IVI-COM Instrument Driver Programming Guide (Windows Scripting Host / VBS Edition)

Mar 2004 Revision 1.0

1- Overview

1-1 Using IVI-COM driver in Windows Scripting Host

Windows Scripting Host (WSH) is a script engine, which comes with Internet Explorer 4.x or later. This guidebook describes how to use instrument drivers using Visual Basic Script (VBS).

VBS is a BASIC scripting language that is very similar to Visual Basic 6.0 (VB6), however it is originally unsuitable for use with IVI-COM instrument drivers. This is because VBS only allows to Late Binding approach for accessing COM servers. 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 IDispatch interfaces and not allowing Early Binding, therefore they can't be used directly from VBS.

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 VBS environment.

Notes:

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.

1-2 Creating Application

Although you can use arbitrary text editors for editing scripts, use Notepad program here. If you want to debug your scripts, it is recommended to use Visual Basic 6.0 integrated environment or Visual Studio 2003.NET. Visual Basic 6.0 is a superset language of VBS.

2- Sample Codes

Use Notepad or any arbitrary text editor to write the following codes.

```
Option Explicit

Dim adapter
Set adapter = CreateObject("ScriptAdapter.Adapter")

Dim ki
Set ki =
adapter.CreateAndWrap("KikusuiPlz.KikusuiPlz").QueryInterface("IKikusuiPlz")
ki.Initialize "ASRL1::INSTR", True, True, ""

Dim kiPws
Set kiPws = ki.Inputs

Dim kiPw
Set kiPw = kiPws.Item("")

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

ki.Close
```

VBS scripts can also be embedded in HTML documents. In this case use the <Script> tag to enclose the script part. A benefit of embedding scripts in the HTML is you can debug with Visual Studio.NET or Visual InterDev 6.0.

```
<SCRIPT LANGUAGE="VBS">

' the script begins
Option Explicit
Dim adapter
Set adapter = CreateObject("ScriptAdapter.Adapter")
...
...
ki.Close
' end of script

</SCRIPT>
```

2-1 Creating Objects

First, you need create the ScriptAdapter object. To create it, use the `CreateObject` statement. The Program ID must be `"ScriptAdapter.Adapter"`.

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).

This part can also be written like below:

```
Dim o
Set o = CreateObject("KikusuiPlz.KikusuiPlz")
Dim ki
```

```
Set ki = adapter.WrapObject(o).QueryInterface("IKikusuiPlz")
```

2-2 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 2-1 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.

Notes:

In the above table the parameters are explained as String type or Boolean type, however, the expression is originally for VB6 that uses strict data types. Variables in VBS are all Variant type.

2-3 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.

```
Dim kiPws
Set kiPws = ki.Inputs

Dim kiPw
Set kiPw = kiPwrs.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.

2-4 Closing Session

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

```
ki.Close
```

2-5 Saving and Running Scripts

You can execute the previous codes for the time being. First, save the script you edited with the Notepad. Then specify the file extension VBS instead of the default TXT (such as ex01.VBS). Now you can execute the stored VBS file from the Explorer.

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 (WSH runtime error) will be generated.

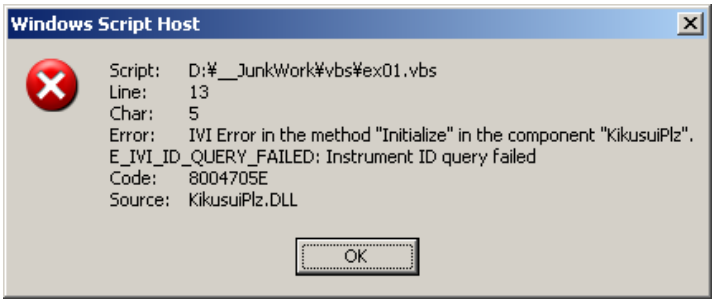

Figure 2-1 COM Exception

3- Error Handling

In the previous example, there was no error handling processed. However, setting an out-of-range value to a property 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-COM instrument drivers, every error generated in the instrument driver is transmitted to the client program as a COM exception. In case of VBS, COM exceptions can be handled by using `On Error Resume Next` statement. You can identify if an error has been occurred through the `Err` object.

```
On Error Resume Next
...
...

kiPw.CurrentLimit = 1.2
If err.Number <> 0 Then
    ' Error has occurred !!!
End If
```