IVI Instrument Drivers Programming Guide Excel VBA Edition

April 2012 Revision 2.2

Contents

Introduction

This guide shows examples of using the KikusuiPwr IVI Instrument Driver (for the Kikusui PWR-01 series DC power supply). You can use other IVI instrument drivers of different manufacturers or models by following almost the same instructions as those described here.

This guide uses Microsoft Excel for Microsoft 365 MSO (16.0.13801.20288) 64-bit Ver. 2102 to show the examples of creating 64-bit programs that run on 64-bit Windows 10.

Instrument drivers used with Excel VBA

Excel VBA provides one of the development environments suitable for use with IVI-COM instrument drivers. Although using COM objects, such as ActiveX controls, is a generally adopted technique in Excel VBA programming, you can create programs using IVI-COM instrument drivers in almost the same instructions as those when creating them using COM objects.

The guide shows examples of how to create programs using IVI-COM instrument drivers.

Supported interfaces

IVI instrument drivers support the following two types of interfaces:

- **Specific interface**
  Interface specific to an individual driver. With the interfaces of this type, you can make the most of the functions of the instruments you will use.
- **Class interface**
  Interface that supports an instrument class defined by the IVI specifications.
  The interfaces of this type enable interchangeability; however, the use of instrument-specific functions is limited.

This guide explains how to create programs using these interfaces.

Memo

- Information about the instrument classes to which individual instrument drivers belong are contained in the Readme file (Readme.txt) for the driver.
  To view the Readme file, click **Start** > **Kikusui** > **KikusuiPwr IVI Driver 1.0.0 Documentation**.
- If the instrument driver does not belong to any instrument class, you cannot use the class interface to create applications that use interchangeability.

Software you need to install before programming

Before programming, download and install the following software from the Download Service on our website (https://www.kikusui.co.jp/download/).

- KI-VISA Ver5.5 or later
- PWR-01 series IVI-COM Multi-Environment Drivers Ver1.0 or later

For details, see “IVI Instrument Drivers Programming Guide — Setup”.

If you wish to use a class interface, you also need to install the following software:

- NI-MAX Ver20.5.0 or later
- IVI compliance Package Ver20.0.0 or later

Preparation for Using VBA

The following shows an example of creating an application for directly adding a button in an Excel sheet. To create such an application, you need to enable macros in Excel to create a new macro.

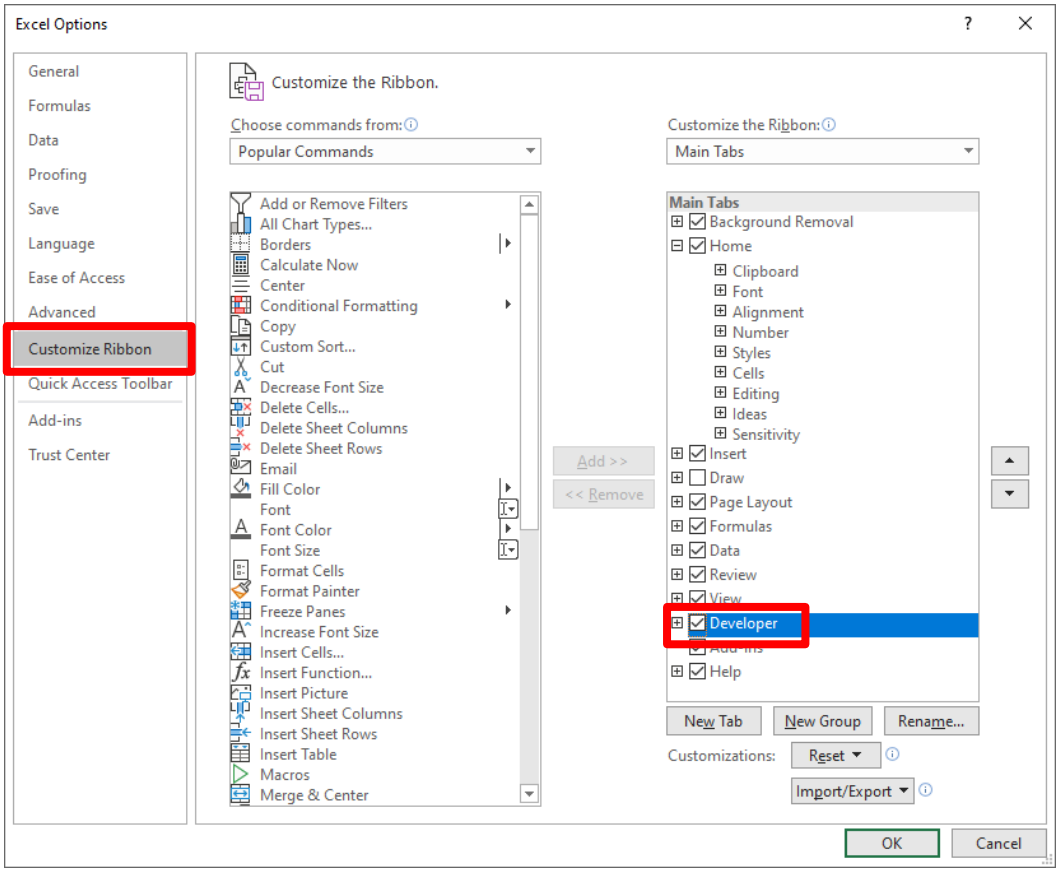
Enabling macros in Excel

To use VBA, enable macros in Excel.

