IVI Instrument Driver Programming Guide (LabWindows/CVI Edition)

June 2012 Revision 2.1

1- Overview

1-1 Recommendation Of IVI-C Driver

LabWindows/CVI is a C language based (C++ unsupported) development environment. Therefore it is easier to use function-based DLL (C DLL) than handling COM DLL. Plus, the fundamental architecture of IVI-C Instrument Driver is what expanded from VXI Plug&Play Instrument Driver, which is in turn expanded from LabWindows/CVI Instrument Driver. (Common requirement of each driver architecture is different, but physical structure of the driver is still the same LabWindows/CVI 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 LabWindows/CVI 2010.

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.

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.

2-1 Creating Application Project

Launch LabWindows/CVI 2010, then from Welcome page select **New | Project** to create a new project. If the existing project is automatically read when launching without seeing Welcome page, select **File | New | Project (*.prj)** menu. Although this guidebook assumes that you use the IVI-C driver with a new application project, you can also apply the same manner for existing projects. Although this guidebook assumes that you use the IVI-C driver with a new application project, you can also apply the same manner for existing projects.

After creating the new project, it is recommended to save it first by choosing **File | Save Untitled.PRJ As...** menu. This example assumes that the project is Ex01.prj. Since there are no C source files yet immediately after creating the project, create a new source file with **File | New | Source (*.c)...** menu then store it as Ex01.C. Furthermore append the source file to the project by choosing **File | Add Ex01.c to Project** menu.

2-2 Loading Instrument Driver

Select **Instrument| Load** menu, then select kipwx.fp located in the **C:/Program Files (x86)/IVI Foundation/IVI/Drivers/kipwx** directory. Then the **Instrument** | **KikusuiPwx...** menu will be added.

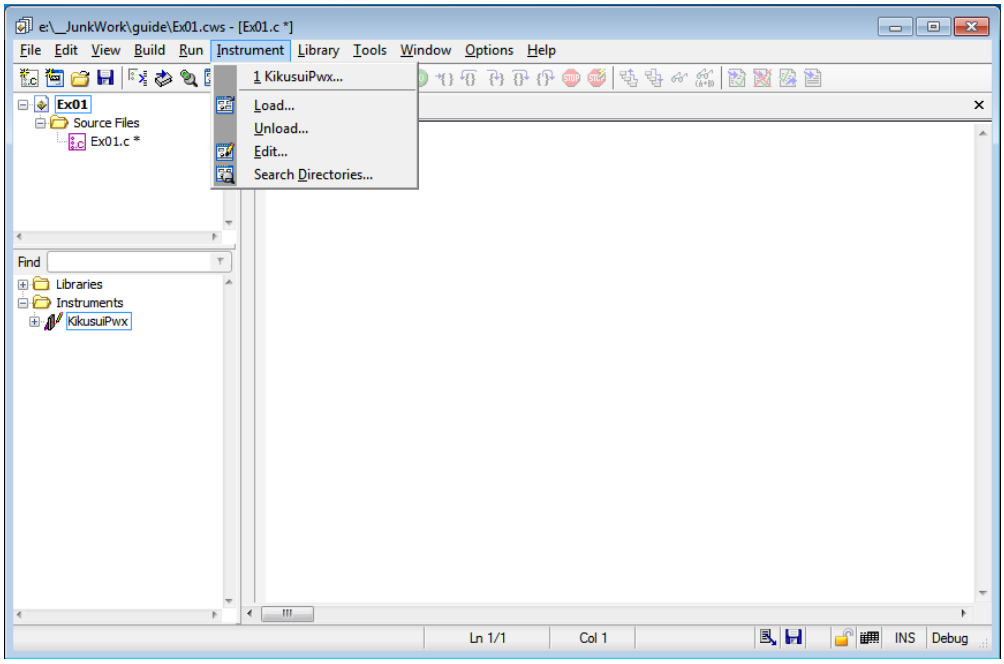

Figure 2-1 Instrument Menu

2-3 Writing Codes

Inserting Function Call

Open the C source code (Ex01.c) that was added to the project. Currently there is no code written. Next, select **Instrument | KikusuiPwx** menu.

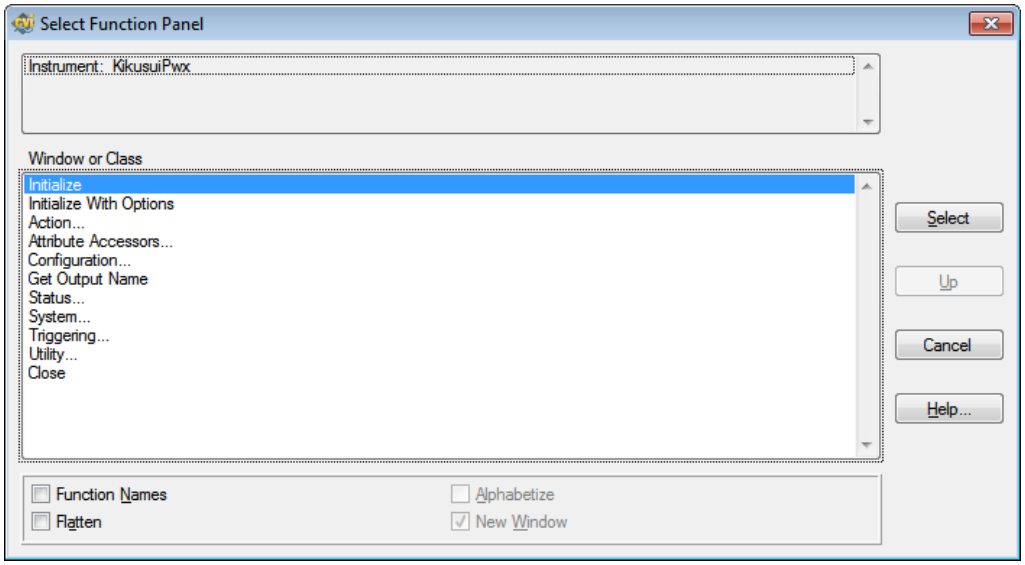

Figure 2-2 Select Function Panel

Select **Initialize With Options** then click the **Select** button. Then **Initialize With Options** function panel appears.

