IVI-COM Instrument Driver Programming Guide (LabWindows/CVI Edition)

Sep 2005 Revision 2.0

1- Overview

1-1 Supporting IVI-C Drivers

LabWindows/CVI is a development environment that assumes to use the C language. Although it is possible to use IVI-COM instrument drivers directly, it is much easier to program utilising IVI-C or VXI Plug&Play instrument drivers if they are supported. IVI-COM instrument drivers from Kikusui provide not only IVI-COM interfaces, but also support IVI-C programming interfaces. The IVI-C specifications are the advanced versions of VXI Plug & Play instrument driver specifications, therefore they are the most suitable driver types for use with LabWindows/CVI.

Notes:

IVI-COM instrument drivers from Kikusui do support both IVI-COM and IVI-C interfaces, as long as the driver version is 2.x.x.x or later. Mind that any versions 1.x.x.x or prior do not support IVI-C.

To use IVI-C instrument drivers, you must separately install NI IVI Compliance Package 2.x. This software package is not automatically installed with Kikusui's IVI driver setup program. Also, not all the LabWindows/CVI versions do install it.

This guidebook assumes that you use IVI-COM Kikusui4800 instrument driver (for KIKUSUI PIA4800 series DC Power Supply Controller). You can also use IVI-COM instrument drivers for other models in the same manner.

This guidebook assumes that you use LabWindows/CVI 7.1.

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 utilise the most of features of the instrument you use. The later is to utilise instrument class interfaces that are defined in the IVI specifications allowing to utilise interchangability 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➔Program➔IVI folder.

If the instrument driver does not belong to any instrument classes, you can't utilise class interfaces. This means that you cannot develop applications that utilise interchangability features.

2- Example Using Specific Interfaces

Here we introduce an example using specific interfaces. By using specific interfaces, you can utilise the maximum power of driver features but you have to spoil interchangability.

2-1 Creating An Application Project

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 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 **ki4800.fp** located in the **Program Files/IVI/Drivers/ki4800** directory. Then the **Instrument** | **Kikusui PIA4800 Power Supply Controller** menu will be added.

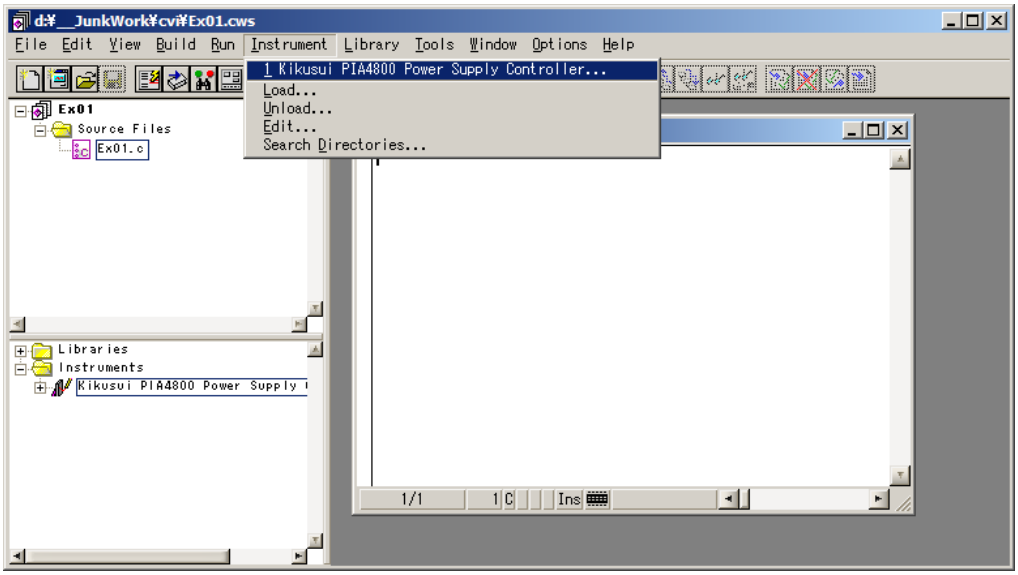


Figure 2-1 Instrument Menu

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 | Kikusui PIA4800 Power Supply Controller** menu.