1 **Launch Excel, and then click the File tab.**

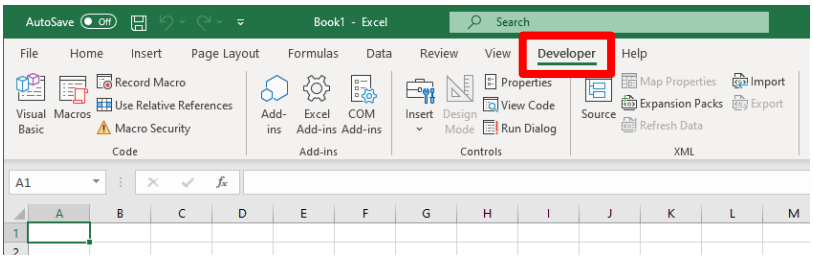
2 **Click Options at the lower left of the window.**

3 **Select Customize Ribbon > Developer.**



You will see the **Developer** tab in your Excel ribbon.

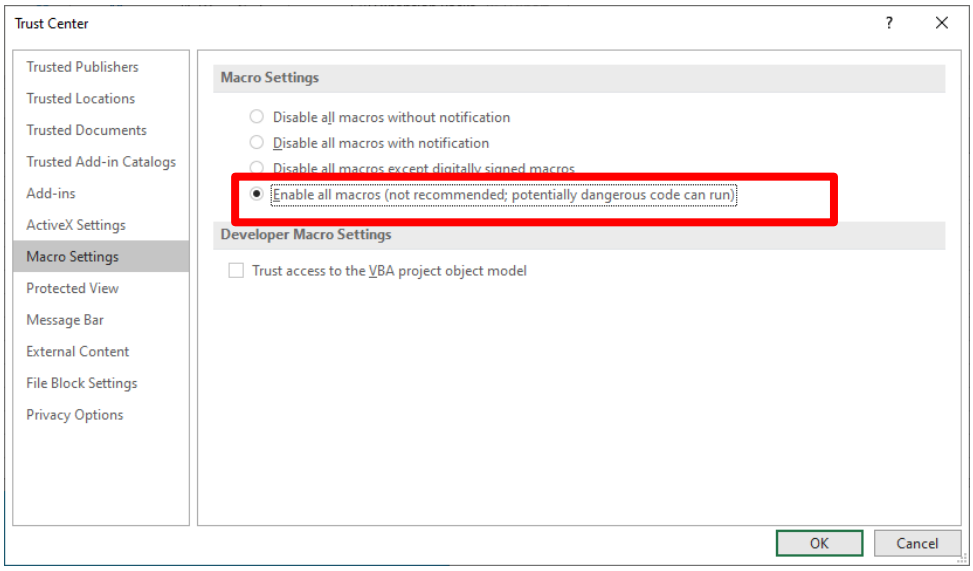
4 Click the Developer tab in the Excel ribbon.



5 In the Code group, click Macro Security.

The Trust Center dialog box appears.

6 Select Macro Settings > Enable all macros.



7 Click OK.

Creating a new macro

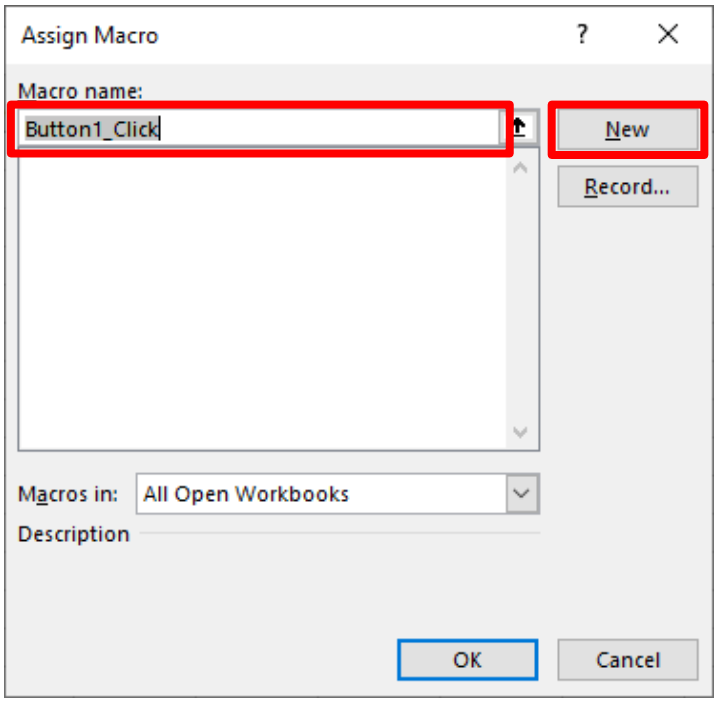
Create a button in Excel and register a new macro.

1 **On the Developer tab, click Insert > Form Controls, and then select a Button (in Form Controls).**

2 **Click the cell in which you want to add the button.**

The Record Macro dialog box appears.

3 **Enter a name for the macro, and then click Create.**



The Visual Basic editor appears. The editor shows the source code for the button handler. From here, you can write a program in the button handler.

Programming using a specific interface

With the specific interface, you can make the most of the instrument driver-specific functions.

The following explains how to create a program using a specific interface.

Memo

Specific interfaces do not support interchangeability. If you which to take advantage of interchangeability, use a class interface (p.7).

