IVI Instrument Driver Programming Guide LabWindows/CVI Edition

Oct 2020 Revision 2.2

Contents

Introduction

This guide shows examples of using a KikusuiPwr IVI Instrument Driver (KIKUSUI PWR-01 series DC power supply). IVI instrument driver made by other manufacturers and ones for other models can be used in almost the same way.

This guide explains using the LabWindows/CVI 2019 referring to the case to create a 32-bit (x86) program that runs on Windows 10 (x64) as an example.

Instrument driver to use with LabWindows/CVI

This guide recommends the IVI-C instrument driver and explains referring to the procedure of programming using the IVI-C instrument driver as an example.

Since LabWindows/CVI is a development environment assuming the use of C language, it is easier to use a function-based C DLL than a COM DLL. Since the IVI-C instrument driver is created by extending the architecture for the LabWindows/CVI instrument driver, it is compatible with LabWindows/CVI.

Usable Interfaces

The IVI instrument driver supports the following two interfaces:

- **Specific interface**
  It is the interface specific to instrument drivers. It will enable you to make the most of the functions of the instrument to be used.
- **Class interface**
  It is the interface of instrument class defined in the IVI specifications.
  Interchangeability is available, but the use of model-specific functions will be limited.

This guide explains how to program using each interface.

Memo

- The instrument class to which the instrument driver belongs can be checked from the Windows start menu's [Kikusui] > [KikusuiPwr IVI Driver 1.0.0 Documentation].
- In the case that the instrument driver does not belong to any instrument class, no applications can be created that use interchangeability as the class interface is not available.

Programming Using Specific Interfaces

This section describes the procedures for programming using a specific interface. The specific interface will enable you to make the most of the functions specific to the instrument driver.

Memo

In the case of a specific interface, the interchangeability function cannot be used. When using the interchangeability function, be sure to use the class interface. (p.17)

Preparation for programming

Creating a new project

An example of using the IVI-C driver in a new application is explained below:

1 **Start LabWindows/CVI.**

Welcome page is displayed.

2 **Click [New] > [Project].**

If the welcome page is not displayed and the existing projects are imported, click [File] > [New] > [Project (*.prj)].

A new project will be created.

3 **Click [File] > [Save Untitled.prj As] to save the project.**

In this case, save it as “Ex01.prj”.

4 **Click [File] > [New] > [Source (*.c)].**

A source file is created.

5 **Click [File] > [Save Untitled1.c As] to save the source file.**

In this case, save it as “Ex01.c”.

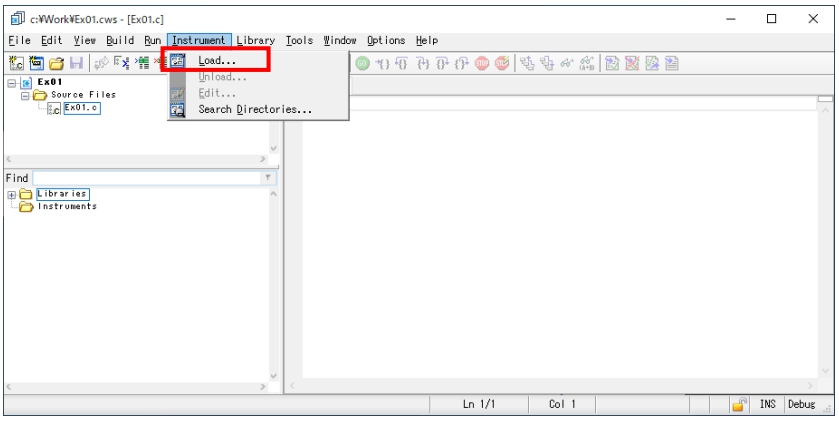
6 **Click [File] > [Add Ex01.c to Project].**

The source file is added to the project.

This completes the creation of a new project.

Loading an instrument driver

1 **Click [Instrument] > [Load].**



2 **Load “kipwr.fp” saved in “C:/Program Files (x86)/IVI Foundation/IVI/Drivers/kipwr” directory.**

In the [Instrument] menu, [KikusuiPwr01] will be added.

Configuring the program

Setting the Initialize With Options function

Set the Initialize With Options function, declare the variables and insert them into the source code.

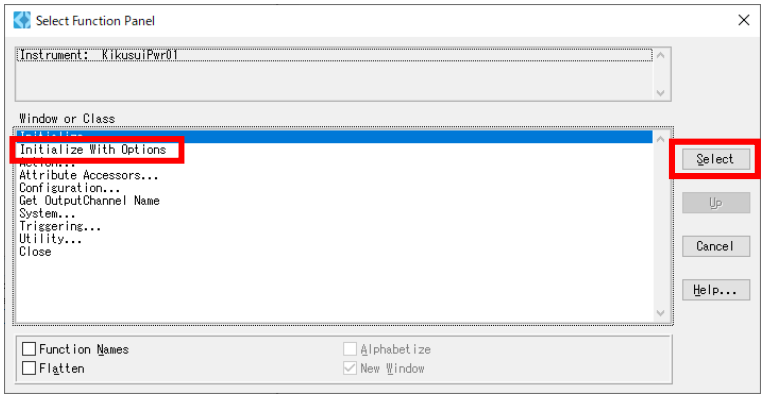
Setting the parameters

1 **Open the source code of C (Ex01.c) added to the project.**

2 **Click [Instrument] > [KikusuiPwr01].**

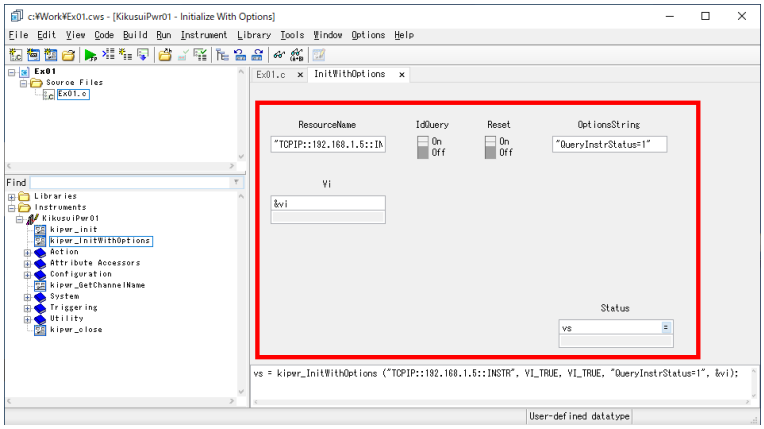
Select Function Panel will be displayed.

