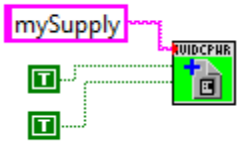

Figure 5-9 Params for Initialize With Options

Subsequently, add parameters that set voltage, current, and output. Here, we set 20V/2A and the output ON.

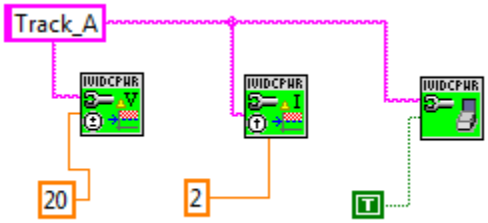

Figure 5-10 Params for Configure Functions

Mind the string -- **`"Track_A"`**. This is the target channel name for the DC power supply to be controlled. Details are described later.

Finally, wire the controls/functions between **error in** and **error out** clusters as like the picture below. Not only connecting error ins/outs, but make sure to connect instrument session (handle) wires also.

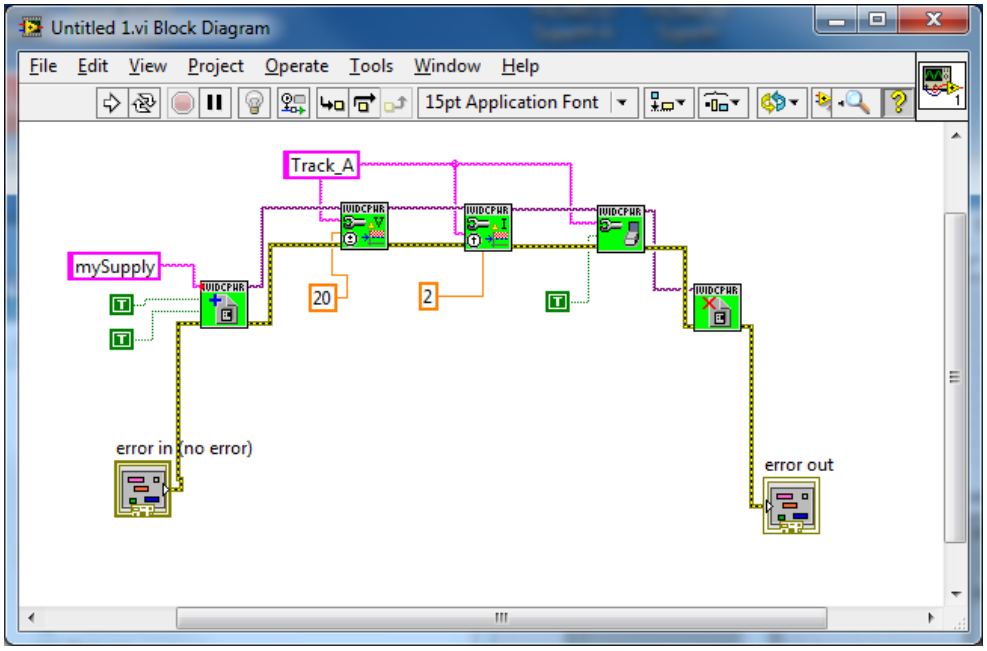

Figure 5-11 Open/Configure/Close

6- Description

6-1 Opening Class Driver Session

To open the driver session, the **`IviDCPwr Initialize With Options.vi`** is used. The prefix **`IviDCPwr`**, which is applied to the vi (function), is specific to the IviDCPwr class driver. In this program that utilizes the class driver, there never be dependency to instrument drivers of specific models such as kipwx (our PWX series DC Power Supply) or AgN57xx (Agilent N5700 series DC Power Supply).