Preparation for programming

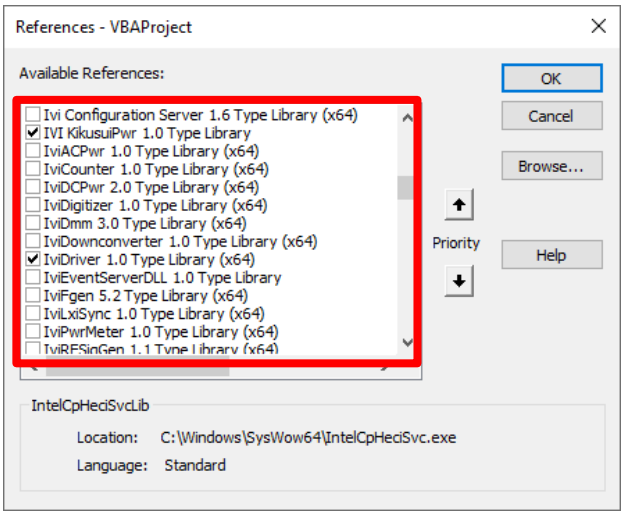
Importing a Type Library

Import a type library for the IVI-COM instrument driver you will use.

1 **Open the Visual Basic editor, and then click Tools > References.**

The References dialog box appears.

2 **Select IVI KikusuiPwr 1.0 Type Library and IviDriver 1.0 Type Library.**



3 **Click OK.**

Building a program

Creating an object and opening/closing a session

The following is a programming code example, which contains the code for creating an instrument driver object and opening and closing the session in addition to the button handler source code you opened in “Creating a new macro” (p.7). Here, it is assumed that the instrument (Kikusui PWR-01 series DC power supply) with IP address of 192.168.1.5 is connected via LAN.

```
Sub CommandButton1_Click()
      Dim inst As IKikusuiPwr01
      Set inst = New KikusuiPwr01

      inst.Initialize "TCPIP::192.168.1.5::INSTR", True, True, ""

      inst.Close
End Sub
```

The above code can run as an independent program. To execute the program for validation, see “Executing a program” (p.14).

Parameters for Initialize method

All IVI-COM instrument drivers include the Initialize method defined by the IVI specifications. The Initialize method has the following parameters:

Parameters	Type	Description
ResourceName	String	Character string of the VISA resource name determined by the I/O interface or address 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, the parameter will be [TCPIP::192.168.1.5::INSTR].
IdQuery	Boolean	If TRUE, this method issues a query command, such as the *IDN? query, to ask the instrument model information.
Reset	Boolean	If TRUE, this method issues a restore command, such as the *RST command, to reset the instrument settings.
OptionString	String	Allows you to change the settings of the IVI definition.

How to set OptionString

OptionString of the initialize method can specify the following IVI definitions:

IVI definition	Default value
RangeCheck	TRUE
Cache	TRUE
Simulate	FALSE
QueryInstrStatus	FALSE
RecordCoercions	FALSE
Interchange Check	FALSE

OptionString is character string parameters. The following shows an example.

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

When formatting, be sure to note the following:

- If the set value is not specified, the default value will be applied.
- The set values are Boolean datatype. You can select a set value from the following: [TRUE], [FALSE], [1], [0].
- Function names and set values are case-insensitive. Uppercase and lowercase letters are not distinguished.
- When you set multiple items, use commas to separate them.
- Depending on the instrument driver, a “DriverSetup” parameters are supported.
  The “DriverSetup” parameters are used to specify an item not defined by the IVI specifications when the Initialize method is invoked. Its usage and format depend on the driver. Therefore, when you set a “DriverSetup” parameter, specify it as an item at the end of the ”OptionString”. For details about the “DriverSetup” parameters, see the Readme file or the online help for the driver.

Setting channels and testing conditions

IVI instrument drivers for power supplies are designed based on the premise that the IVI instrument drivers have multiple channels. Therefore, in many cases, properties or methods that control instrument panel settings implement the concept of an array of objects called Repeated Capabilities (or Collection in general computing terms for COM) in the IVI specifications. For example, instrument drivers for DC power supplies implement the Output Collection.

The KikusuiPwr IVI-COM driver implements IKikusuiPwr01Outputs and IKikusuiPwr01Output. A collection is a group of multiple objects. Therefore, a collection contains more than one object. Typically, objects and their collection have the same name. The plural object name represents a collection, while a singular object name represents an object.

The following shows a programming code example, which includes the code for controlling the output channel, “OutputChannel1”, of the Kikusui PWR-01 series DC supply in addition to the code mentioned in “Building a program” (p.9).

```
Sub CommandButton1_Click()
      Dim inst As IKikusuiPwr01
      Set inst = New KikusuiPwr01

      inst.Initialize "TCPIP::192.168.1.5::INSTR", True, True, ""

      Dim output As IKikusuiPwr01Output
      Set output = inst.Outputs.Item("OutputChannel1")
      output.VoltageLevel = 20.0
      output.CurrentLimit = 2.0
      output.Enabled = True
      inst.Close
End Sub
```

Explanation of the programming code example

In the above code, IKikusuiPwr01Outputs is obtained through the Outputs property for the IKikusuiPwr01 interface. Also, the IKikuauiPwr01Output interface is obtained using the Item property.

The VoltageLevel property specifies a voltage level. The CurrentLimit property specifies a limit value for the current. The Enabled property sets the ON/OFF state of the output.