3 **Click [Initialize With Options] and click [Select].**



The function panel of Initialize With Options will be displayed.

4 **Set the values for each parameter.**



This section explains taking the case in which the instrument is connected by the LAN interface and the IP address is 192.168.1.5 as an example. As the parameters of ResourceName and OptionString are character strings, enclose them in quotation marks.

Parameters	Value
ResourceName	“TCPIP::192.168.1.5::INSTR”
IdQuery	On (VI_TRUE in the function call description)
Reset	On (VI_TRUE in the function call description)
OptionString	“QueryInstrStatus=1”
Vi	&vi
Status	vs

Declaring variable

Declare the variable entered in Vi and Status.

1 **Click the Parameter Vi on the function panel.**

2 **Click [Code] > [Declare Variable].**

Declare Variable dialog box will be displayed.

3 **Check both the Execute declaration in Interactive Window and the Add declaration to top of target file “Ex01.c”.**

4 **Click [OK].**

5 **Click the Parameter Status on the function panel.**

6 **Operate in the same way as steps 2 to 4.**

This completes the declaration.

Inserting the code to recall the Initialize With Options function

1 **Click [Code] > [Insert Function Call].**

The code to recall the kipwr_InitWithOptions function will be inserted to the source code (Ex01.c).

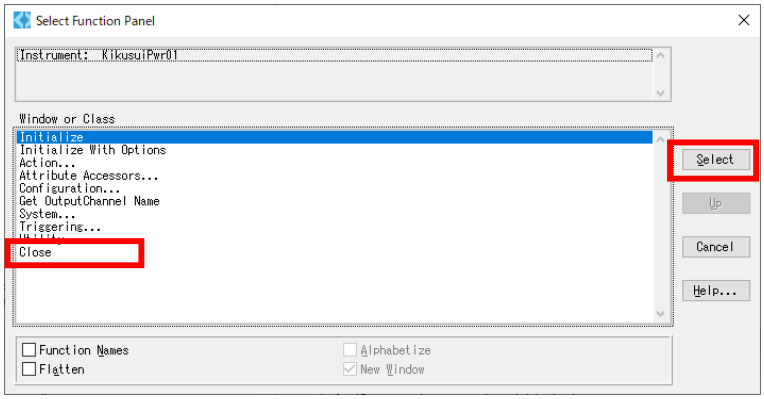
Setting the Close function

Set the Close function in the same way as the Initialize With Options function, and insert them into the source code.

1 **Click [Instrument] > [KikusuiPwr01].**

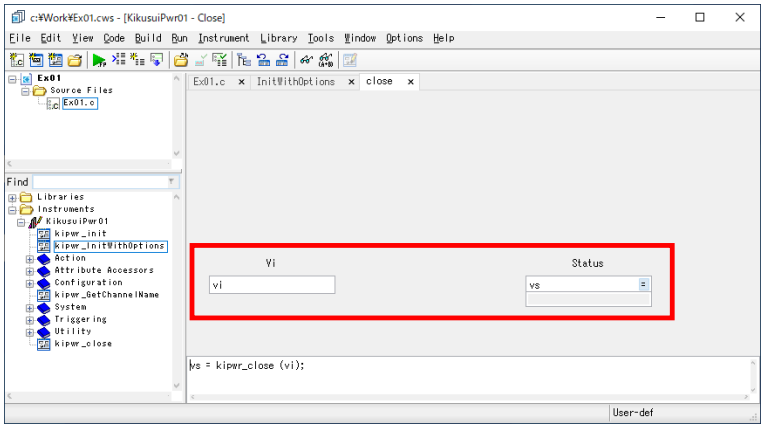
Select Function Panel will be displayed.

2 **Click [Close] and then [Select].**



The function panel of Close will be displayed.

3 **Enter vi in the parameter [Vi], and enter vs in [Status].**



4 **Click [Code] > [Insert Function Call].**

The code to recall the kipwr_close function will be inserted to the source code (Ex01.c).

Adding other functions

By selecting [Ex01.c] tab, the source code will be displayed. By executing the steps in pages 7 to 9, the source code will be as follows:

```
static ViStatus vs;
static ViSession vi;
vs = kipwr_InitWithOptions ("TCPIP::192.168.1.5::INSTR", VI_TRUE, VI_
     TRUE, "QueryInstrStatus=1", &vi);
vs = kipwr_close (vi);
```

In order to make this source code an executable program, include the following content:

- Specify loading of the included file of the instrument driver.
- Enclose the recall to InitWithOptions and close in the “main” function.
- Between the recalls to InitWithOptions and close, add a recall code for the function setting the voltage and current, and the function controlling the output ON/OFF. Besides the method of inserting it using the function panel, there is also the method of entering the source code directly with the editor as shown below.

Shown below is a sample that opens an instrument driver session, sets voltage, current, and output, and closes immediately.

```
#include <kipwr.h>
static ViStatus vs;
static ViSession vi;
void main()
{
vs = kipwr_InitWithOptions ("TCPIP::192.168.1.5::INSTR", VI_TRUE, VI_
     TRUE, "QueryInstrStatus=1", &vi);
vs = kipwr_ConfigureVoltageLevel (vi, "", 20);
vs = kipwr_ConfigureCurrentLimit (vi, "", KIPWR_VAL_CURRENT_REGULATE,
     2.0);
vs = kipwr_ConfigureOutputEnabled (vi, "", 1);
vs = kipwr_close (vi);
}
```

Memo

IVI instrument drivers for DC power supply units are usually structured in consideration of multiple channels. Therefore, many of the device setting functions have the ChannelName parameter.

Building projects

1. **Click [Build] > [Build].**

   This completes building.

