IVI-COM Instrument Driver Programming Guide (VEE Edition)

Mar 2004 Revision 1.0

1- Overview

1-1 Using IVI-COM drivers in VEE

Originally, Agilent VEE is not suitable for use with IVI-COM instrument drivers. As a generic approach for utilising COM servers in VEE, the script engine that is a part of Formula functionality is used. In this case, however, the Late-Binding approach can only be used. Therefore, a COM server that you want to control has to equip IDispatch interfaces (or automation interfaces). Unfortunately IVI-COM instrument drivers equip custom interfaces that are directly derived from IUnknown without having any IDispatch interfaces, therefore they can't be used directly from VEE. At this point in time, the condition for using IVI-COM instrument drivers from VEE looks like the restriction on using them in Windows Scripting Host (WSH) environment.

However, there is a special tool that can wrap custom interfaces as if they are IDispatch interfaces. It is called "Script Adapter". By using this, you can use IVI-COM instrument drivers from VEE environment.

Notes:

How to use ScriptAdapteris is also introduced in the IVI-COM Instrument Driver Programming Guide (Windows Scripting Host / VBS Edition).

ScriptAdapter is a free software as DLL format and you can obtain it with source codes at http://homepage.interaccess.com/~hollp/ScriptAdapter.htm. The latest versions of Kikusui IVI-COM instrument drivers (VER 1.1.x.x or later) all come with the CoScriptAdapter.DLL that is already built, and it will be automatically installed when you set up the driver.

If you want to set up CoScriptAdapter.DLL manually, you need copy the file to an arbitrary directory on the hard disk, then perform self-registration (invoke DllRegisterServer) by using a tool such as REGSVR32.EXE. In the case that CoScriptAdapter.DLL is being installed by the IVI-COM driver setup program, the manual registration job is not needed. Normally this file is placed in the /Program Files/IVI/BIN directory.

This guidebook assumes that you use IVI-COM KikusuiPlz instrument driver (for KIKUSUI PLZ-4W/4WA series electronic load). You can also use IVI-COM instrument drivers for other models in the same manner.

2- Creating Application

As you launch VEE, an empty project with the "Main" screen will appear.

2-1 Importing Type Library

What you should do first is import the type library for the Script Adapter, by adding object references. Choose **Device | ActiveX Automation References** menu to show the **ActiveX Automation References** dialogue. Select "CoScriptAdapter 1.0 Type Library"

from the list. If it is not found in the list, click the **Browse** button and search for the CoScriptAdapter.DLL. The DLL is normally placed in the /Program Files/IVI/BIN directory.

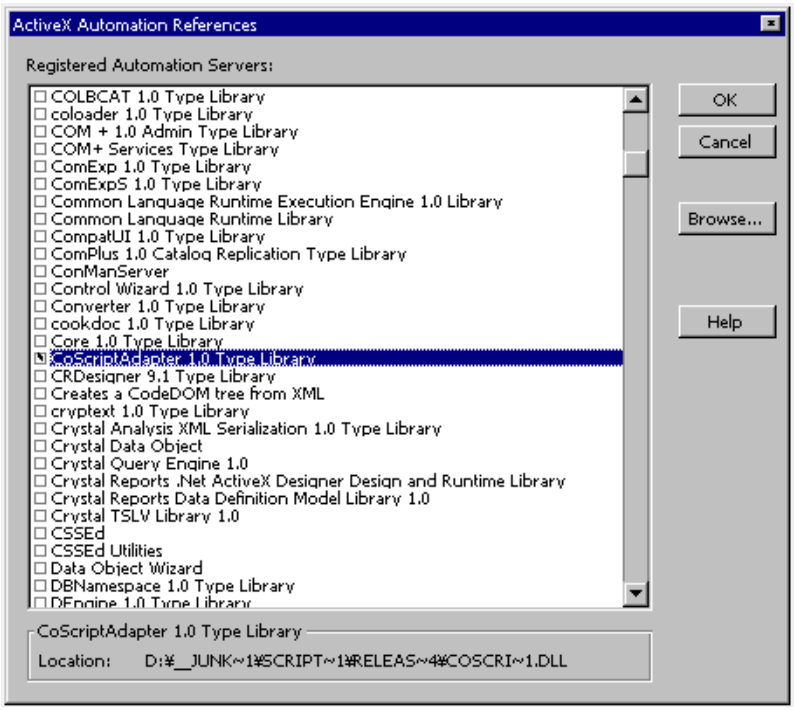

Figure 2-1 Importing CoScriptAdapter.DLL Type Library

2-2 Declaring Variables

The Formula functionality of VEE allows you to write scripts that are almost similar to VBS, however you can't declare variables with the Dim statement. Variables must be declared in advance in the VEE environment.

From the Main screen, choose **Data | Variable | Declare Variable** menu, then declare the following four variables.