A single Output object name ("OutputChannel1" in the example code) to be referenced is specified for the parameter to be passed to the Item property. The object name varies by instrument driver. You can usually find the list of object names available for a specific instrument driver in the driver’s Help file. Alternatively, you can check an object name by writing the code mentioned below.

Checking an object name specific to an instrument driver

If you are uncertain about the object name specific to the instrument you will use, you can check it using Count or Name (both are read-only) in the Output collection.

The following shows a programming code example.

```
Dim outputs As IKikusuiPwr01Outputs
Set outputs = inst.outputs

Dim n As Integer
Dim c As Integer
c = outputs.Count

For n = 1 To c
     Dim name As String
     name = outputs.name(n)
     Debug.Print name
Next
```

The Count property returns the number of single objects containing in the collection. The Name property returns the name of a single object that matches the given index number.

The name returned by the Name property is the parameter to be passed to the Item property mentioned in “Setting channels and testing conditions” (p.11).

The above example repeatedly execute the steps from index 1 to Count using the For...Next statement. Note that the index parameter to be passed to the Name parameter is not zero-based but 1-based.

Handling an error

If you attempt to pass an invalid value, such as a value outside the range, to the property or call an unsupported function, an error occurs in the instrument driver. The IVI-COM instrument driver transmits an error to the client program as a COM exception each time it occurs in the instrument driver. VBA handles COM exceptions using the On Error Goto statement.

The following shows a programming code example, which includes the code for handling an error in addition to the one mentioned in “Setting channels and testing conditions” (p.11).

```
Sub CommandButton1_Click()
      On Error GoTo DRIVER_ERR:
      Dim inst As IKikusuiPwr01
      Set inst = New KikusuiPwr01Lib.KikusuiPwr01

      inst.Initialize "TCPIP::192.168.1.5::INSTR", True, True, ""

      Dim output As IKikusuiPwr01Output
      Set output = inst.Outputs.Item("OutputChannel1")
      output.VoltageLevel = 20.0
      output.CurrentLimit = 2.0
      output.Enabled = True

      inst.Close

      Exit Sub
DRIVER_ERR:
      Debug.Print Err.Description
End Sub
```

Explanation of the programming code example

This code handles an error using the On Error Goto statement. For example, if a name passed to the Item property is wrong, a value specified to VoltageLevel is outside the range, or communication with the instrument fails, a COM exception occurs in the instrument driver. The above code displays a simple message in the immediate window when an exception occurs.

Executing a program

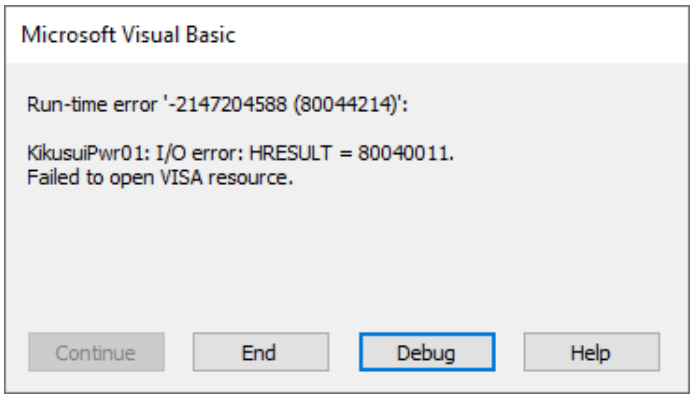
Execute the program you have created.

1 Click the button you have added in the Excel worksheet in Visual Basic.

The VBA macro starts, and then the program ends.

When the program fails to run

A COM exception (VBA runtime error) occurs.(p.13)



Programming Using a Class Interface

The following explains how to create a program using a class interface. With class interfaces, you can implement interchangeability that uses an instrument class interface by programming using an instrument class interface defined by the IVI specifications. Using interchangeability, you can replace the instruments without recompiling and re-linking the application.

Memo

- To use interchangeability, the IVI-COM instrument drivers must be provided for both instruments before and after the replacement and belong to the same instrument class. Interchangeability between different instrument classes is not supported.
- In programming using a class interface, the available instrument-specific functions are limited. To make the most of the functions specific to an instrument, use a specific interface (p.8) to create programs.

Preparation for programming

Creating virtual instruments

Before creating an application that uses interchangeability, you need to create a virtual instrument.

Memo

To avoid losing interchangeability, the application code cannot include statements dependent on specific IVI-COM instrument drivers (e.g. a directly created KikusuiPwr object) or specific VISA addresses (resource names) (e.g. [TCPIP::192.168.1.5::INSTR]).

The IVI specifications enable interchangeability by providing an IVI Configuration Store outside the instrument drivers and applications.

The applications control instruments through a special instrument driver called instrument class driver instead of directly control them using instrument-specific drivers.

The application controls the instrument by selecting the instrument driver DLL according to the data stored in the IVI Configuration Store and indirectly accessing the loaded instrument driver through a class driver function that does not depend on the instrument.

The IVI Configuration Store stores data as an XML file (e.g. C:/ProgramData/IVI Foundation/IVI/IviConfigurationStore.xml). Mainly IVI instrument drivers or some of the VISA/IVI configuration tools access the IVI Configuration Store through the IVI Configuration Server DLLs. Applications do not usually use it.

When using VBA, use the software NI-MAX (NIMeasurement and Automation Explorer) provided by National Instruments to configure the IVI driver.