Executing Programs

This section introduces the breakpoints that are useful for debugging at the time of program execution and explains how to run the program and the values stored in variables after execution.

Setting the breakpoints

It is not clear how to execute the program of the sample on page 10 because it is not interactive. Inserting a breakpoint in a recall line pauses the program before it is recalled so that you can check the status of the program in progress.

1 **Move the cursor to the row where you want to insert the breakpoint.**

2 **Click [Run] > [Toggle Breakpoint].**

Breakpoints will be set.

By executing a program

1 **Click [Run] > [Debug Ex01.exe].**

The program will be executed. Pause before calling the row where the breakpoint has been set.

2 **Click [Run] > [Continue].**

Program execution will be restarted.

Values to be stored in variables vi and vs

In the case of being able to open a session for an instrument driver, vi will store a session handle (usually 0x00000001 or more as an IVI handle). The execution result (0x00000000 on success or a negative value on failure) will be stored in vs.

Explanation of functions

This section describes the functions comprising the program, including their contents and setting values.

Memo

The IVI-C instrument driver and VXI Plug&Play instrument driver have the notation <prefix> as a rule for function names. This notation represents the identification name of each instrument driver.

For example, the notation “<prefix>_init()” is “kipwr_init()” in the kipwr instrument driver.

Starting sessions

To start a session on a specific interface, use <prefix>_InitWithOptions.

```
vs = kipwr_InitWithOptions (“TCPIP::192.168.1.5::INSTR”, VI_TRUE, VI_
     TRUE, “QueryInstrStatus=1”, &vi);
```

The parameters that can be set in <prefix>_InitWithOptions are as follows:

Parameters	Type	Description
ResourceName	ViRsrc (const char*)	This is a character string of VISA resource names determined by the I/O interface, address, etc. to which the instrument is connected. For example, when an instrument with an IP address of 192.168.1.5 is connected to a LAN and controlled by a VXI-11 interface, it will be “TCPIP::192.168.1.5::INSTR”.
idQuery	ViBoolean	In the case that VI_TRUE is specified, an ID query such as “*IDN?” is issued in response to the instrument to make queries for model information.
Reset	ViBoolean	In the case that VI_TRUE is specified, the “*RST” command is issued to reset the settings of the instrument.
OptionString	ViConstString (const char*)	The settings of the IVI definition can be changed. (p.13)
Vi	ViSession*	Receive an instrument session (pointers passed).

Setting OptionString

In OptionString, the following IVI definitions can be set up:

IVI definition	Default value
RangeCheck	VI_TRUE
Cache	VI_TRUE
Simulate	VI_FALSE
QueryInstrStatus	VI_FALSE
RecordCoercions	VI_FALSE
Interchange Check	VI_FALSE

OptionString is a parameter of the character string. The following is a sample:

```
QueryInstrStatus = VI_TRUE, Cache = VI_TRUE, DriverSetup=12345
```

Regarding the formatting, note the following:

- If the set value is not specified, the default value will be applied.
- The set value is of the ViBoolean type. Either of “VI_TRUE”, “VI_FALSE”, “1” or “0” can be set.
- Function names and values are case-insensitive. Uppercase and lowercase letters are not distinguished.
- When setting multiple items, they should be separated by commas.
- Depending on the instrument driver, the “DriverSetup” parameters are supported.
  The “DriverSetup” is a parameter that specifies items not defined in the IVI specifications when recalling InitWithOptions. Usage and formatting are dependent on the driver. Therefore, in the case of setting the “DriverSetup”, it should be specified as the last item in the “Option String”. For the details of what is specified in “DriverSetup”, see the Readme of the driver or the online help.

Setting the channel name

In the case of power supplies and oscilloscopes, the IVI instrument drivers are designed with the assumption that they are equipped with multiple channels. Therefore, many of the driver functions that work on the instrument panel settings need to specify a channel for the second parameter, as shown below:

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

In this example, a blank ("") is specified as the channel name. If the number of channels is one, a blank is fine, but if there are multiple channels, it needs to be specified explicitly.

The names of the channels that can be used depend on the instrument driver. For details, see the help for each driver.

Closing sessions

To close the instrument driver sessions, use <prefix>_close.

```
vs = kipwr_close (vi);
```

Handling errors

If a value out of range is passed to a parameter or an unsupported function is recalled, an error can occur in the instrument driver. In the IVI-C instrument driver, all errors that have occurred in the instrument driver are conveyed to the client program as a return value of ViStatus-type, as shown below.

Range of values	Description
vs = 0	Success
vs > 0	Warning
vs < 0	Error

Although no error correction has been done in the previous examples, the error_message function can be used to convert a ViStatus-type return value into readable messages. The error_message exceptionally accepts VI_NULL as ViSession. The receiving buffer size should be 256 bytes or more.

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

Programming using a class interface

This section describes the procedures for programming using a class interface. In the class interface, you can achieve interchangeability with an instrument class interface by programming using the instrument class interface defined in the IVI specifications. Using interchangeability, the instruments can be replaced without recompiling and linking the application once again.

Memo

- To utilize interchangeability, an IVI-C instrument driver of the same class of instrument must be provided in response to both models before and after replacement. Interchangeability between different classes of instrument is not available.
- In programming using the class interface, the machine-specific functions that are available are limited. To maximize the use of model-specific functions, program using the specific interface (p.5).

Preparation for programming

Creating virtual instruments

To create an application that uses the interchangeability functions, it is necessary to create a virtual instrument in advance.

Memo

To avoid loss of interchangeability function, do not include descriptions dependent on specific IVI-C instrument drivers (e.g. direct calls to kipwr_init function) or specific VISA addresses (resource names) (e.g. “TCPIP::192.168.1.5:: INSTR”) in the application code.

In the IVI specifications, the interchangeability function is achieved by placing the IVI configuration store outside the instrument drivers and applications. The application is not controlled by using model-specific instrument drivers directly, but through a special instrument driver, called the instrument class interface.

At the time of control, select the instrument driver DLLs according to the contents of the IVI configuration store and access the instrument driver loaded indirectly through a model-independent class interface function.