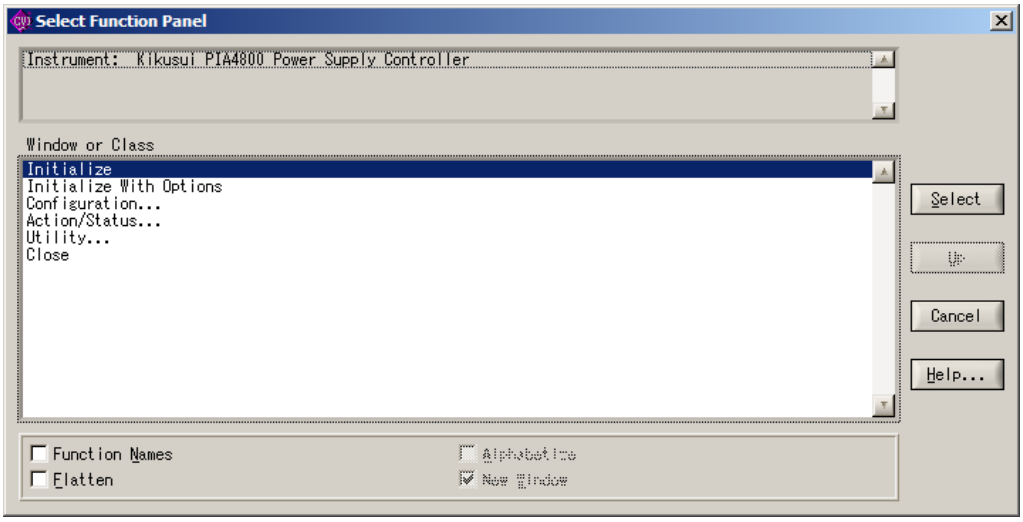


Figure 2-2 Select Function Panel

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**. Keep the **Option String** default. Specify the VISA address through which your instrument is connected for **Resource Name**. After that, select **Code | Insert Function Call** menu to insert the function call code for `ki4800_InitWithOptions()` into the source code (Ex01.c).

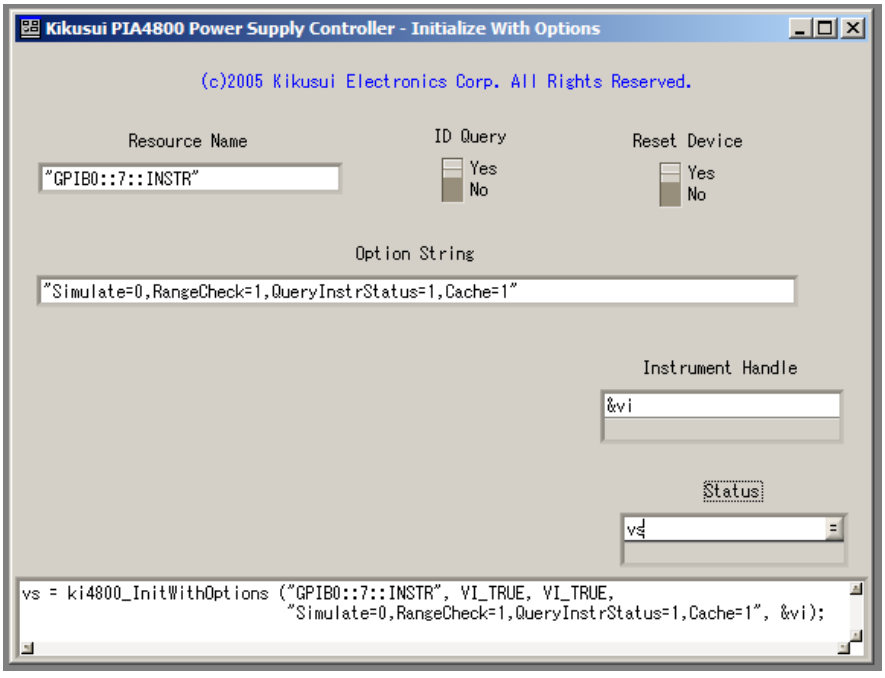


Figure 2-3 Initialize With Options Function Panel

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 `ki4800_close()` into the source code (Ex01.c).