The following explains how to create a virtual instrument using NI-MAX.

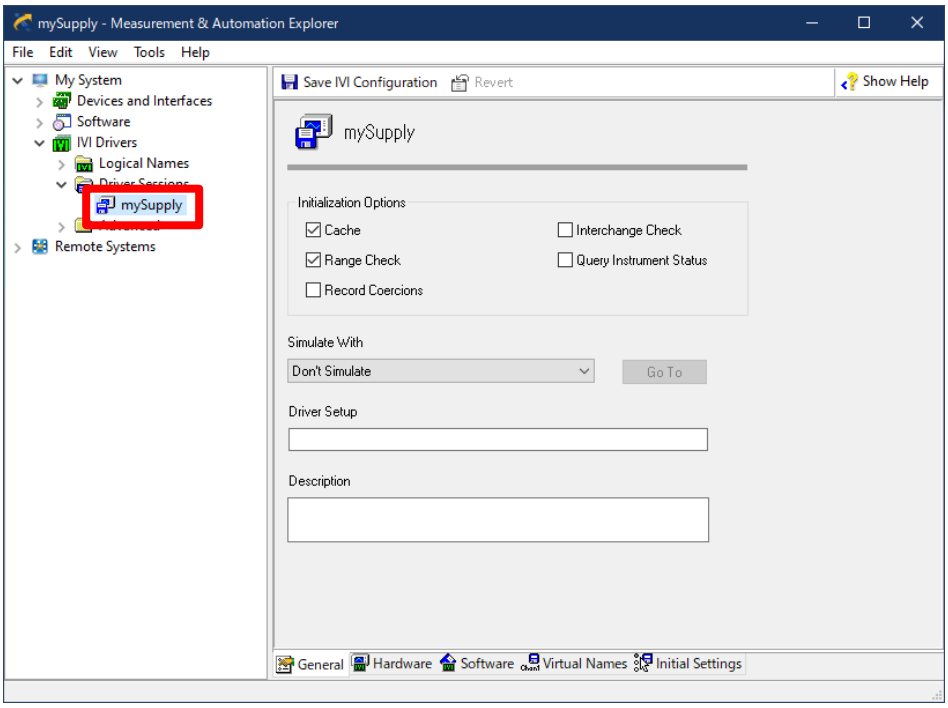
Creating Driver Sessions

1 **Launch NI-MAX, and then check the hierarchy under the IVI Drivers in the tree view on the left of the screen.**

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

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

In this example, mySupply is specified.

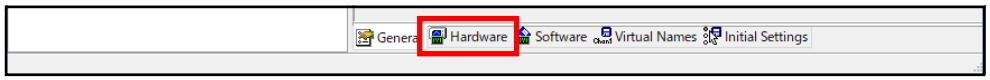


The Driver Sessions have now been created.

Creating a Hardware Asset

Hardware Asset allows you to specify the path to connect the instrument you will use.

1 **Select the Hardware tab.**



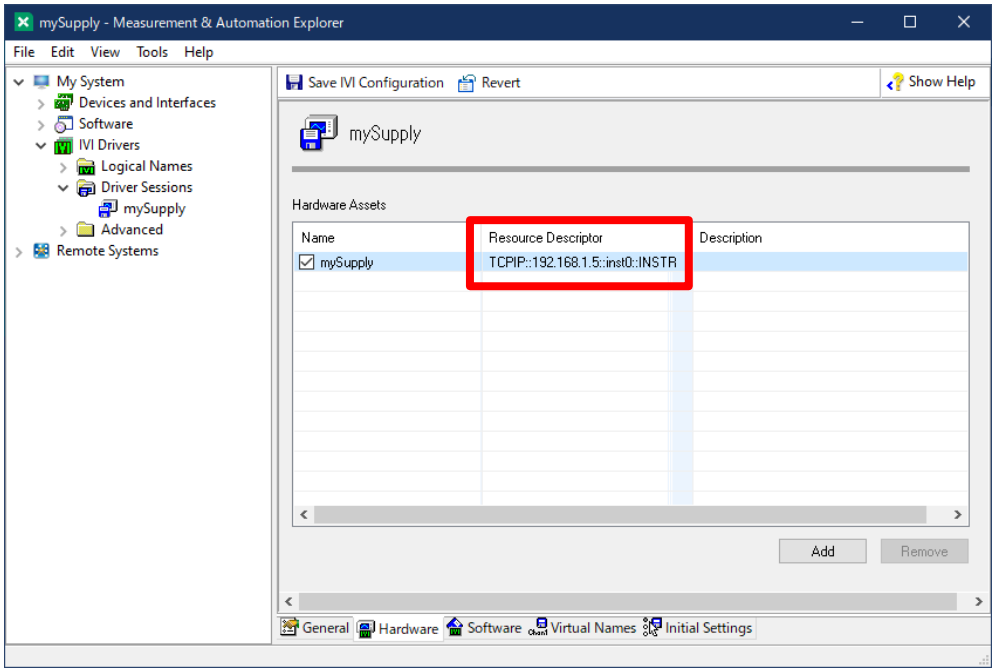
2 **Click Add to create a new Hardware Asset.**

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

In this example, mySupply is specified.

4 **In Resource Descriptor, specify the VISA address to which the instrument is connected.**

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

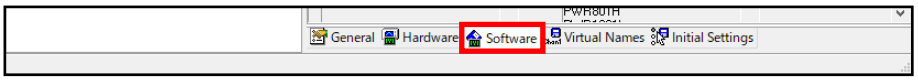


Hardware Asset has now been created.