An XML file (C:/ProgramData/IVI Foundation/IVI/IviConfigurationStore.xml) will be used for the IVI configuration store. Mainly IVI instrument drivers and some VISA/IVI configuration tools access through the IVI Configuration Server DLL. Applications do not usually use it. When using LabWindows/CVI, use the NI-MAX (NI Measurement and Automation Explorer) software manufactured by National Instruments to configure the IVI driver.

This section describes the procedure of creating a virtual instrument using NI-MAX.

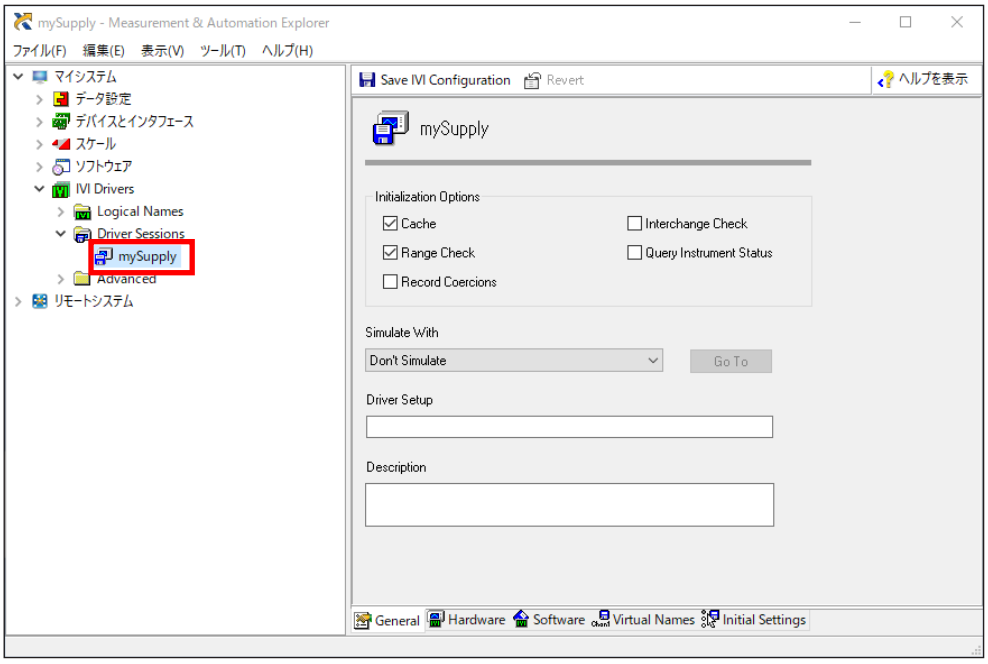
Creating Driver Sessions

1 **Launch NI-MAX and check the hierarchy under the [IVI Drivers] in the tree display on the left of the screen.**

2 **Right-click [Driver Sessions] and select [Create New (case-sensitive)].**

3 **Set a name for Driver Sessions.**

Here, set it as [mySupply].

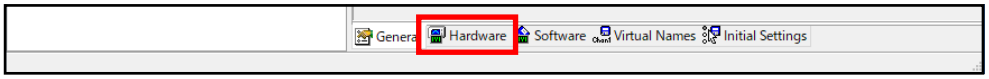


Driver Sessions have now been created.

Creating Hardware Asset

In Hardware Asset, specify with what path the instruments to be used will be connected.

1 **Select the [Hardware] tab.**



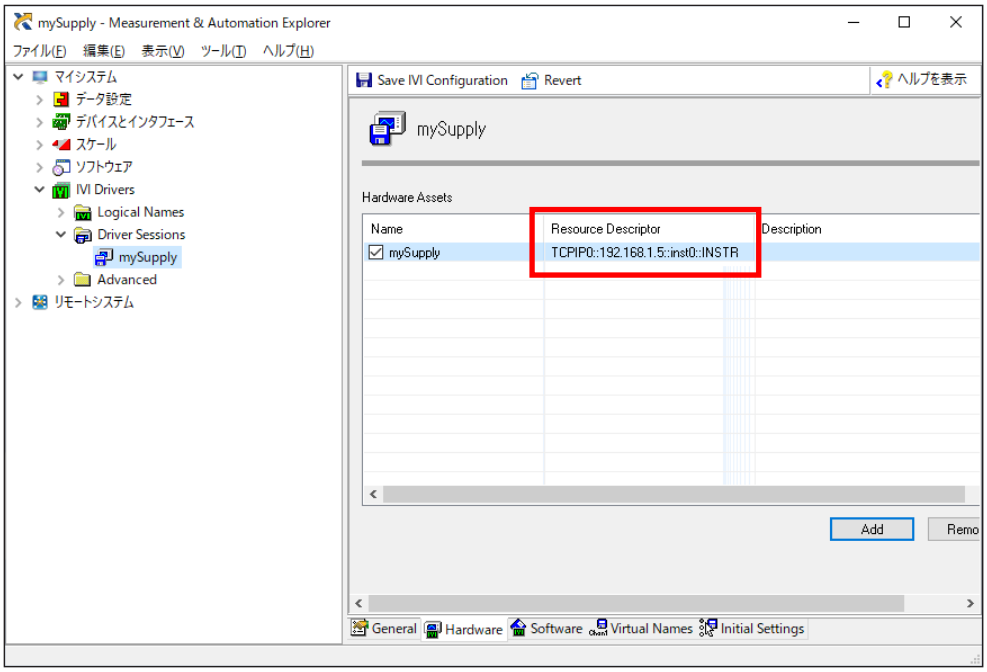
2 **Click [Add] to create a new [Hardware Asset].**

3 **Set a name for Hardware Asset.**

Here, set it as [mySupply].

4 **At [Resource Descriptor], specify the VISA address to which the instrument is connected.**

In this example, [TCPIP::192.168.1.5::inst0::INSTR] is specified.

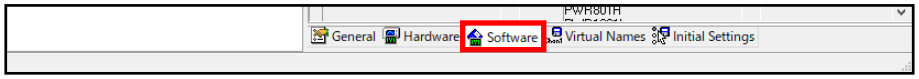


Hardware Asset has now been created.