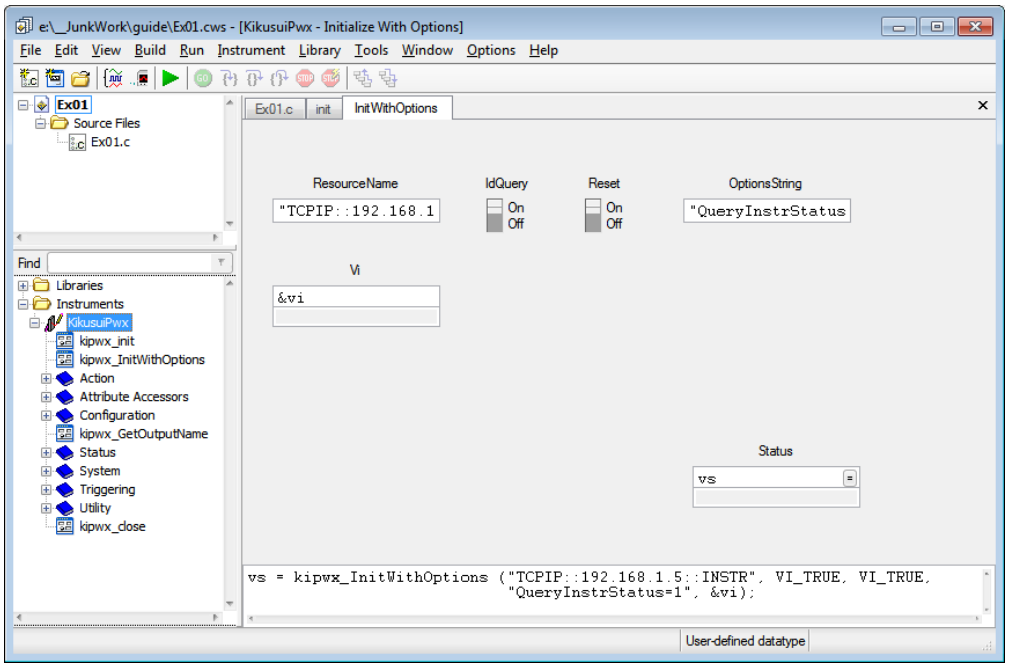

Figure 2-3 Initialize With Options function

Here, specify each parameter of the function as shown below.

Table 2-1 Each Parameter Value

Parameter	Value
ResourceName	"TCPIP::192.168.1.5::INSTR"
IdQuery	On (VI_TRUE as in function call code)
Reset	On (VI_TRUE as in function call code)
OptionString	"QueryInstrStatus=1"
Vi	&vi
Status	Vs

Notes:

- This example assumes that the instrument is connected through LAN interface having IP address 192.168.1.5.
- Do not forget to enclose **ResourceName** and **OptionString** with double-quotations because these parameters are string.

The variable **vi** you input here requires declaration. While selecting this control on the function panel, select **Code | Declare Variable...** menu (or Ctrl+D) to open **Declare Variable** dialogue, then check both **Execute declaration in Interactive Window** and **Add declaration to top of target file "Ex01.c"** and then click **OK**. Similarly the variable **vs** also requires declaration. Add declaration on the top of code in the same manner.

And then, insert a calling code of **kipwx_InitWithOptions** in the source code (Ex01.c) with **Code | Insert Function Call** menu (or Ctrl+I).

Similarly, from **Select Function Panel** dialog, select **Close** to show the **Close** function pane. Here type **vi** for the parameter **Vi**, type **vs** for **Status**. Then insert a calling code of **kipwx_close** in the source code (Ex01.c) with **Code | Insert Function Call** menu.