Setting Software Module

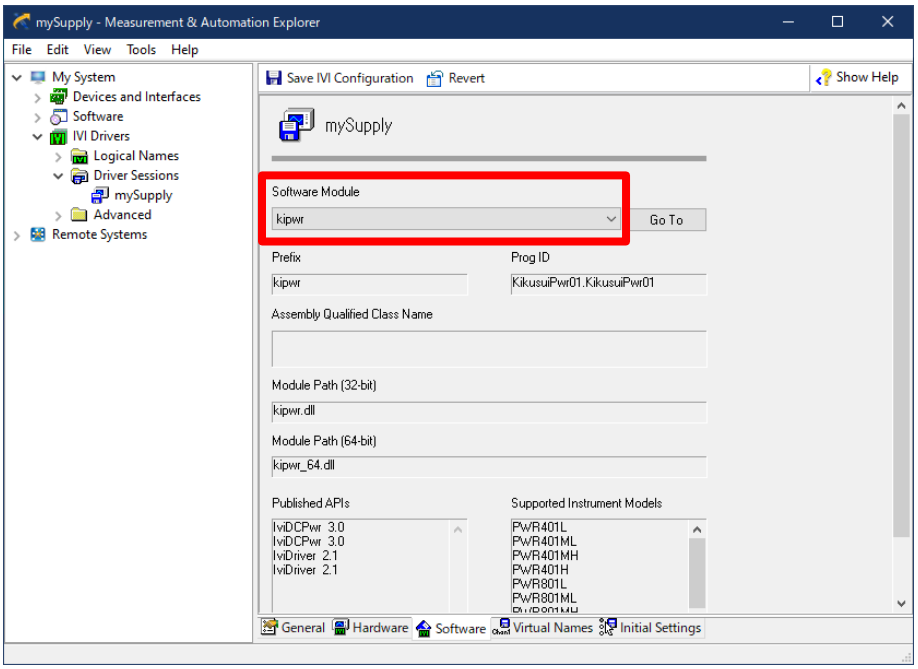
Software Module allows you to specify an instrument driver module (DLL module).

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.

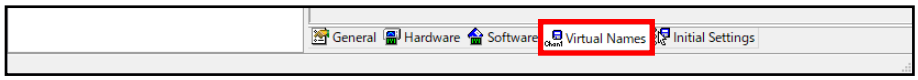


Software Module has now been specified.

Creating a Virtual Name

Virtual Name allows you to create a virtualized channel name for the instrument driver. This is because the valid channel name varies by instrument driver when you use an instrument driver to which channel name needs to be specified.

1 **Select the Virtual Names tab.**



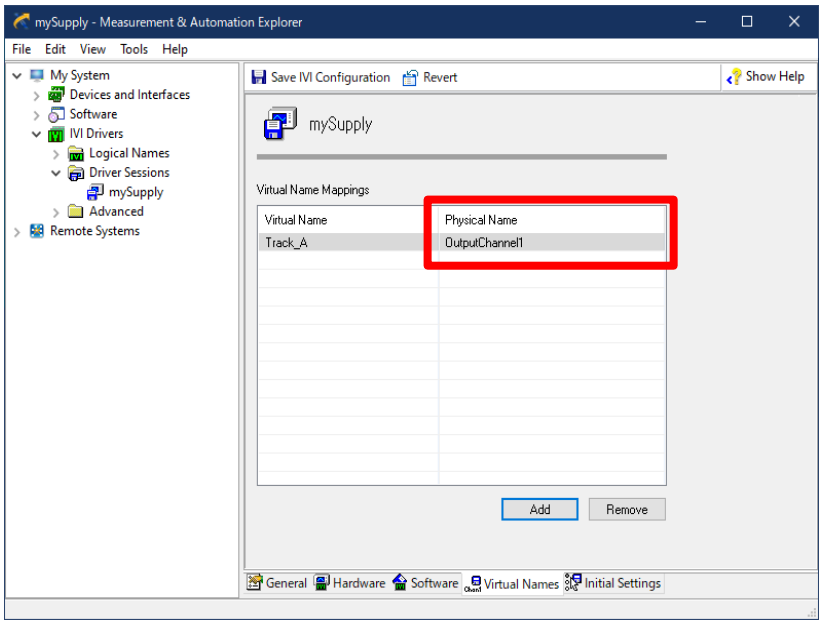
2 **Click Add to add a virtual name, and then enter Track_A in Virtual Name.**

In the **Physical Name** list, you will see the channel the instrument will use.

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

In this example, OutputChannel1 is selected.

> **Memo**
>
> In some cases, the list may not display all channel names, depending on the driver's mounting conditions or the configuration of the multi-channel power supply. For details about the available driver channel names, see the Readme file or the online help for the driver.



Virtual Name has now been created.