Setting Software Module

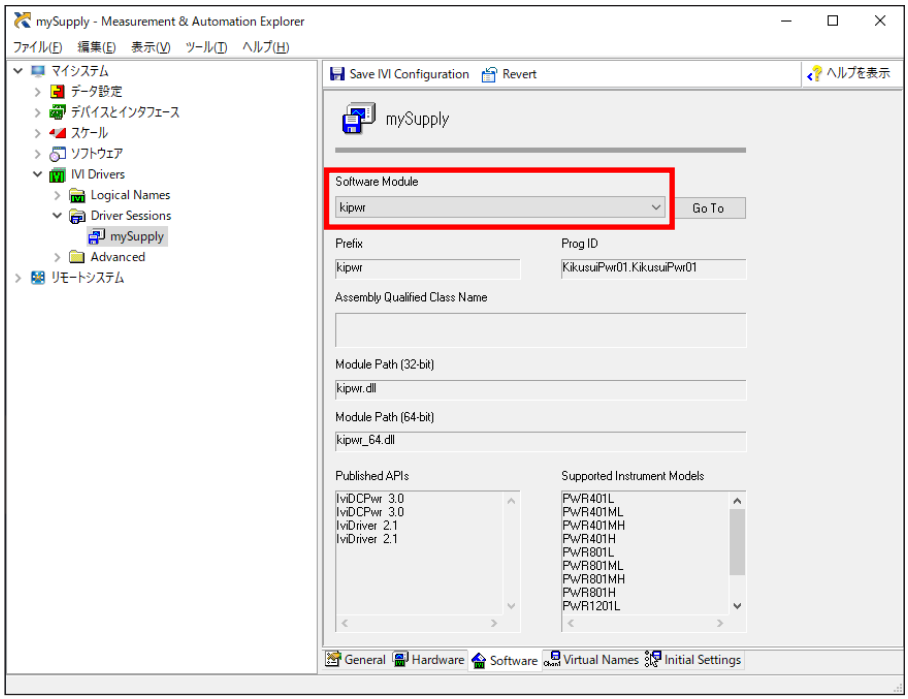
In the Software Module, the instrument driver module (DLL module) is set.

1 **Select the [Software] tab.**



2 **From the [Software Module] list, select the instrument driver module to be used.**

In this example, [kipwr] is selected.

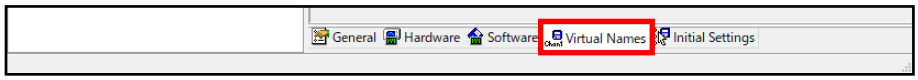


Software Module has now been set.

Creating Virtual Name

With Virtual Name, a virtualized name is created for the channel name of the instrument driver. This is because different instrument drivers have different valid channel names in the case of instrument drivers requiring channel specification.

1 **Select the [Virtual Names] tab.**



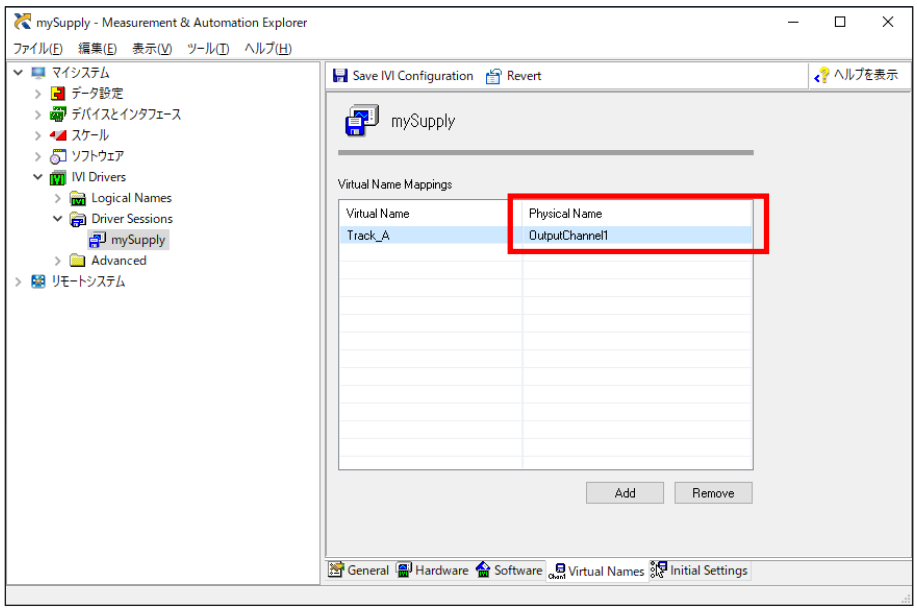
2 **Click [Add] to add a virtual name and enter [Track_A] in [Virtual Name].**

On the [Physical Name] list, the channel on which the instrument operates will be displayed.

3 **In the [Physical Name] column, select a channel name displayed on the list or enter a valid channel name.**

In this example, select or enter [OutputChannel1].

> **Memo**
>
> In some cases, not all channel names may be displayed, depending on the driver's mounting conditions or the configuration of the multi-channel power supply unit. For details on the names of the available channels for the driver, see the Readme document for each driver or the online help.



Virtual Name has now been created.

Setting Logical Name

A Logical Name refers to the name of a virtual instrument that is configured in NI-MAX.