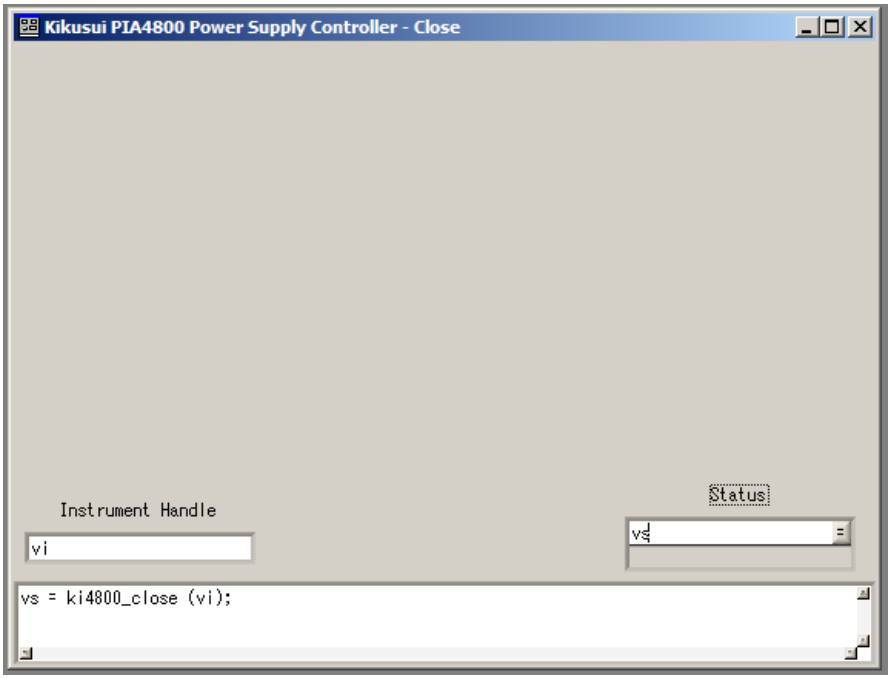


Figure 2-4 Close Function Panel

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

```
vs = ki4800_InitWithOptions ("GPIB0::7::INSTR", VI_TRUE, VI_TRUE,
    "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1", &vi);

vs = ki4800_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. 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 <ki4800.h>

void main()
{
ViSession vi = 0;
ViStatus vs = 0;

vs = ki4800_InitWithOptions ("GPIB0::7::INSTR", VI_TRUE, VI_TRUE,
    "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1", &vi);

vs = ki4800_ConfigureVoltageLevel (vi, "N5!C1", 20);
vs = ki4800_ConfigureCurrentLimit (vi, "N5!C1",
KI4800_VAL_CURRENT_REGULATE, 2.0);
vs = ki4800_ConfigureOutputEnabled (vi, "N5!C1", 1);

vs = ki4800_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.

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 `ki4800_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 `ki4800_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 `ki4800_InitWithOptions()` is invoked. When the driver session has been successfully opened, `vi` contains the session handle (normally 0x00000001 or greater), and `vs` contains the error code (0x00000000 if succeeded, or negative value if failed).

Pressing F10 furthermore, execute the `ki4800_ConfigureVoltageLevel()` and `ki4800_ConfigureCurrentLimit()` 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];
...
ki4800_error_message (VI_NULL, vs, buf);
```

Note:

The KIKUSUI PIA4800 series DC Power Supply Controller requires a couple of minutes to detect connectivity conditions of the DC power supplies through the TP-BUS. It should not be performed at every Initialize call. Therefore, it is necessary to configure the connectivity conditions of the DC power supplies to be controlled in advance by using Scan Utility. Make sure to run the Scan Utility when the first time to use the driver. To run the utility, choose [Start] button → All Programs → IVI → Kikusui4800 → Scan Utility menu.

You will need to run Scan Utility once again if you have changed number of DC power supplies, node addresses, communication interfaces (GPIB/RS232), and/or the port number or the GPIB address.

The configuration work with Scan Utility is specific to the IVI-COM Kikusui4800(ki4800) driver. No need to do it for other instrument drivers.

3- Description

3-1 Opening Session

To open the driver session, the `ki4800_InitWithOptions()` function is used. Although the prefix `ki4800_`, 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 = ki4800_InitWithOptions ("GPIB0::7::INSTR", VI_TRUE, VI_TRUE,
        "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1", &vi);
```

Notes:

As a technical term of IVI-C and VXI Plug&Play instrument drivers, "<prefix>" is often used. This is a name that identifies each of specific instrument drivers, and "ki4800" is the one in this document. For example, a generic expression <prefix>_init(), designates the ki4800_init() function for the ki4800 instrument driver.

Every function, except for <prefix>_init() and <prefix>_InitWithOptions(), takes ViSession as the 1st parameter and returns ViStatus.

The <prefix>_init() function is remained for the compatibility to VXI Plug&Play driver specifications. The function is equivalent with <prefix>_InitWithOptions() with an exception that the OptionString parameter cannot be specified.

Now let 's talk about parameters of the `ki4800_InitWithOptions` function. Every IVI instrument driver has an `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. If the instrument has the address 3 on the GPIB board #0, for example, it can be `GPIB0::3::INSTR`.
`IDQuery`	`ViBoolean`	Specifying `VI_TRUE` performs ID query to the instrument.
`resetDevice`	`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.

`newVi`	`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 electronic load 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 = ki4800_ConfigureVoltageLevel( vi, "N5!C1", 20.0);
```

As above example uses the Kikusui4800 (ki4800) driver that operates the DC power supply, we use a channel name that contains the NODE and CHANNEL. The channel name `"N5!C1"` in the above example is specific to an instrument driver, 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.

This example sets 20V for the DC power supply, which is connected at NODE5 and CHANNEL1 of the PIA4800 series DC Power Supply Controller.

3-3 Closing Session

To close the instrument driver session, use the `close` function.

```
vs = ki4800_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];
...
ki4800_error_message (VI_NULL, vs, buf);
```

5- Example Using Class Interfaces

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 interchangability available between different instrument classes.