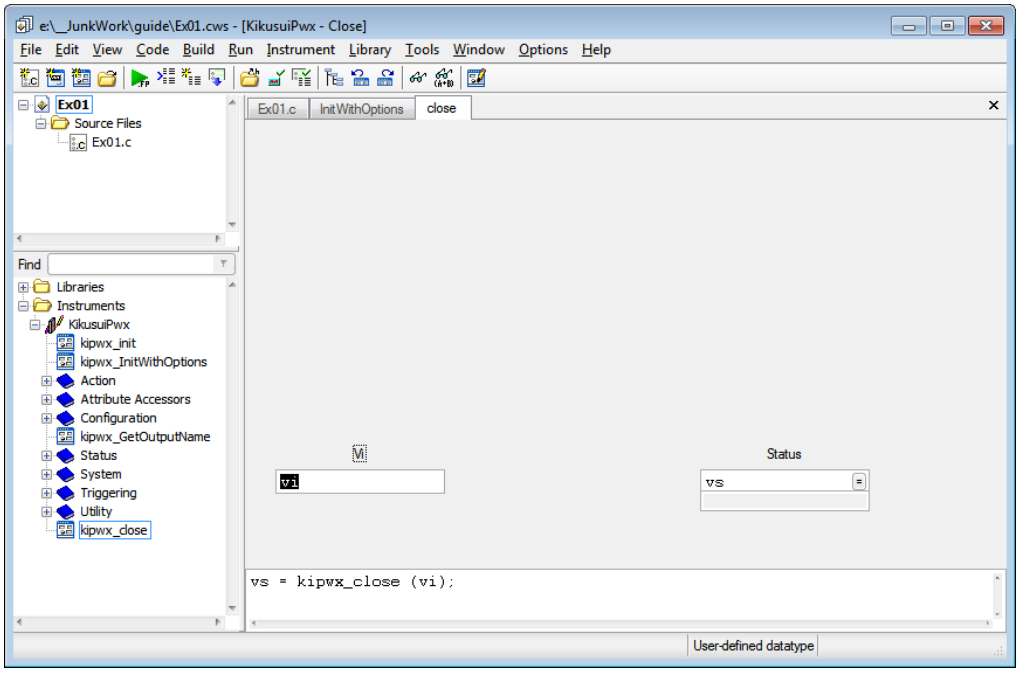

Figure 2-4 Close Function Panel

At this point of time, the C source code is like below.

```
static ViStatus vs;
```



```
static ViSession vi;

vs = kipwx_InitWithOptions ("TCPIP::192.168.1.5::INSTR", VI_TRUE, VI_TRUE,
    "QueryInstrStatus=1", &vi);

vs = kipwx_close (vi);
```

However, this is not a valid form for executable program. Enclose the entire block with the **main** function, then add an include line that specifies to read the instrument driver's include file. Finally add function calls, which configure voltage and current settings and output ON/OFF control, between the **InitWithOptions** and **close** calls. You may add source codes directly into the source code editor.

```
#include <kipwx.h>
static ViStatus vs;
static ViSession vi;

void main()
{
vs = kipwx_InitWithOptions ("TCPIP::192.168.1.5::INSTR", VI_TRUE, VI_TRUE,
    "QueryInstrStatus=1", &vi);

vs = kipwx_ConfigureVoltageLevel (vi, "", 20);
vs = kipwx_ConfigureCurrentLimit (vi, "", KIPWX_VAL_CURRENT_REGULATE, 2.0);
vs = kipwx_ConfigureOutputEnabled (vi, "", 1);

vs = kipwx_close (vi);

}
```

Building Project

Select **Build | Create Debuggable Executable** (Ctrl+M) menu to build your project. The build process will soon complete if there is no error especially.

2-4 Program Execution

Above example opens the instrument driver session, configures voltage/current/output, then immediately closes the session. Just executing the program, you can't see what is how executed because the program is not interactive. Here, set the Breakpoint on the function call code for **kipwx_InitWithOptions**. A breakpoint can be set with the **Run | Toggle Breakpoint** (or F9) menu.

Selecting **Run | Debug Ex01_dbg.exe** menu will execute the program, and the instruction automatically stops at the **kipwx_InitWithOptions** function call, where the Breakpoint is set. Select **Run | Step Over** (or F10) to execute that line.

Pay attention to the **vs** and **vi** values after **kipwx_InitWithOptions** is invoked. When the driver session has been successfully opened, **vi** contains the session handle (normally 0x00000001 or greater as an IVI handle), and **vs** contains the error code (0x00000000 if succeeded, or negative value if failed).

Pressing F10 furthermore, execute the **kipwx_ConfigureVoltageLevel** call and **kipwx_ConfigureCurrentLimit** call sequentially. For each case, the error code is stored in the **vs**.

If the **vs** value is negative, it is meant that a function call has failed generating an error. By adding the following code fragment, you can convert the error code to the corresponding human-readable English message.

```
char buf[256];
...
kipwx_error_message (VI_NULL, vs, buf);
```

Note:

- An IVI instrument driver for DC Power Supply is generally designed with considering multi-channel use, therefore many of instrument setting functions have the **ChannelName** parameter.
- When the instrument only has one channel, the **ChannelName** can be a black string, however explicit name is required when the instrument has multiple channels.
- In many case the **ChannelName** can be like **"Output1"** or **"Channel"** but they may be different for each IVI driver. For example, Kikusui PWX Series DC Power Supply assigns channel numbers from zero under Multi-Drop expanded operations, so the first channel name is **"Output0"**.

3- Description

3-1 Opening Instrument Driver Session

To open the driver session, the **kipwx_InitWithOptions** function is used. Although the prefix **kipwx_**, which is applied to each function is different on the instrument driver basis, such naming convention is applied to every IVI-C instrument driver.

```
vs = kipwx_InitWithOptions ("TCPIP::192.168.1.5::INSTR", VI_TRUE, VI_TRUE,
        "QueryInstrStatus=1", &vi);
```

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> init()** specifies **kipwx init()**for the kipwx instrument driver.
- Every driver function other than **<prefix> init()** and **<prefix> InitWithOptions()** has the 1st parameter as **ViSession** and the return value is all **ViStatus** type..
- **<prefix> init()** function is remained for the compatibility with VXI Plug&Play drivers. This is equivalent to **<prefix> InitWithOptions()** with exception that **OptionString** cannot be specified.