Setting a Logical Name

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

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

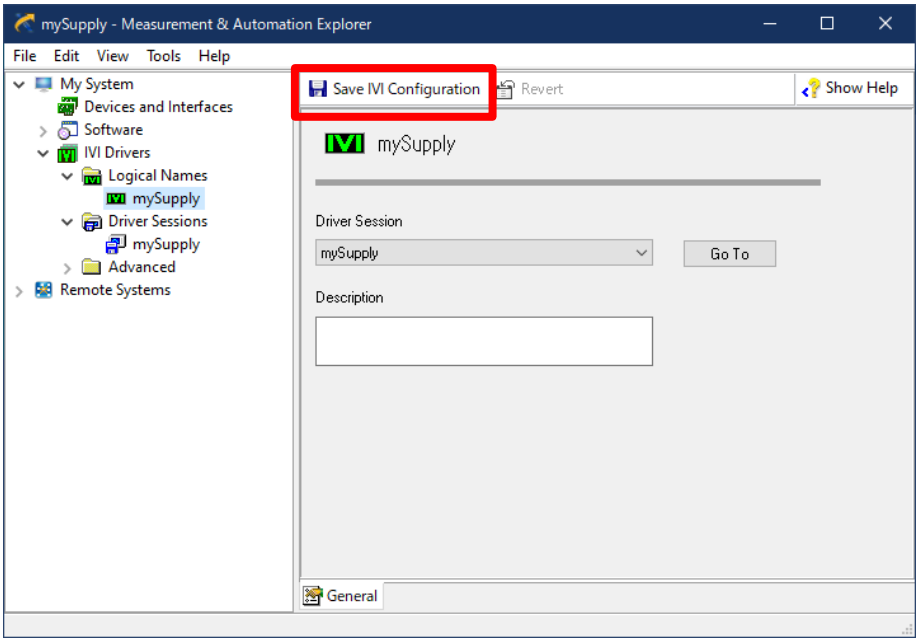
2 **Right-click Logical Name, and then select Create New (case-sensitive).**

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

In this example, mySupply is specified.

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

5 **Click Save IVI Configuration on the toolbar to save the settings.**



Logical Name has now been set.

Now, you have completed the creation of the virtual instrument.

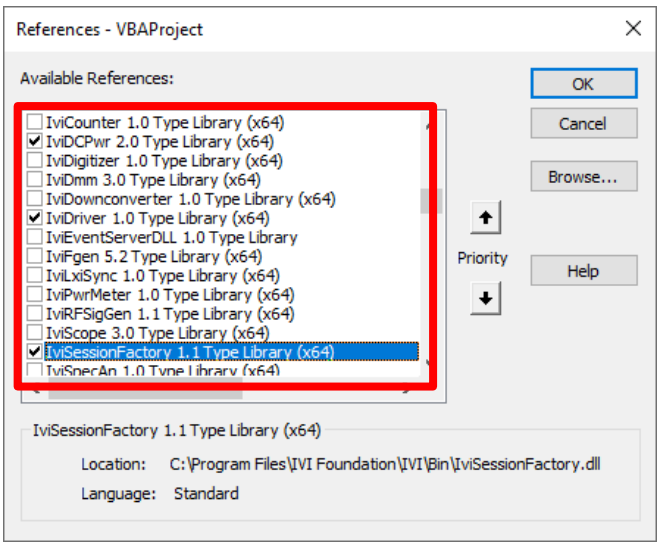
Importing a Type Library

Import a type library for the IVI-COM instrument driver you will use.

1 **Open the Visual Basic editor, and then click Tools > References.**

The References dialog box appears.

2 **Select IviDCPwr 2.0 TypeLibrary, IviDriver 1.0 Type Library and IviSessionFactory 1.1 Type Library.**



3 **Click OK.**

Building a program

Creating an object and opening/closing a session

An example of a programing code for creating an instrument driver object and opening and closing a session is shown below. To complete the programming, add the programming code below in the button handler source code you opened in “Creating a new macro” (p.7).

```
Sub CommandButton1_Click()

      Dim sf As IIviSessionFactory
      Set sf = New IviSessionFactory

      Dim inst As IIviDCPwr
      Set inst = sf.CreateDriver("mySupply")

      inst.Initialize "mySupply", True, True, ""

      inst.Close

End Sub
```

Explanation of the programming code example

For class interfaces, you cannot use any code that depends on a specific instrument. Therefore, this source code does not use the name, Kikusui. This code creates an IviSessionFactory object and obtains the reference to the IIviSessionFactory interface.

Next, it specifies the IVI logical name (virtual instrument), mySupply, created in “Setting a Logical Name” (p.20), and then calls the CreateDriver method.

Finally, it calls the Initialize method to pass the IVI logical name, and then closes the session. When the IVI logical name is passed, the VISA address specified in “Creating a Hardware Asset” (p.17) is used to execute the program. At this point, communication with the instrument begins.

The above code can run as an independent program. To execute the program for validation, see “Executing a program” (p.27).

Parameters for Initialize method

All IVI-COM instrument drivers include the Initialize method defined by the IVI specifications. The Initialize method has the following parameters:

Parameters	Type	Description
ResourceName	String	Character string to identify an instrument. Class interfaces use an IVI logical name to identify it.
IdQuery	Boolean	If TRUE, this method issues a query command, such as the *IDN? query, to ask the instrument model information.
Reset	Boolean	If TRUE, this method issues a restore command, such as the *RST command, to reset the instrument settings.
OptionString	String	Allows you to change the settings of the IVI definition.

How to set OptionString

OptionString allows you to set the following IVI definitions:

IVI definition	Default value
RangeCheck	TRUE
Cache	TRUE
Simulate	FALSE
QueryInstrStatus	FALSE
RecordCoercions	FALSE
Interchange Check	FALSE

OptionString is character string parameters. The following shows an example.