5-1 Virtual Instrument

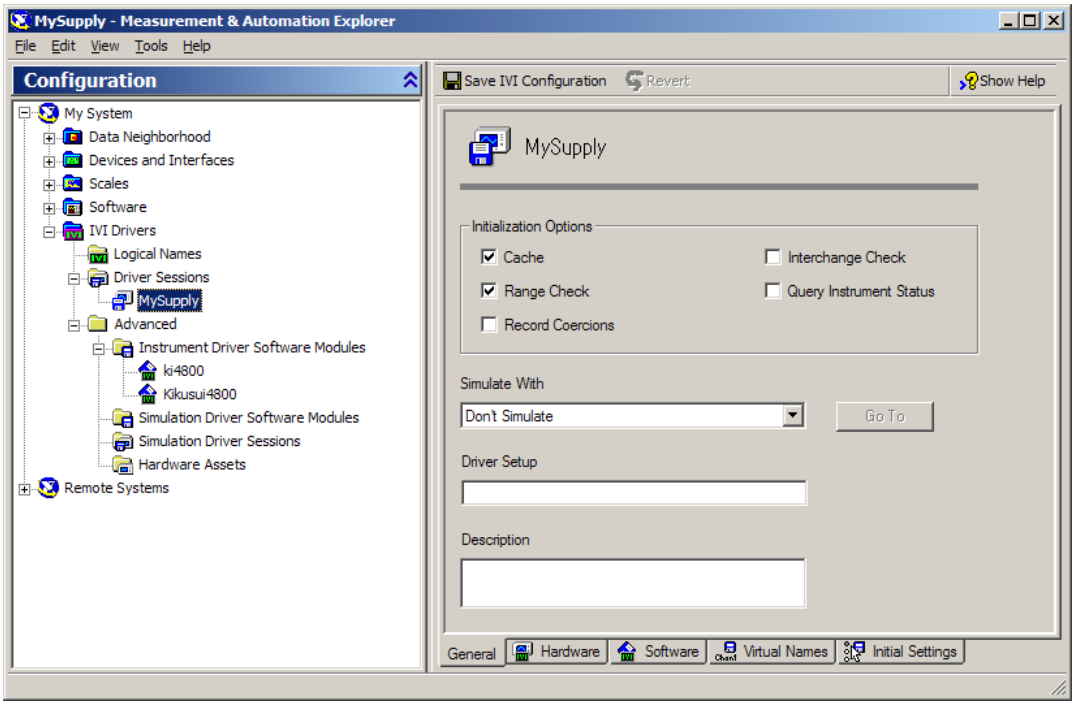
What you have to do before creating an application that utilises interchangability features is create a virtual instrument. To realise interchangability features, you should not write codes that are very specific to a particular IVI-C instrument driver (e.g. invoking the ki4800_init() function) and should not write a specific VISA address (resource name) such as "GPIB0::3::INSTR". Writing them directly in the application spoils interchangability.

Instead, the IVI specifications define methods to realise interchangability 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 /Program Files/IVI/Data/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.

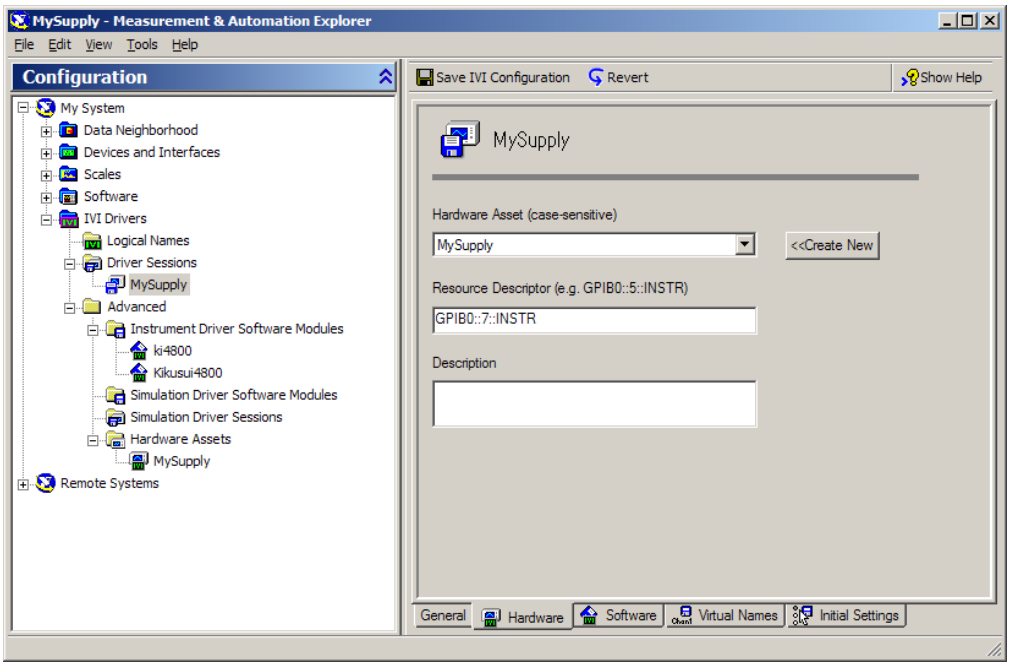
Creating Driver Session

After launching NI-MAX, refer to the **IVI Drivers** node on the tree. Right-click on the **Driver Session** then select **Create New** menu to create a new Driver Session. Being asked for its name, give the name `"MySupply"`.



