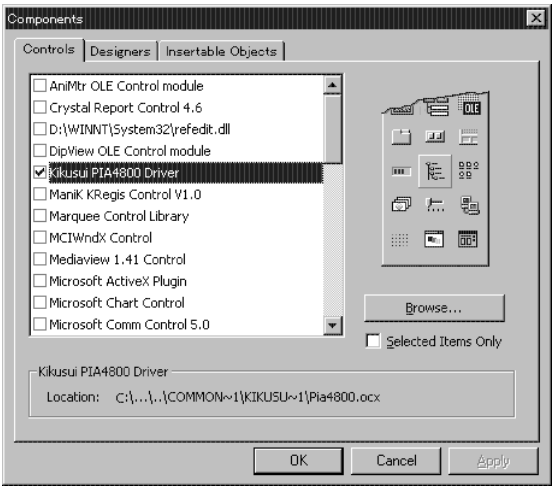
Part No. Z1-002-172, IA002253
Nov. 2003

Programming Guidebook

PIA4800 SERIES
INSTRUMENT DRIVER

PIA4800
Driver Objects



KIKUSUI

About This Manual

If you find any incorrectly arranged or missing pages in this manual, they will be replaced. If the manual it gets lost or soiled, a new copy can be provided for a fee. In either case, please contact Kikusui distributor/agent, and provide the "Part No." given on the cover.
This manual has been prepared with the utmost care; however, if you have any questions, or note any errors or omissions, please contact Kikusui distributor/agent.

Using PIA4800 Driver Objects

PIA4800 Driver Objects is a development kit used to create application programs for control of our DC power-supply units using the power-supply controller series PIA4800 connected to a GPIB or RS-232C. This kit allows PIA4800 control using virtually no character-based I/O routines.
PIA4800 Driver Objects is based on Microsoft's COM (Component Object Model) technology and power-supply controller technology. Therefore, this kit requires language tools, including Microsoft Visual Basic and Microsoft Office, which support COM clients and automation control systems. Basic knowledge of object programming such as Visual Basic is also required.
Kikusui supports only driver modules. Our support does not cover Visual Basic programming or other techniques including debugging. In addition, Kikusui is not responsible for the results of the application of a driver module. Each programmer shall be responsible for such results.

Trademarks

- Microsoft, Windows, Visual Basic, Visual C++, and ActiveX are trademarks or registered trademarks of Microsoft Corporation.
- All other trademarks or registered trademarks mentioned in this operation manual are the intellectual property of their respective owners.

Printed in Japan

⚠Safety Symbols

For the safe use of the product, the following symbols are used throughout this manual. Understand the meanings of the symbols and observe the instructions they indicate (the choice of symbols used depends on the products).

⚠WARNING	Indicates a potentially hazardous situation which, if ignored, could result in death or serious injury.
⚠CAUTION	Indicates a potentially hazardous situation which, if ignored, may result in damage to the product and other property.
	Shows that the act indicated is prohibited.
	Is placed before the sign “WARNING,” or “CAUTION” to emphasize these.

■ Inquiries

For questions concerning items not described in this document and abnormal performance of the system, contact us at the address below. Please note that our support covers only items related to the driver module, and not general inquiries, including those regarding programming techniques.

■ Before making inquiries

Before making inquiries, check the following information:

1. Operating environment

- Software environment, including OS and development tools
- Hardware environment, including the PC
  (When a GPIB is used, also include the GPIB environment.)
- System configuration

2. Details of problems

- Status of the problem
- History of the problem
- Location of the problem (if identifiable)
- Presence/absence of a sample code enabling repetition of the problem

■ Contact

As a rule, phone inquiries cannot be made.

Feel free to send us a fax or e-mail.

■ Address and other contact information

Kikusui Electronic Corp., CS Department, Service Section
Fax: (+81)45-593-0928
e-mail: service@kikusui.co.jp

Document Structure

This guidebook is composed of the following sections:

Chapter 1 Outline

Provides an outline of the system and its operating environment

Chapter 2 Introduction and Setup

Describes the setup procedures for and operation of Visual Basic before programming is begun

Chapter 3 Basic Programming Techniques

Covers basic programming techniques and sample programs

Appendices

The following appendices are provided:

Appendix A: Distribution of Applications

Appendix B: Use of Other Programming Languages

Appendix C: Object Quick Reference

Contents

Chapter 1 Outline

⚠CAUTION
- Incorrect using a DC power-supply may result in electrical shock or in damage to load. It is very important to write the program that the DC power-supply always operates correctly in programming.
- Before starting programming, be sure to read the operation manuals for both the PIA4800 and connected instruments.