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

When formatting, be sure to note the following:

- If the set value is not specified, the default value will be applied.
- The set values are Boolean datatype. You can select a set value from the following: [TRUE], [FALSE], [1], [0].
- Function names and set values are case-insensitive. Uppercase and lowercase letters are not distinguished.
- When you set multiple items, use commas to separate them.
- Depending on the instrument driver, a “DriverSetup” parameters are supported.
  The “DriverSetup” parameters are used to specify an item not defined by the IVI specifications when the Initialize method is invoked. Its usage and format depend on the driver. Therefore, when you set a “DriverSetup” parameter, specify it as an item at the end of the ”OptionString”. For details about the “DriverSetup” parameters, see the Readme file or the online help for the driver.

Setting channels and testing conditions

IVI instrument drivers for power supplies are designed based on the premise that the IVI instrument drivers have multiple channels. Therefore, in many cases, properties or methods that control instrument panel settings implement the concept of an array of objects called Repeated Capabilities (or Collection in general computing terms for COM) in the IVI specifications. For example, instrument drivers for DC power supplies implement the Output Collection.

For class interfaces, IviDCPwrOutput is used. The following shows a programming code example, which includes the code for controlling the instrument's output channel in addition to the one mentioned in “Building a program” (p.22).

```
Sub CommandButton1_Click()
      Dim sf As IIviSessionFactory
      Set sf = New IviSessionFactory

      Dim inst As IIviDCPwr
      Set inst = sf.CreateDriver("mySupply")

      inst.Initialize "mySupply", True, True, ""

      Dim output As IIviDCPwrOutput
      Set output = inst.outputs.Item("Track_A")

      output.VoltageLevel = 20.0
      output.CurrentLimit = 2.0
      output.Enabled = True
      inst.Close
End Sub
```

Explanation of the programming code example

The above code obtains IIviDCPwrOutputs through the Outputs property of the IviDCPwr interface. The Item property passes the virtual name, Track_A, created in “Creating a Virtual Name” (p.19). When the virtual name is passed, the channel name specified by Physical Name is used to execute the program.

The VoltageLevel property specifies a voltage level. The CurrentLimit property specifies a limit value for the current. The Enabled property sets the ON/OFF state of the output.

Handling an error

If you attempt to pass an invalid value, such as a value outside the range, to the property or call an unsupported function, an error occurs in the instrument driver. The IVI-COM instrument driver transmits an error to the client program as a COM exception each time it occurs in the instrument driver. VBA handles COM exceptions using the On Error Goto statement.

The following shows a programming code example, which includes the code for handling an error in addition to the one mentioned in “Setting channels and testing conditions” (p.25).

```
Sub CommandButton1_Click()
      On Error GoTo DRIVER_ERR:
      Dim sf As IIviSessionFactory
      Set sf = New IviSessionFactory
      Dim inst As IIviDCPwr
      Set inst = sf.CreateDriver("mySupply")
      inst.Initialize "mySupply", True, True, ""
      Dim output As IIviDCPwrOutput
      Set output = inst.outputs.Item("Track_A")
      output.VoltageLevel = 20.0
      output.CurrentLimit = 2.0
      output.Enabled = True
      inst.Close
      Exit Sub
DRIVER_ERR:
      Debug.Print Err.Description
End Sub
```

Explanation of the programming code example

This code handles an error using the On Error Goto statement. For example, if a name passed to the Item property is wrong, a value specified to VoltageLevel is outside the range, or communication with the instrument fails, a COM exception occurs in the instrument driver. The above code displays a simple message in the immediate window when an exception occurs.

Executing a program

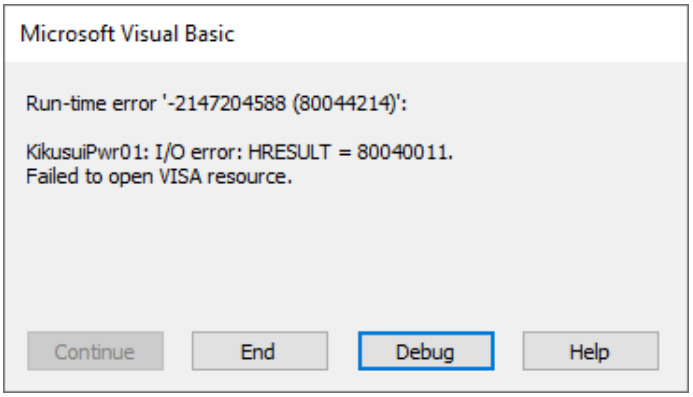
Execute the program you have created.

1 Click the button you have added in the Excel worksheet in Visual Basic.

The VBA macro starts, and then the program ends.

When the program fails to run

A COM exception (VBA runtime error) occurs.(p.26)



Replacing instruments

You can continue operating the system by changing only the Driver Session (IVI Configuration) settings for the instrument driver even after replacing the instruments. There is no need to modify the application.

You need to change the Driver Session settings for the following three items:

Item	Description
Hardware tab > **Hardware Assets** > **Resource Descriptor**	VISA address to which the instrument is connected.
Software tab > **Software Module**	Instrument driver to be used.
Virtual Names tab > **Physical Names**	Physical name to which the virtual channel name is mapped.

The application can run without being recompiled and re-linked if the settings are properly configured for the new instrument after replacement.

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

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

Memo

Interchangeability using an IVI class driver does not guarantee the operation before and after the replacement of instruments. After replacing any of the instruments, be sure to fully verify that the system operates properly.