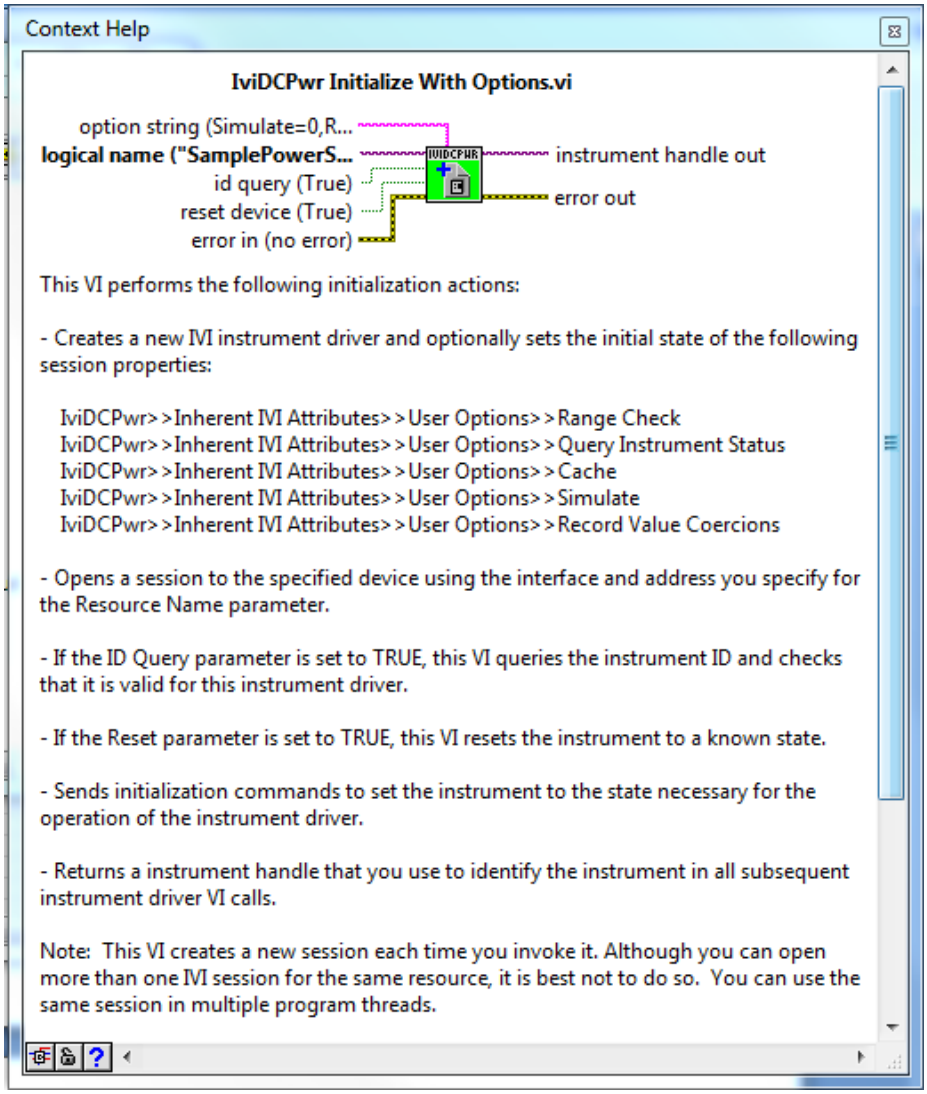

Figure 6-1 Initialize With Options.vi Help

Class drivers are different with normal instrument drivers, thus you cannot pass VISA address to the **`Initialize With Options.vi`** directly. Instead, pass the logical name **`"mySupply"`** configured in the NI-MAX. The class driver, by referencing to the logical name, searching for the appropriate instrument driver DLL (Software Module) and VISA address (Hardware Asset), then at last invokes the **`kipwx Initialize With Options.vi`** indirectly.

Although the contents for **Option String** (Cache、Range Check, Record Coercions, Interchange Check, Query Instrument Status, and Driver Setup string) are exactly the same as when using the specific driver, the default values for the case the parameter was omitted are different. The default values when using a specific driver were the ones that were defined by the IVI specifications, however, the default values when using the a class driver are the ones that are configured at the **Driver Session** in the IVI Configuration Store. In any cases, what explicitly specified through **`OptionString`** parameter of **`Initialize With Options`** function is the first priority.

6-2 Channel Access

When supporting power supply and/or oscilloscope instruments, the IVI instrument driver is generally designed assuming the instrument has multiple channels. Therefore, driver functions operating instrument panel settings often have the **`channel name`** parameter, which specifies the channel.

.