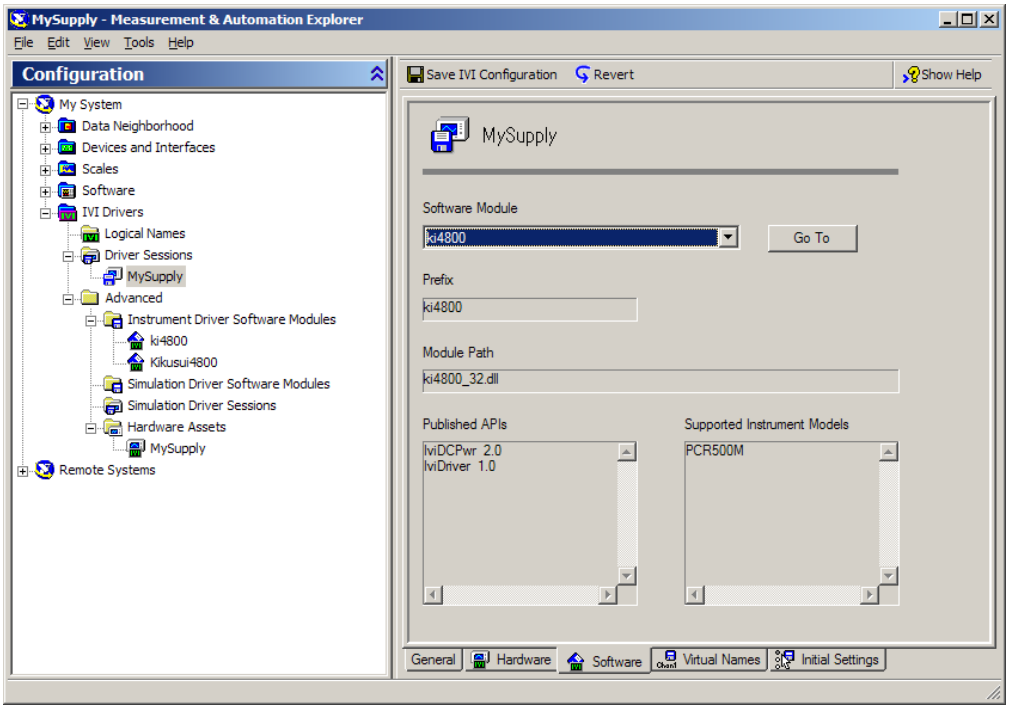
Creating Hardware Asset

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 **Create New** button to create a new Hardware Asset. Being asked for its name, give the name `"MySupply"` again, then click the **Create** button. Furthermore specify a valid VISA address though which your instrument is connected, as **Resource Descriptor**.