1 **Check the hierarchy under [IVI Drivers] in the tree display on the left side of the screen.**

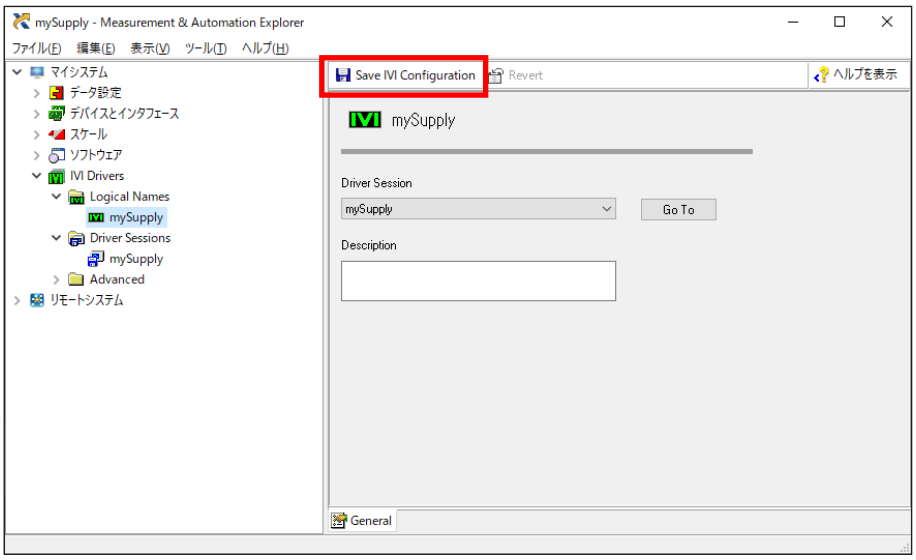
2 **Right-click [Logical Name] and select [Create New (case-sensitive)].**

3 **Set a name for Logical Name.**

Here, set it as [mySupply].

4 **In [Driver Session], select [mySupply].**

5 **Click [Save IVI Configuration] on the toolbar and save the settings.**



Logical Name has now been set.

This completes the creation of the virtual instrument.

Creating a new project

An example of using the IVI-C driver in a new application is explained below:

1 **Start LabWindows/CVI.**

Welcome page is displayed.

2 **Click [New] > [Project].**

If the welcome page is not displayed and the existing projects are imported, click [File] > [New] > [Project (*.prj)].

A new project will be created.

3 **Click [File] > [Save Untitled.prj As] to save the project.**

In this case, save it as “Ex02.prj”.

4 **Click [File] > [New] > [Source (*.c)].**

A source file is created.

5 **Click [File] > [Save Untitled1.c As] to save the source file.**

In this case, save it as “Ex02.c”.

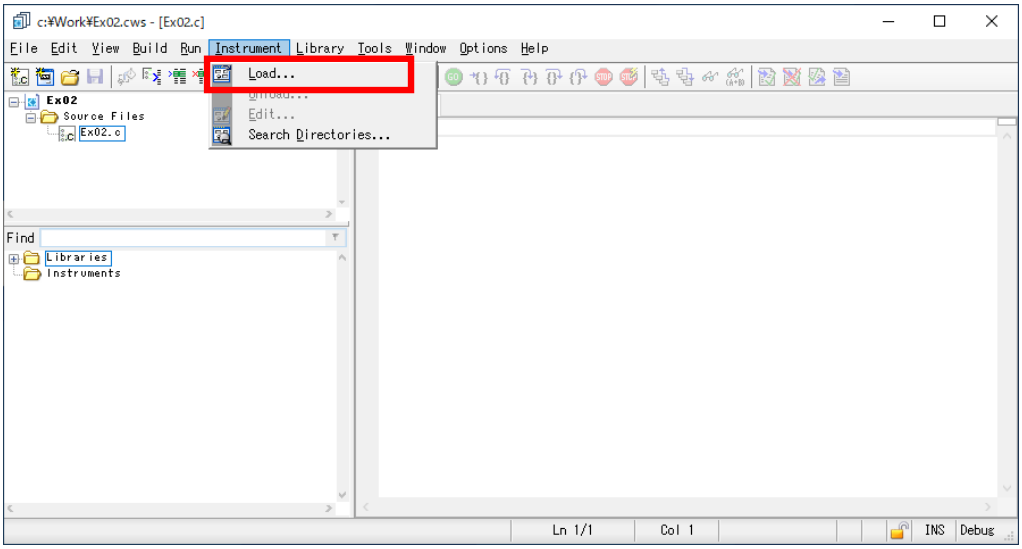
6 **Click [File] > [Add Ex02.c to Project].**

The source file is added to the project.

This completes the creation of a new project.

Loading an instrument driver

1 **Click [Instrument] > [Load].**



2 **Load the "IviDCPwr.fp" saved in "C:/Program Files (x86)/IVI Foundation/IVI/Drivers/IviDCPwr" directory.**

In the [Instrument] menu, [IviDCPwr Class Driver] will be added.

Configuring the program

Set the Initialize With Options function, declare the variables and insert them into the source code.