Now let 's talk about parameters of the **kipwx_InitWithOptions** function. Every IVI-C instrument driver has the **<prefix>_InitWithOptions** function that is defined by the IVI specifications. This function has the following parameters.

Table 3-1 Parameters for InitWithOptions function

Parameter	Type	Description
ResourceName	ViRsrc (const char*)	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).
idQuery	ViBoolean	Specifying VI_TRUE performs ID query to the instrument.
Reset	ViBoolean	Specifying VI_TRUE resets the instrument settings.
OptionString	ViConstString (const char*)	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.
Vi	ViSession*	Receives the instrument session. (The parameter is a pointer)

ResourceName specifies a VISA address (resource name). If **IdQuery** is VI_TRUE, the driver queries the instrument identities using a query command such as **"*IDN?"**. If **resetDevice** is VI_TRUE, the driver resets the instrument settings using a reset command such as **"*RST"**.

OptionString 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 **OptionString** 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 **OptionString** 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 **InitWithOptions** function, and its purpose and syntax are driver-specific. Therefore, specifying the **DriverSetup** must be at the last part on the **OptionString** parameter. Because the contents of **DriverSetup** are different depending on each driver, refer to driver's Readme document or online help.

3-2 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 2nd parameter, which specifies the channel.

```
Example :

vs = kipwx_ConfigureVoltageLevel( vi, "", 20.0);
```

This example uses the KikusuiPwx (prefix name is kipwx in IVI-C) driver that operates the Kikusui PWX DC power supply, the channel name can be blank (when the channel is only one) or **"Output0"**. Channel names are defined by the instrument driver for each, therefore different naming convention is applied on driver basis. Refer to the driver's on-line help for what channel names can be actually used.

3-3 Closing Session

To close the instrument driver session, use the **<prefix>_close** function.

.

```
vs = kipwx_close (vi);
```

4- Error Handling

In the previous example, there was no error handling processed. However, setting an out-of-range value to a function or invoking an unsupported function may generate an error from the instrument driver. Furthermore, no matter how the application is designed and implemented robustly, it is impossible to avoid instrument I/O communication errors.

When using IVI-C instrument drivers, every error generated in the instrument driver is transmitted to the client program through the return value as the **ViStatus** type.

Table 4-1 Rough classification of ViStatus

Value Range	Description
vs=0	Success
vs>0	Warning
vs<0	Error

Although you can identify what error is generated with the **ViStatus** return, you can convert the code to more readable message by using **error_message** function. This function exceptionally accepts the **VI_NULL** as the **ViSession** parameter. Make sure that the receive buffer has 256 bytes or larger area.

```
char buf[256];
...
kipwx_error_message (VI_NULL, vs, buf);
```

5- Example Using Class Driver

Now we explain how to use class interfaces. By using class interfaces, 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. invoking the `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.

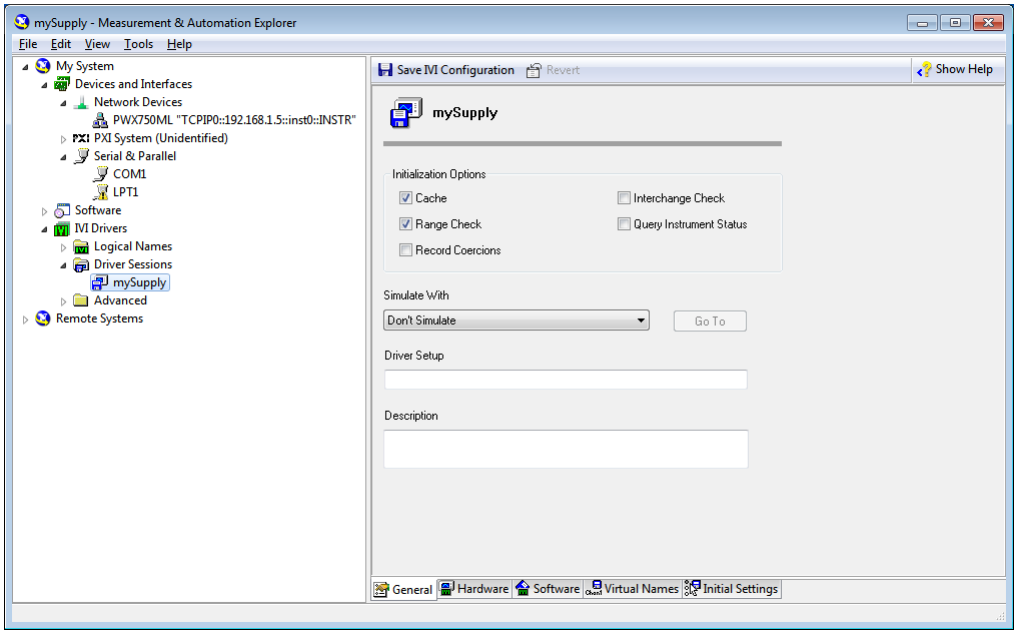

Figure 5-1 NI-MAX General Tab

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**.