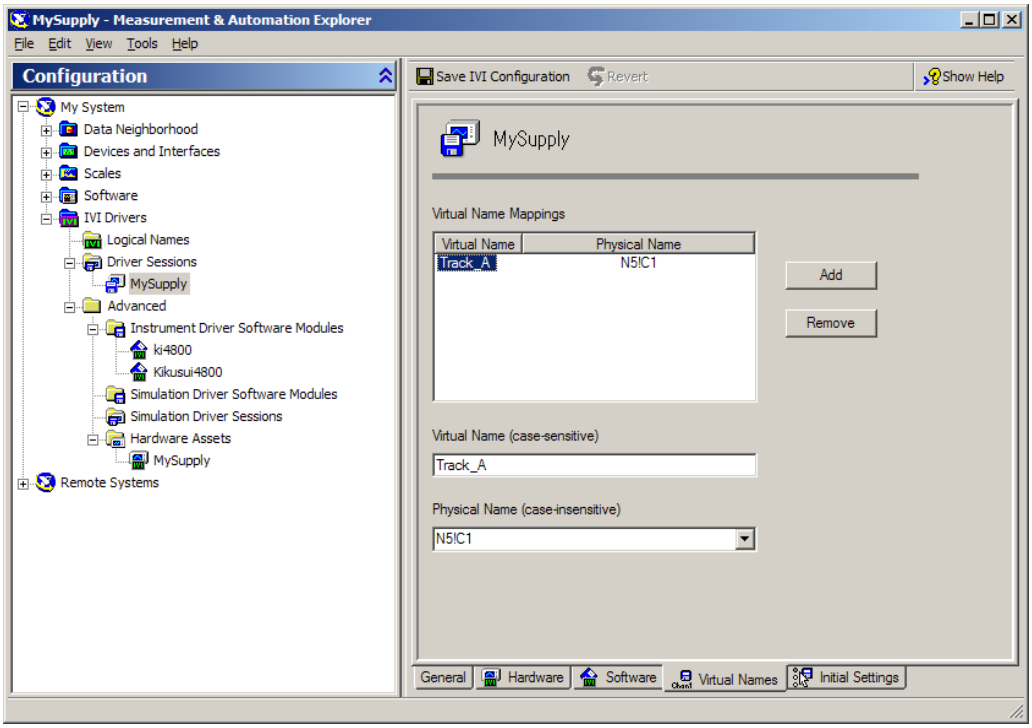
Setting Linkage for Software Module

Subsequently select the **Software** tab to show the software module management screen. The software module specifies the instrument driver module (DLL module). Here select **ki4800** from the **Software Module** list.



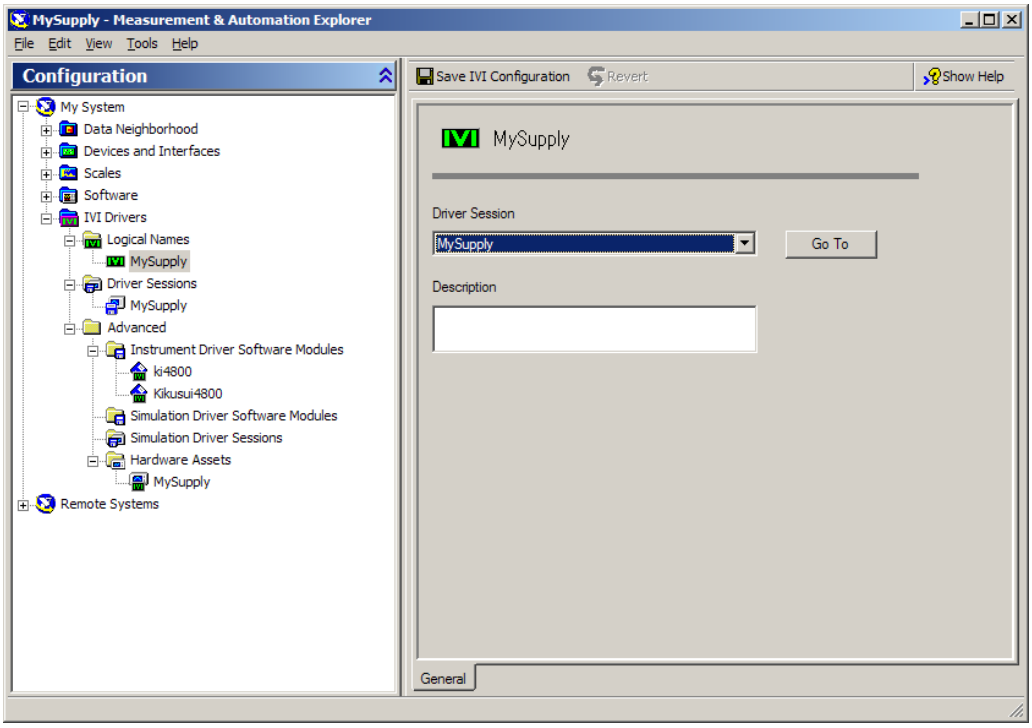
Creating Virtual Name

Subsequently select the **Virtual Names** tab to show the virtual name management screen. Normally, when channel names are related such as for power supply or electronic load 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, select an appropriate name suitable for the NODE/CHANNEL from **Physical Name**, through which your actual DC power supply is connected. The example shown below selects **N5!C1**, indicating it is connected through NODE 5, CHANNEL 1.



Creating Logical Name

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** menu to create the new logical name. Being asked for its name, give the name `"MySupply"`. Furthermore, select `"MySupply"` from the **Driver Session** list.



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 An Application Project

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 **Program Files/IVI/Drivers/ividcpwr** directory. Then the **Instrument | IviDCPwr Class Driver** menu will be added.