Setting the Initialize With Options function

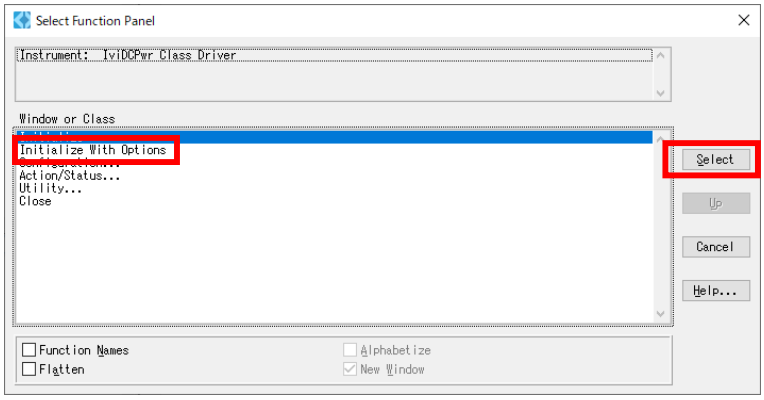
Setting the parameters

1 **Open the source code of C (Ex02.c) added to the project.**

2 **Click [Instrument] > [IviDCPwr Class Driver].**

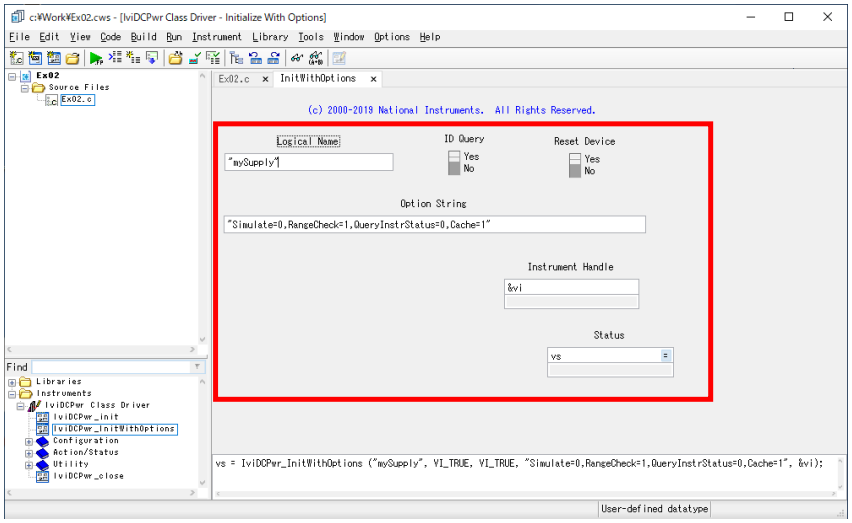
Select Function Panel will be displayed.

3 **Click [Initialize With Options] and click [Select].**



The function panel of Initialize With Options will be displayed.

4 **Set the values for each parameter.**



As the parameters of Logical Name and OptionString are character strings, enclose them in quotation marks.

Parameters	Value
Logical Name	“mySupply”
ID Query	Yes (VI_TRUE in the function call description)
Reset Device	Yes (VI_TRUE in the function call description)
Option String	Leave as default.
Instrument Handle	&vi
Status	vs

Declaring variable

Declare the variables entered in [Instrument Handle] and [Status].

1 **Click the parameter Instrument Handle in the function panel.**

2 **Click [Code] > [Declare Variable].**

Declare Variable dialog box will be displayed.

3 **Check both the Execute declaration in Interactive Window and the Add declaration to top of target file “Ex02.c”.**

4 **Click [OK].**

5 **Click the Parameter Status on the function panel.**

6 **Operate in the same way as steps 2 to 4.**

This completes the declaration.

Inserting the code to recall the Initialize With Options function

1 **Click [Code] > [Insert Function Call].**

The code to recall the IviDCPwr_InitWithOptions function will be inserted to the source code (Ex02.c).

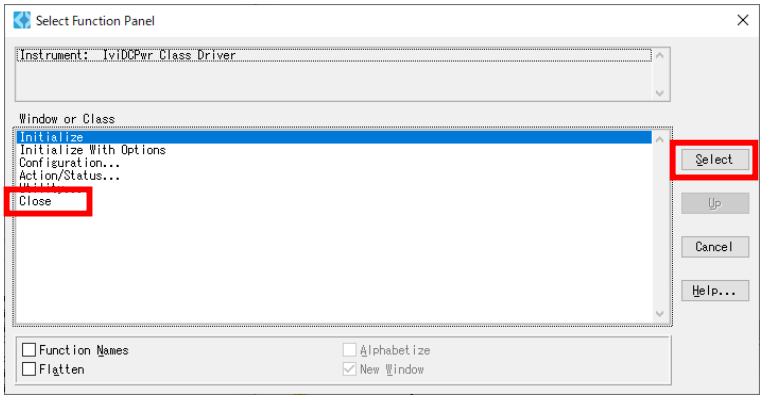
Setting the Close function

Set the Close function in the same way as the Initialize With Options function, and insert them into the source code.

1 **Click [Instrument] > [IviDCPwr Class Driver].**

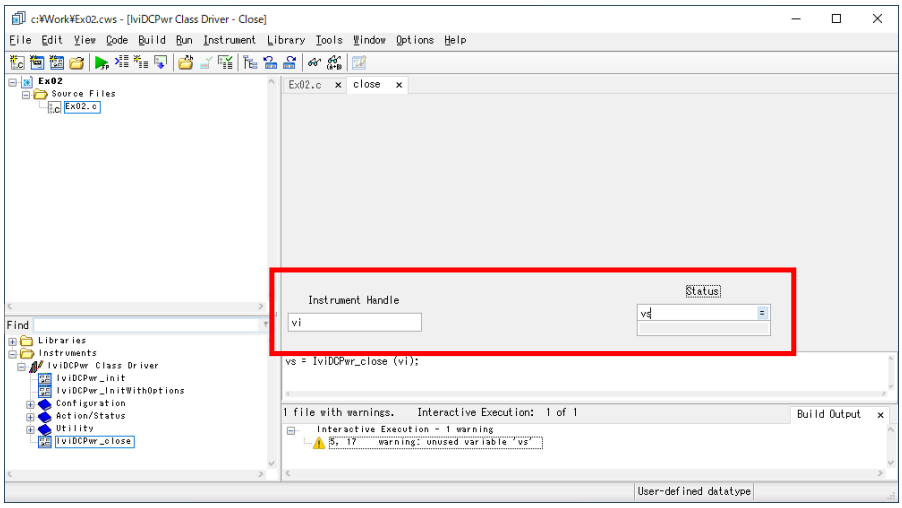
Select Function Panel will be displayed.

2 **Click [Close] and then [Select].**



The function panel of Close will be displayed.

3 **Enter vi in the parameter [Instrument Handle], and enter vs in [Status].**



4 **Click [Code] > [Insert Function Call].**

The code to recall the IviDCPwr_close function will be inserted to the source code (Ex02.c).

Adding other functions

By selecting [Ex02.c] tab, the source code will be displayed. By executing the steps in pages 25 to 27, the source code will be as follows:

```
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);
```

In order to make this source code an executable program, include the following content:

- Specify loading of the included file of the instrument driver.
- Set the initial values of the variables vi and vs at 0.
- Enclose the recall to InitWithOptions and close in the “main” function.
- Between the recalls to InitWithOptions and close, add a recall code for the function setting the voltage and current, and the function controlling the output ON/OFF. Besides the method of inserting it using the function panel, there is also the method of entering the source code directly with the editor as shown below.