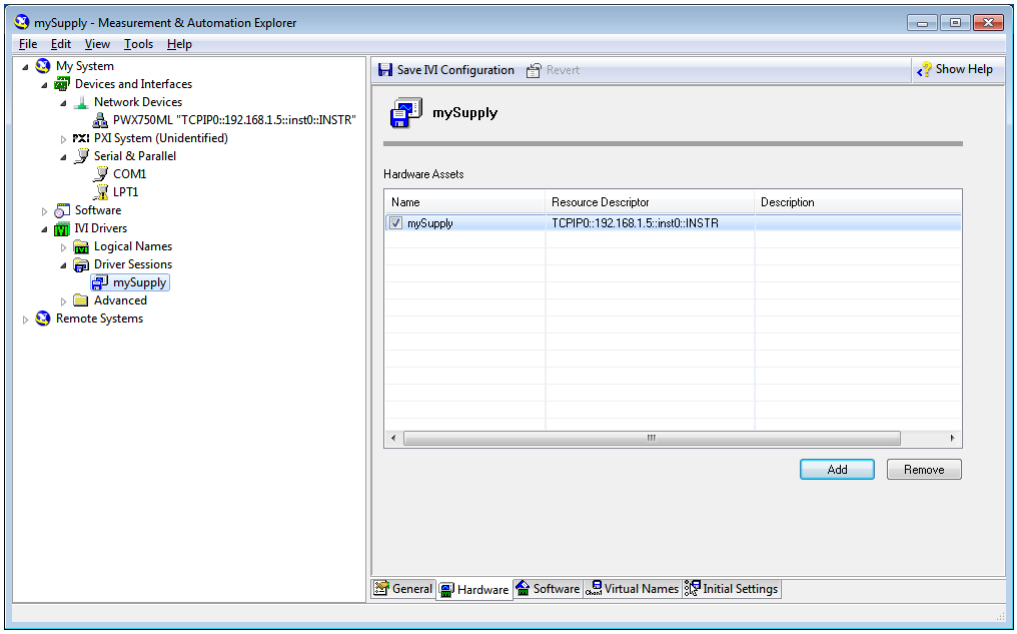

Figure 5-2 NI-MAX Hardware Tab

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.

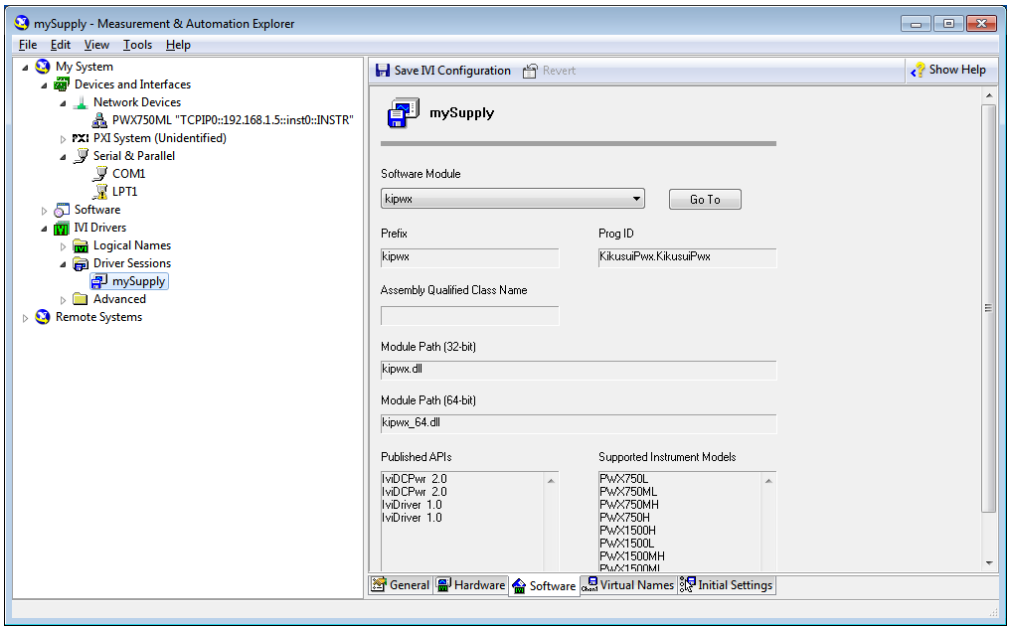

Figure 5-3 NI-MAX Software Tab

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.

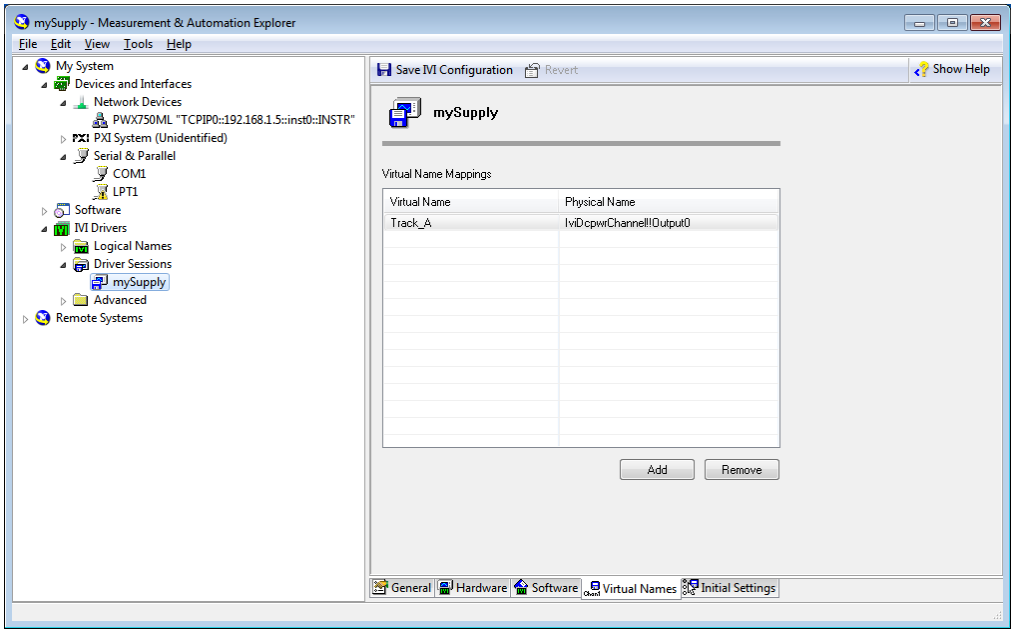

Figure 5-4 NI-MAX Virtual Names Tab

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.