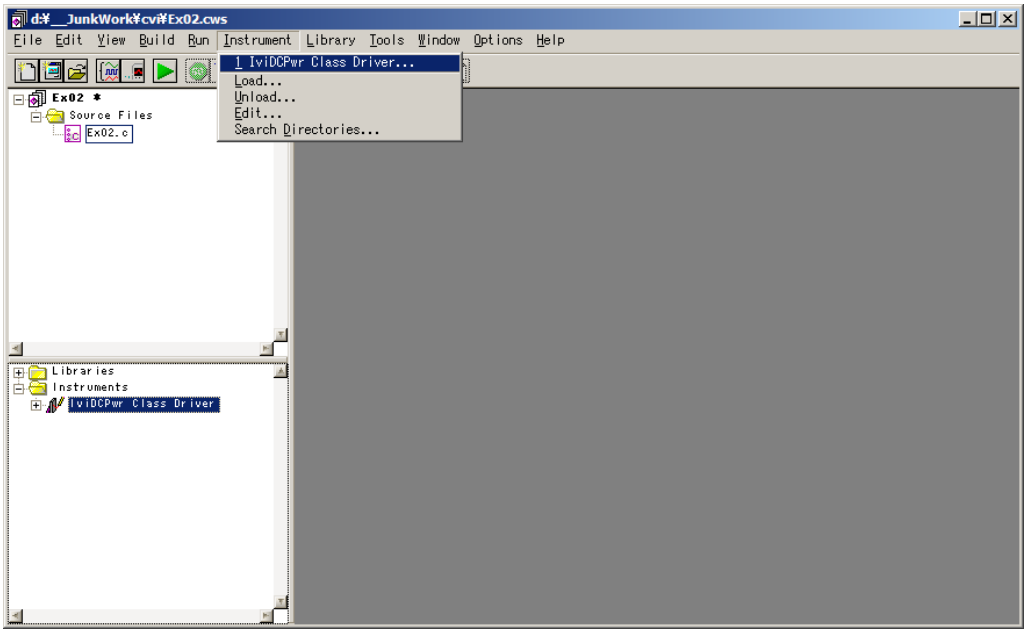


Figure 5-1 Instrument Menu

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.

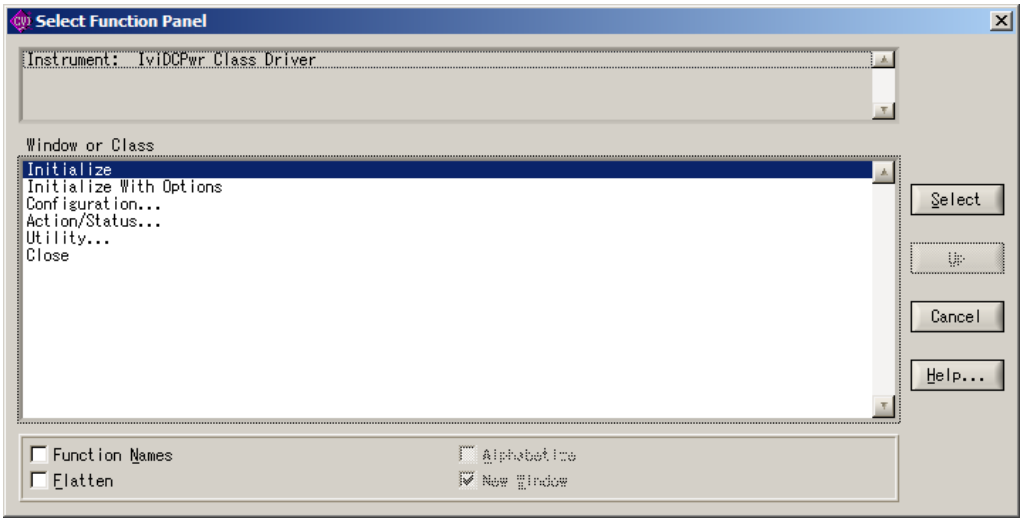


Figure 5-2 Select Function Panel

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**. 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).