Shown below is a sample that opens an instrument driver session, sets voltage, current, and output, and closes immediately.

```
#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 projects

1 **Click [Build] > [Build].**

This completes building.

Executing Programs

This section introduces the breakpoints that are useful for debugging at the time of program execution and explains how to run the program and the values stored in variables after execution.

Setting the breakpoints

It is not clear how to execute the program of the sample on page 28 because it is not interactive. Inserting a breakpoint in a recall line pauses the program before it is recalled so that you can check the status of the program in progress.

1 **Move the cursor to the row where you want to insert the breakpoint.**

2 **Click [Run] > [Toggle Breakpoint].**

Breakpoints will be set.

By executing a program

3 **Click [Run] > [Debug Ex02.exe].**

The program runs and pauses before calling the row where the breakpoint has been set.

4 **Click [Run] > [Continue].**

Program execution will be restarted.

Values to be stored in variables vi and vs

In the case of being able to open a session for an instrument driver, vi will store a session handle (usually 0x00000001 or more as an IVI handle). The execution result (0x00000000 on success or a negative value on failure) will be stored in vs.

Explanation of functions

Functions that comprise the program are described with a case in which the IviDCPwr class driver is used as an example. The prefix “IviDCPwr” attached to the VI (function) is unique to the IviDCPwr class driver.

Starting sessions

To start a session of the class interface, use the IviDCPwr_InitWithOptions.

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

The parameters that can be set with IviDCPwr_InitWithOptions are as follows:

Parameters	Description
Logical Name	Specify the logical name of the virtual instrument created with NI-MAX. The class driver locates the appropriate instrument driver DLL (Software Module) or VISA address (Hardware Asset) from the logical name and recalls kipwr_InitWithOption indirectly at the end. The class driver cannot provide a VISA address directly to the IviDCPwr_InitWithOptions function.
ID Query	In the case that VI_TRUE is specified, an ID query such as “*IDN?” is issued in response to the instrument to make queries for model information.
Reset Device	In the case that VI_TRUE is specified, the “*RST” command is issued to reset the settings of the instrument.
Option String	The settings of the IVI definition can be changed. (p.13)
Vi	Receive an instrument session (pointers passed).

Setting OptionString

In OptionString, the following IVI definitions can be set up:

- RangeCheck
- Cache
- Simulate
- QueryInstrStatus
- RecordCoercions
- Interchange Check

OptionString is a parameter of the character string. The following is a sample:

```
QueryInstrStatus = VI_TRUE, Cache = VI_TRUE, DriverSetup=12345
```

Regarding the formatting, note the following:

- If the set value is not specified, the default value will be applied. The default value is the value specified on the [Driver Session] > [General] page of the IVI configuration.
- The set value is of the ViBoolean type. Either of “VI_TRUE”, “VI_FALSE”, “1” or “0” can be set.
- Function names and values are case-insensitive. Uppercase and lowercase letters are not distinguished.
- When setting multiple items, they should be separated by commas.
- Depending on the instrument driver, the “DriverSetup” parameters are supported. The “DriverSetup” is a parameter that specifies items not defined in the IVI specifications when recalling InitWithOptions. Usage and formatting are dependent on the driver. Therefore, in the case of setting the “DriverSetup”, it should be specified as the last item in the “Option String”. For the details of what is specified in “DriverSetup”, see the Readme of the driver or the online help.

Setting the channel name

In the case of power supplies and oscilloscopes, the IVI instrument drivers are designed with the assumption that they are equipped with multiple channels. Therefore, many of the driver functions that work on the instrument panel settings need to specify a channel for the second parameter, as shown below:

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

In this example, “Track_A” is specified as the channel name. Track_A is the virtual name that was set in IVI Configuration (p.17). By specifying a virtual name, programming can be done independently of the instrument driver of a particular model.

In the IVI configuration, the virtual name “Track_A” has been configured so that it can be converted to the channel name that can be used only by a specific instrument driver (in this case, the kipwr driver) called “OutputChannel1”. Therefore, if the instrument is replaced, its operation can be continued by simply changing the IVI configuration settings without having to rewrite the program (p.34).

Instruments can be controlled by directly specifying channel names that are dependent on the instrument driver, but interchangeability will be lost. For example, since the valid channel name for the AgN57xx instrument driver is “Output1”, if the channel name is specified as “OutputChannel1”, the program must be rewritten in order to replace the instrument with the AgN57xx.

Memo

- Do not manually edit the XML file (IviConfigurationStore.xml) in which the IVI configuration information is saved.
- The IVI configuration will be shared by all 32-bit/64-bit measurement applications and all logged-on users in the same PC.

Closing sessions

To close the instrument driver session, use IviDCPwr_close.

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

Handling errors

If a value out of range is passed to a parameter or an unsupported function is recalled, an error can occur in the instrument driver. In the IVI-C instrument driver, all errors that have occurred in the instrument driver are conveyed to the client program as a return value of ViStatus-type, as shown below.

Range of values	Description
vs = 0	Success
vs > 0	Warning
vs < 0	Error

Replacing instrument

When replacing the instrument, the operation can be continued by simply changing the Driver Session of the virtual instrument (IVI configuration). There is no need to change the application itself.

There are three Driver Session settings to be changed.

Item	Settings
[Hardware] tab > [Hardware Assets] > [Resource Descriptor]	VISA address to which the instrument is connected
[Software] tab > [Software Module]	Instrument drivers to be used
[Virtual Names] tab > [Physical Names]	The physical name to which the virtual channel name is mapped

If properly configured for the replaced instrument, the application can be operated without having to recompile and link it again.

In the examples in this guide, if the instrument is replaced from a Kikusui PWR-01 series DC power supply (instrument hosted by the kipwr instrument driver) to an Agilent N5700 series DC power supply (instrument hosted by the AgN57xx driver), the [mySupply] settings should be changed as follows:

Item	Description of change
[Hardware] tab > [Hardware Assets] > [Resource Descriptor]	VISA address to which the Kikusui PWR-01 series DC power supply is connected => VISA address to which Agilent N5700 series DC power supply is connected
[Software] tab > [Software Module]	“kipwr” => “AgN57xx”
[Virtual Names] tab > [Physical Names]	“OutputChannel1” => “Output1”

Memo

The interchangeability function using IVI class drivers does not guarantee the operation before and after replacement of the instrument. Be sure to fully verify that the replaced system is functioning properly before operating it.