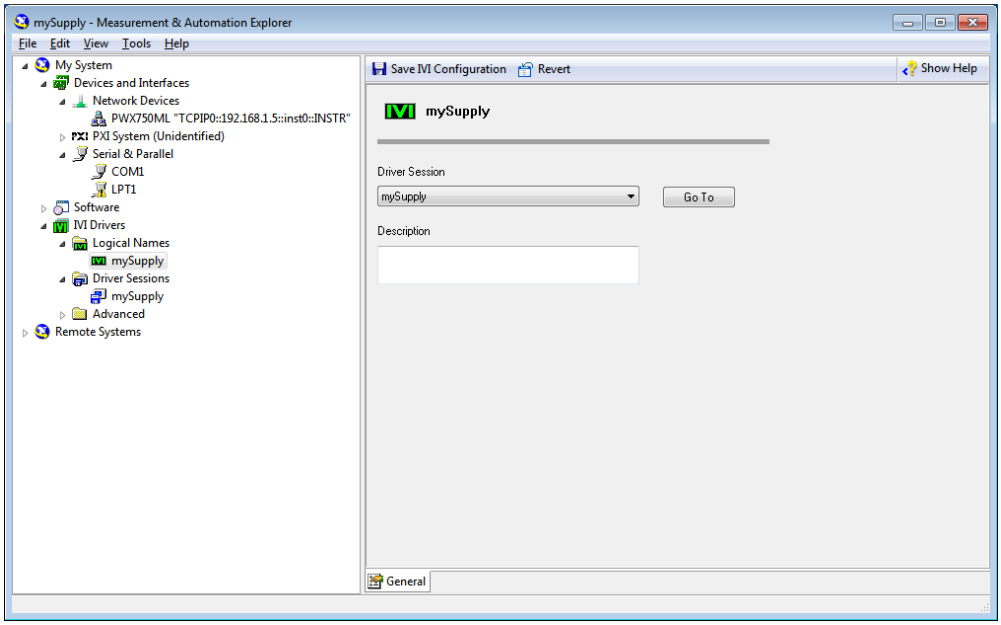

Figure 5-5 NI-MAX General Tab

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 Creating Application Project

Here launch LabWindows/CVI.

As you launch the LabWindows/CVI integrated environment, a new application project is created. If the existing project is automatically read when launching, select **File | New | Project (*.prj)** menu. Although this guidebook assumes that you use the IVI-C driver with a new application project, you can also apply the same manner for existing projects.

After creating the new project, it is recommended to save it first by choosing **File | Save Untitled.PRJ As...** menu. This example assumes that the project is Ex02.prj. Since there are no C source files yet immediately after creating the project, create a new source file with **File | New | Source (*.c)...** menu then store it as Ex02.C. Furthermore append the source file to the project by choosing **File | Add Ex02.c to Project** menu.

5-3 Loading Instrument Driver

Select **Instrument| Load** menu, then select **IviDCPwr.fp** located in the **C:/Program Files (x86)/IVI Foundation/IVI/Drivers/ividcpwr** directory. Then the **Instrument | IviDCPwr Class Driver** menu will be added.