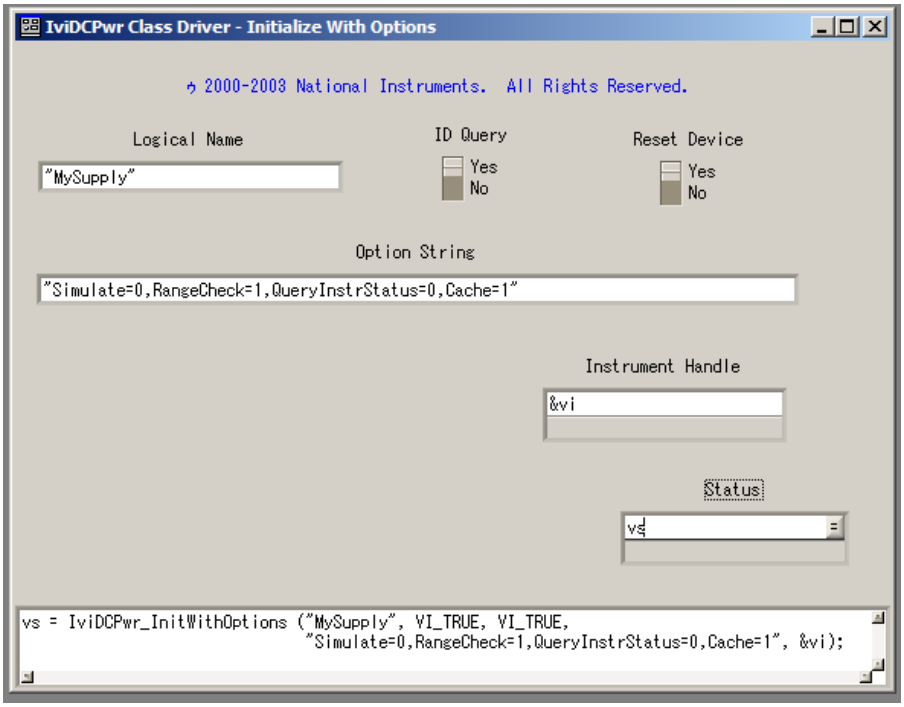


Figure 5-3 Initialize With Options Function Panel

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).

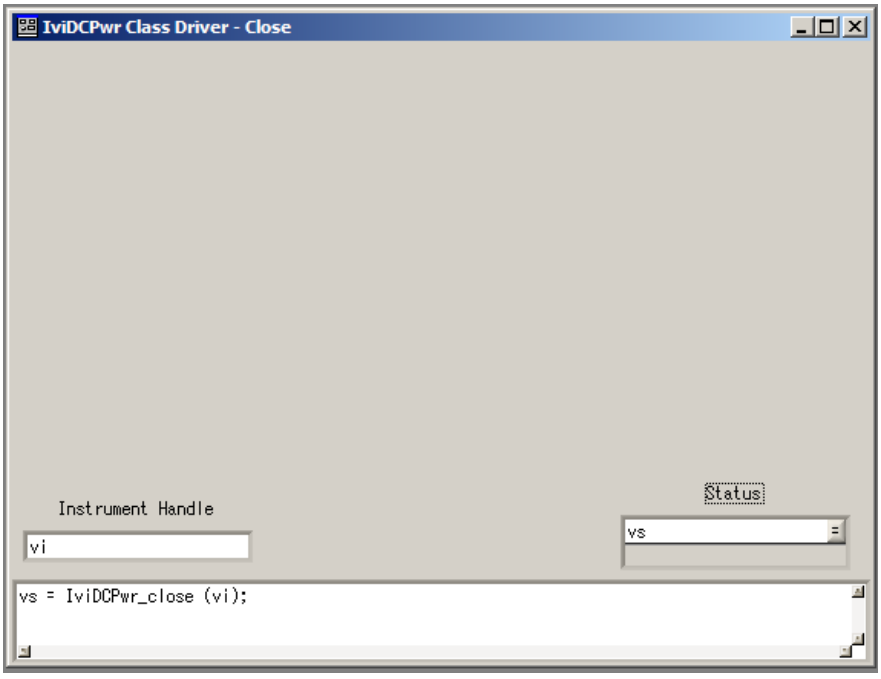


Figure 5-4 Close Function Panel

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

```
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>

void main()
{
ViSession vi = 0;
ViStatus vs = 0;

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.

5-5 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 `IviDCPwr_InitWithOptions()`. A breakpoint can be set with the **Run | Toggle Breakpoint** (or F9) menu.

Selecting **Run | Debug Ex02_dbg.exe** menu will execute the program, and the instruction automatically stops at the `IviDCPwr_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 `IviDCPwr_InitWithOptions()` is invoked. When the driver session has been successfully opened, `vi` contains the session handle (normally 0x00000001 or greater), and `vs` contains the error code (0x00000000 if succeeded, or negative value if failed).

Pressing F10 furthermore, execute the `IviDCPwr_ConfigureVoltageLevel()` and `IviDCPwr_ConfigureCurrentLimit()` sequentially. For each case, the error code is stored in the `vs`.

6- Description

6-1 Opening Session

To open the driver session, the `IviDCPwr_InitWithOptions()` function is used. The prefix `IviDCPwr` is specific to the IviDCPwr class driver.

```
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 `IviDCPwr_InitWithOptions()` 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 `ki4800_InitWithOption()` function indirectly.

Although the contents for OptionString 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.

6-2 Channel Access

When supporting power supply and/or electronic load 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 = IviDCPwr_ConfigureVoltageLevel( vi, "N5!C1", 20.0);
```

Above example uses the class driver, however, the channel name is specified to `"N5!C1"`, which is specific to a particular instrument driver (ki4800 driver in this case). Although this approach could control the instrument, using such driver-specific names makes the interchangeability being spoiled.

In above NI-MAX configuration, we added the virtual name `"Track_A"` and configured as it can be converted to the physical name `"N5!C1"`. Therefore we can use the virtual name for the channel name.

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

6-3 Closing Session

To close the instrument driver session, use `IviDCPwr_close` function.

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