Table 2-1 4 Variables

Name	**Type**	**Object Type**
adapter	Object	Library: CoScriptAdapter Class: ScriptAdapter Events: Not Enabled
ki	Object	Do not specify
kiPws	Object	Do not specify
kiPw	Object	Do not specify

2-3 Adding Formulas

Choose **Data | Formula** menu to add three Formulas, which of each has the Title setting "Open, "Settings", and "Close". The example described here does not use the input parameter "A" for simplicity. Choose **Delete Terminal | Input** on the context menu to delete unnecessary input parameters.

2-4 Writing Scripts

Write the following script in the "Open" Formula.

```
Set adapter = CreateObject("ScriptAdapter.Adapter");
Set ki = adapter.CreateAndWrap(
    "KikusuiPlz.KikusuiPlz").QueryInterface("IKikusuiPlz");
ki.Initialize( "ASRL1::INSTR", True, True, "");
Set kiPws = ki.Inputs;
Set kiPw = kiPws.Item("");
```

Write the following script in the "Settings" Formula.

```
kiPw.Function = 1;
kiPw.CurrentLimit = 1.2;
KiPw.SlewRate = 0.5;
kiPw.Enabled = True;
```

Write the following script in the "Close" Formula.

```
ki.Close();
```

After writing the scripts, wire them in the Open→Settings→Close order.

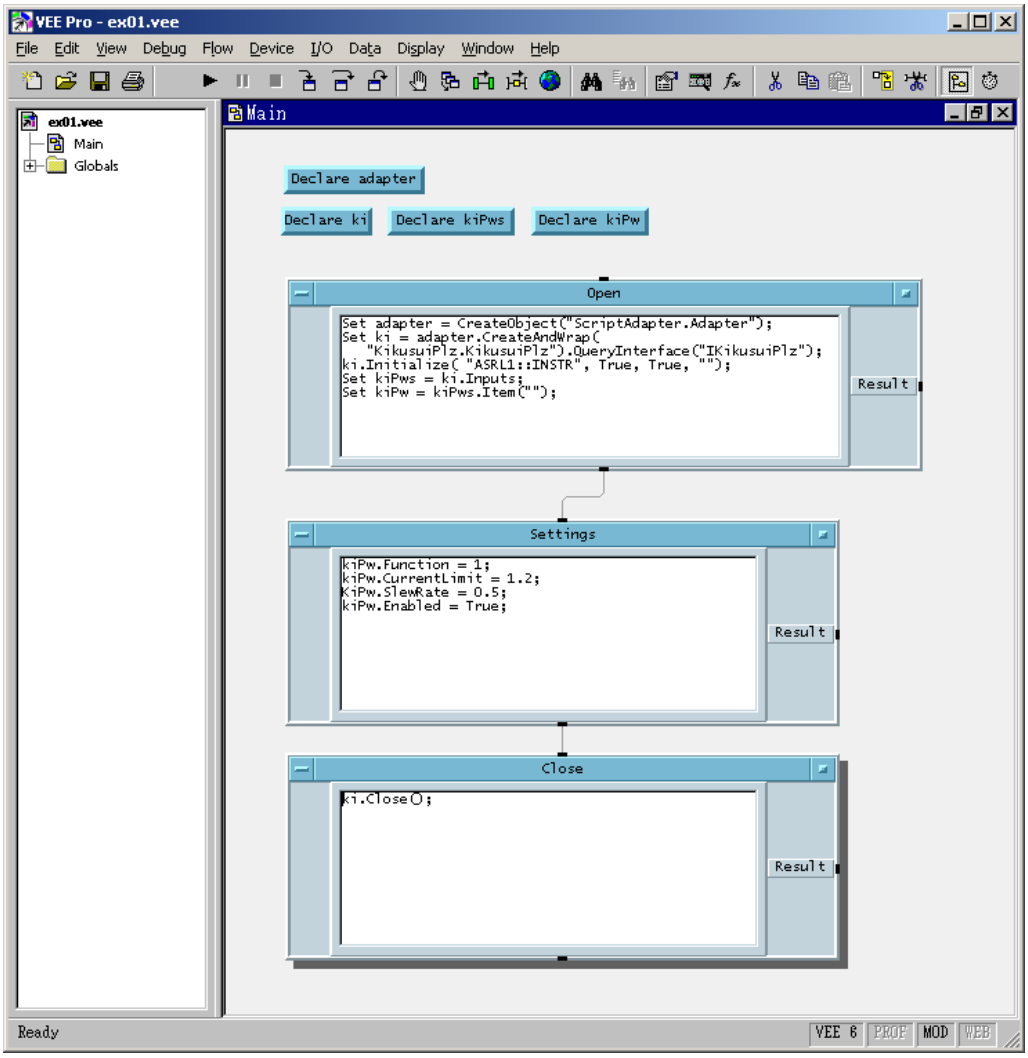

Figure 2-2 Writing Scripts

3- Description

3-1 Creating Objects

In the application, it is necessary to create the ScriptAdapter object first. The object is automatically created by the variable declaration, therefore there is no need to write the script.