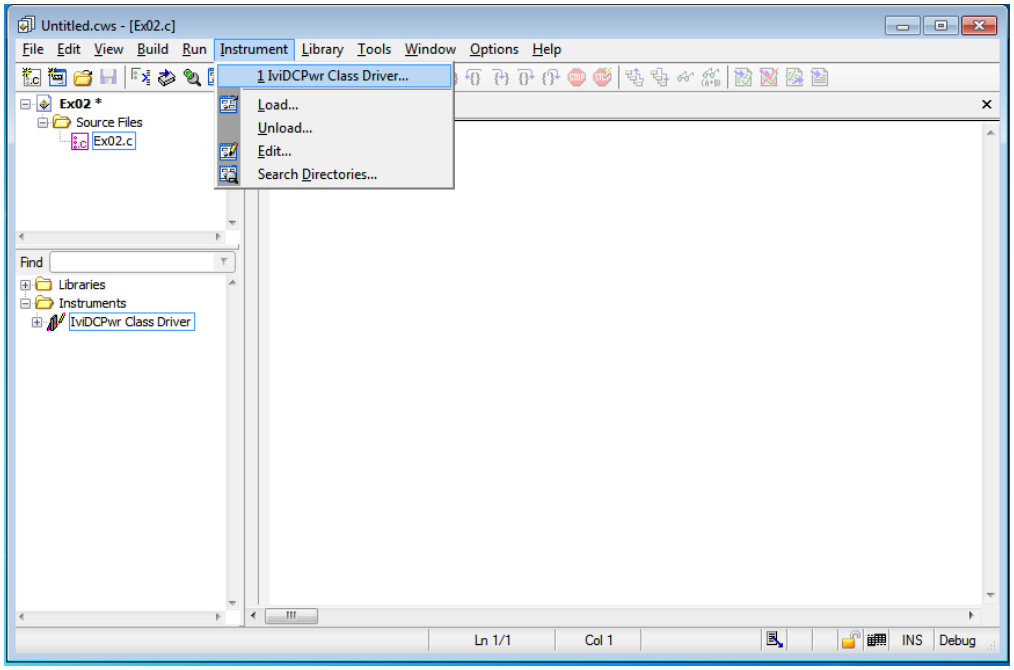

Figure 5-6 Instrument Menu

5-4 Writing Codes

Inserting Function Call

Open the C source code (Ex02.c) that was added to the project. Currently there is no code written. Next, select **Instrument | IviDCPwr Class Driver** menu.

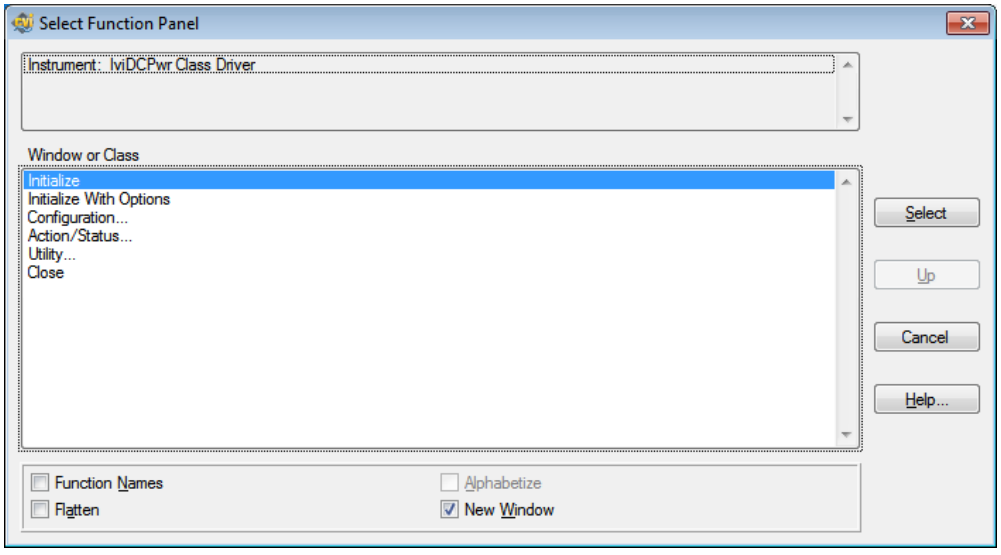

Figure 5-7 Select Function Panel

Select **Initialize With Options** then click the **Select** button. Then the Initialize With Options function panel appears. Here, type **`&vi`** for **Instrument Handle**, and type **`vs`** for **Status**.

The variable **`vi`** you input requires declaration. While selecting this control on the function panel, select **Code | Declare Variable...** menu (or Ctrl+D) to open **Declare Variable** dialogue, then check both **Execute declaration in Interactive Window** and **Add declaration to top of target file "Ex02.c"** and then click **OK**. Similarly the variable **`vs`** also requires declaration. Add declaration on the top of code in the same manner.

Keep the **Option String** default. At **Logical Name**, specify **"mySupply"** that was configured in the NI-MAX previously. After that, select **Code | Insert Function Call** menu to insert the function call code for **IviDCPwr_InitWithOptions** into the source code (Ex02.c).

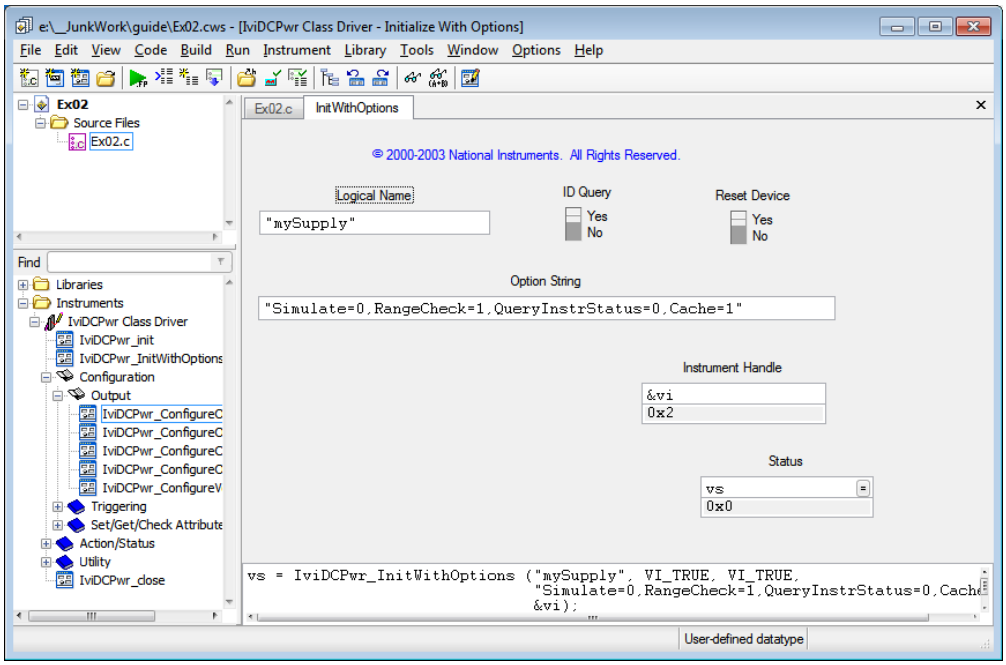

Figure 5-8 Initialize With Options Function Panel

Similarly, select **Close** at the Select Function Panel dialog to show the Close function panel. Here, type `vi` for **Instrument Handle** and type `vs` for **Status**. After that, select **Code | Insert Function Call** menu to insert the function call code for **IviDCPwr_close** into the source code (Ex02.c).