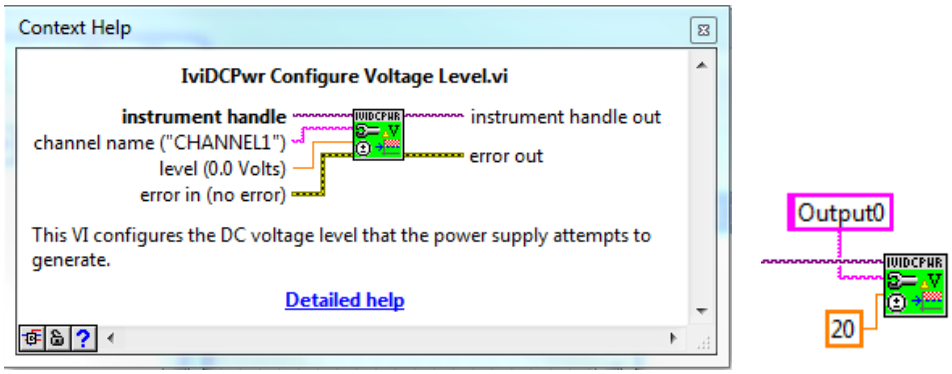

Figure 6-2 Configure Voltage Level.vi Help

This example uses the class driver specifying the channel name **"Output0"** that can only be applied to a specific instrument driver (kipwx driver in this case). This method can control the instrument, however, using names that depend on specific instrument driver will spoil interchangeability. For example, a valid channel name for AgN57xx instrument driver is **"Output1"**.

In above NI-MAX configuration, we added the virtual name **"Track_A"** and configured as it can be converted to the physical name **"Output0"**. Therefore we can use the virtual name for the channel name.

When exchanging the instrument drivers, change the IVI configuration for Hardware (VISA address for instrument I/O connection) set by Driver Session, Software (instrument driver to be used), and Physical Names (physical name to which the virtual name is mapped), so that operation can continue to work.

Notes:

- The setting info of IVI Configuration is stored in C:/ProgramData/IVI Foundation/IVI/IviConfigurationStore.xml. Do not edit this XML file by hands.
- The IVI Configuration is commonly shared between all 32bit/64bit T&M applications and all log-on users in the same PC.

6-3 Closing Session

To close the instrument driver session, use **IviDCPwr Close.vi**.

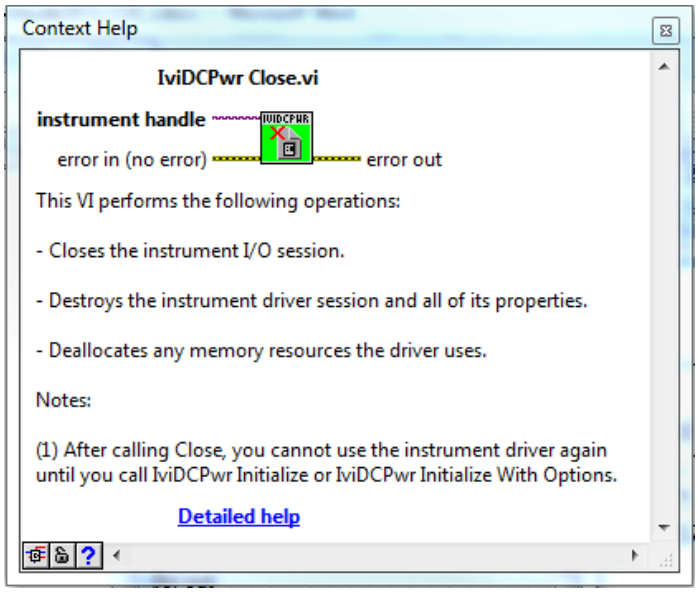

Figure 6-3 Close.vi Help

6-4 Exchanging Instruments

Example shown so far were set to use kipwx instrument driver as the virtual instrument configuration. Now what happens if changing the instrument to the one that is hosted by AgN57xx driver (Agilent N5700 series DC Power Supply)? In this case, you don't have to recompile/relink your application, however you have to change the configuration for IVI Logical Name (virtual instrument). Basically the configuration shall change:

- Software Module in Driver Session tab (kipwx➔AgN57xx)
- map target of Virtual Names (Output0➔Output1)
- IO Resource Descriptor in Hardware Asset (changing to the VISAaddress of post-swap instrument)

Once the configuration is properly set, the above example will function with the post-swap instrument without having to recompile.

Notes:

- The interchangeablity feature utilizing IVI class drivers does not guarantee the correct operation between pre-swapping and post-swapping instruments. Please make sure to confirm that your system correctly functions after swapping the instruments.

IVI Instrument Driver Programming Guide