Subsequently, create the instrument driver object. Here you use the `CreateAndWrap` method of the ScriptAdapter. This is equivalent to invoke the normal `CreateObject` method and then invoke the `WrapObject` method of the ScriptAdapter. The driver object created by `CreateObject` returns IIviDriver interface once, but the ScriptAdapter returns the wrapped IDispatch interface through the specified COM interface (IKikusuiPlz in this case).

3-2 Initiating Session

To initiate the instrument session, use the `Initialize` method. Now let's talk about the parameters for the `Initialize` method. Every IVI-COM instrument driver has an `Initialize` method that is defined in the IVI specifications. This method has the following parameters.

Table 3-1 Parameters for Initialize method

Parameter	Type	Description
`ResourceName`	`String`	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`	`Boolean`	Specifying TRUE performs ID query to the instrument.
`Reset`	`Boolean`	Specifying TRUE resets the instrument settings.

`OptionString`	`String`	Overrides the following settings instead of default:
		`RangeCheck`
		`Cache`
		`Simulate`
		`QueryInstrStatus`
		`RecordCoercions`
		`Interchange Check`
		Furthermore you can specify driver-specific options if the driver supports `DriverSetup` features.

`ResourceName` specifies a VISA resource. If `IdQuery` is TRUE, the driver queries the instrument identities using a query command such as `"*IDN?"`. If `Reset` is 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 Boolean type, you can use any of TRUE, 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 TRUE for `RangeCheck` and `Cache`, and 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 `Initialize` method, 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.

3-3 Accessing Channels

In general IVI-COM instrument drivers are, if in the case of instrument such as power supply or electronic load, designed assuming that multiple channels are equipped. Therefore the instrument driver's root interface (the variable "ki" in the above example) does not control instrument panel settings normally. To access each of instrument channels, acquire the reference to the collection through the `Outputs`(or `Inputs`) property once, then acquire the reference to the specified channel through the `Item` method.

```
Set kiPws = ki.Inputs;
Set kiPw = kiPws.Item("");
```

As this example uses KikusuiPlz driver to control electronic loads, use the `Inputs` property. Since the PLZ-4W/4W series is a mono channel electronic load, use the blank string (zero-length string) for the channel name to be passed to the `Item` method. For the instrument that supports multiple channels, it is necessary to specify an explicit channel name such as "CH1". See the online-help for detail about what channel names can be actually used.

Once you have acquired the reference to the specific channel, you can perform concrete instrument settings.

```
kiPw.Function = 1;
kiPw.CurrentLimit = 1.2;
kiPw.SlewRate = 0.5;
kiPw.Enabled = True;
```

This example sets the function to CC mode, current setting 1.2A, slew rate 0.5 A/µs, and input ON. Although the Function property originally accepts an integer of enumerated type, it is necessary to specify an immediate integer value since script environments do not support symbolic enumeration constants. Function = 1 means CC mode. See the online-help for detail about properties and methods that you can use.

3-4 Closing

Use the `Close` method to close the instrument driver session.

```
ki.Close();
```

3-5 Running Scripts

You can execute the previous code for the time being. Save the VEE project and then execute the program by selecting **Debug | Run/Resume** menu.

As you execute the program, instrument communications immediately start. If the instrument is actually connected and the `Initialize` method has succeeded, the script will

immediately finish. If a communication problem has occurred or the VISA library is not configured properly, a COM exception (runtime error) will be generated.

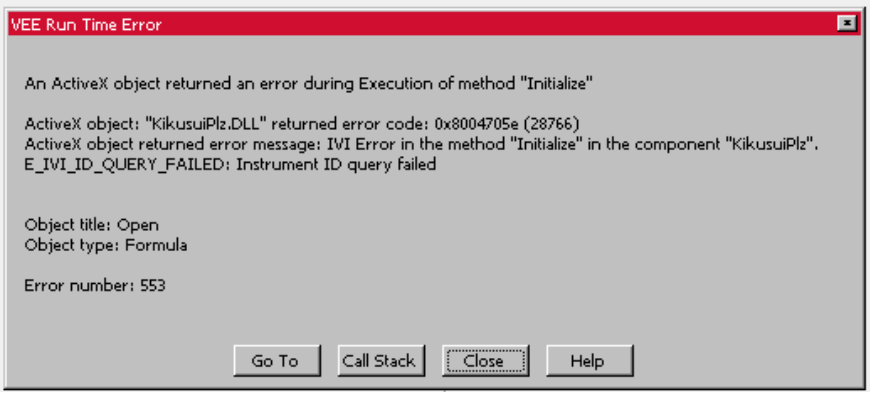

Figure 3-1 COM Exception