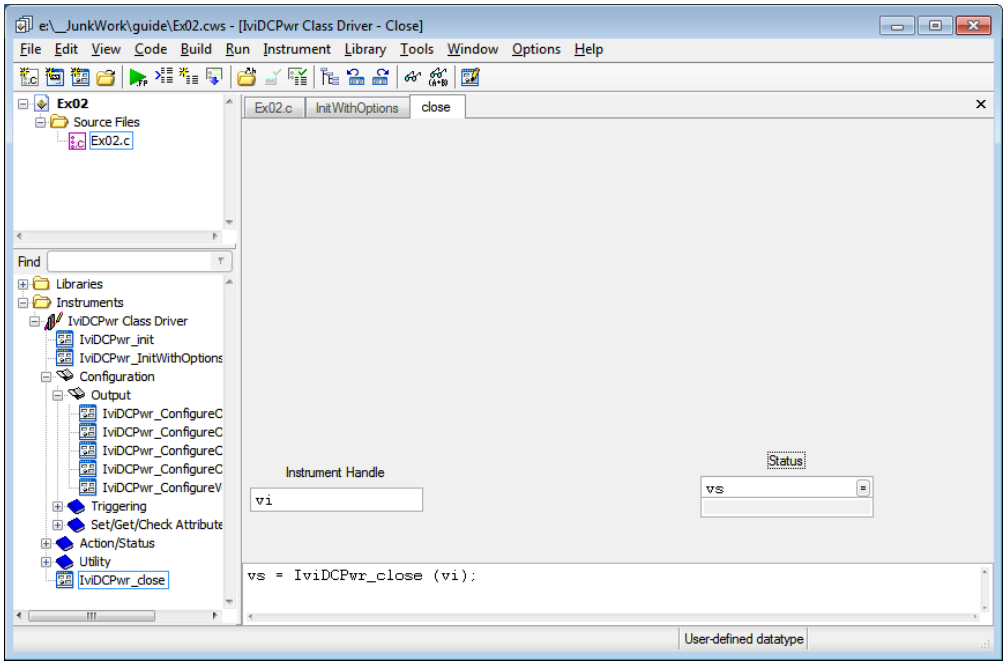

Figure 5-9 Close Function Panel

At this point of time, the C source code is like below:

```
static ViStatus vs;
static ViSession vi;
```

```
vs = IviDCPwr_InitWithOptions ("mySupply", VI_TRUE, VI_TRUE,
       "Simulate=0,RangeCheck=1,QueryInstrStatus=0,Cache=1", &vi);

vs = IviDCPwr_close (vi);
```

However, this is not a valid form for executable program. Enclose the entire block with the **main** function, then add an include line that specifies to read the class driver's include file. Furthermore add variable declarations for **vi** and **vs**. Finally add function calls, which configure voltage and current settings and output ON/OFF control, between the **InitWithOptions** and **close** calls. You may add source codes directly into the source code editor.

```
#include <IviDCPwr.h>
static ViSession vi = 0;
static ViStatus vs = 0;

void main()
{

vs = IviDCPwr_InitWithOptions ("mySupply", VI_TRUE, VI_TRUE,
       " Simulate=0,RangeCheck=1,QueryInstrStatus=0,Cache=1", &vi);

vs = IviDCPwr_ConfigureVoltageLevel (vi, "Track_A", 20);
vs = IviDCPwr_ConfigureCurrentLimit (vi, "Track_A",
IVIDCPWR_VAL_CURRENT_REGULATE, 2.0);
vs = IviDCPwr_ConfigureOutputEnabled (vi, "Track_A", 1);

vs = IviDCPwr_close (vi);

}
```

Building Project

Select **Build | Create Debuggable Executable** menu to build your project. The build process will soon complete if there is no error especially.

6- Description

6-1 Opening Class Driver Session

To open the driver session, the **IviDCPwr_InitWithOptions** function is used. The prefix **IviDCPwr_** 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).

```
vs = IviDCPwr_InitWithOptions ("mySupply", VI_TRUE, VI_TRUE,
       "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1", &vi);
```

Class drivers are different with normal instrument drivers, thus you cannot pass VISA address to the **InitiWithOptions** function 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_InitWithOptions** indirectly.

Although the contents for **OptionString** (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 `Init WithOptions` 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 2nd parameter, which specifies the channel.

例：

```
vs = IviDCPwr_ConfigureVoltageLevel( vi, "Output0", 20.0);
```

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.

```
vs = IviDCPwr_ConfigureVoltageLevel( vi, "Track_A", 20.0);
```

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` function.

```
vs = IviDCPwr_close (vi);
```

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.

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
Product names and company names that appear in this guidebook are trademarks or registered trademarks of their respective companies.