When you write a program for controlling GPIB devices in BASIC or C, it is generally necessary to be acquainted with the operation of such devices and to have a substantial knowledge of GPIB commands. However, if PIA4800 Driver Objects is used, these devices can be controlled through the creation of an application program without knowledge of GPIB commands or complex syntax.

PIA4800 Driver Objects is composed of library modules that encapsulate operations based on GPIB and RS-232C commands. PIA4800 Driver Objects enables the control of DC power-supply units connected to the power-supply controller PIA4800 series by a VISA-compatible GPIB card or RS-232C port.

This guidebook presents examples based on Microsoft Visual Basic 5.0 and Excel 97.

When Using a Programming Language Other than Visual Basic 5.0 or Excel 97

This guidebook presents examples based on Microsoft Visual Basic 5.0 and Excel 97. The guidebook applies to other programming environments, as long as they support ActiveX control. When Visual Basic is not to be used, refer to Appendix B, “Use of Other Programming Languages.”

Although it is possible to use the Driver Object with C language, it is not recommended because there are tremendous amount of processing regarding COM and it not realistic.

Operating Environment for Driver Objects

GPIB users:

- A PC running on Windows 95, Windows 98, Windows NT4.0 Windows 2000 or Windows XP
- A VISA-compatible GPIB card
- A GPIB cable
- Microsoft Visual Basic 4.0 or later, or Microsoft Office 97 or later
- NI-VISA Ver.2.5 or later, Agilent I/O Library K.01.00 or later or KI-VISA2.2.3 or later

RS-232C users:

- A PC running on Windows 95, Windows 98, Windows NT4.0 Windows 2000 or Windows XP
- At least one RS-232C port
- A RS-232C cross cable
- Microsoft Visual Basic 4.0 or later, or Microsoft Office 97 or later
- NI-VISA Ver.2.5 or later, Agilent I/O Library K.01.00 or later or KI-VISA2.2.3 or later

---

NOTE

- If RS-232C is to be used, set the baud rate to 19,200 bps in the communication settings of the PIA4810 or PIA4830.
- For Windows NT4.0, install Service Pack 6a or later.
- For Windows95, install IE4 or later.

Chapter 2 Introduction and Setup

Installation (GPIB/RS-232C Card)

VISA-compatible GPIB card

When a GPIB is to be used, it is necessary to have a VISA-compatible GPIB card on your PC. A number of models are available from the manufacturer. Select one suitable for the PC and Windows version used. For details, see the operation manual for the PIA4810/4820 series.

For the latest information on the GPIB hardware and driver software, contact your respective VISA-compatible GPIB card vendor.

If you are using the RS-232C interface, one serial port will be used. If you do not have a VISA license of other companies, use KI-VISA.

Connecting an interface cable

Install the interface card on your PC, then connect a GPIB cable or an RS-232C cross cable to the PC.

NOTE

- About KI-VISA
  KI-VISA does not support VXI, PXI, or TCP/IP. If VXI, PXI, or TCP/IP is used on the computer, use NI-VISA or Agilent IO Library instead of KI-VISA.

Installation (Visual Basic/Excel 97)

This guidebook presents examples based on Microsoft Visual Basic 5.0 or Excel 97. For installation, see the Visual Basic or Microsoft Office manual.

Installation (PIA4800 Driver Objects)

To install the driver, follow the instructions on the HTML page that opens through the Auto Run function.

When other than D is to be used for the CD-ROM drive, change D to a proper drive name, then follow the instructions on the screen.

Upon completion of this step, the driver module is installed in the following two folders;

C:\Program Files\Kikusui\Pia4800 Utils.V2

C:\Program Files\Common Files\Kikusui Shared

Operating Visual Basic

After making hardware and software settings, start Visual Basic to begin programming.

Visual Basic 5.0

Start VB5 and, in the [New Project] window, select "Standard EXE." A project with a single form is created in preparation for starting programming.

Excel 97

Start Excel and, in the [View] menu, select [toolbars] and then [Visual Basic]. The Visual Basic tool pallet will become active. On the Visual Basic tool pallet, click on the Visual Basic Editor button. Visual Basic will start up. In the [Insert] menu on Visual Basic, select [User Form] to add a new form to the project.

NOTE

- Excel 97 as a programming environment requires different techniques from those necessary for its spreadsheets and graphs. These techniques form the VBA (Visual Basic for Applications) environment, which allows applications to be created as in the ordinary VB (Visual Basic 5.0) environment.

  Unlike VB5, Office 97 does not permit stand-alone EXE modules to be created. All VBA applications created in Office 97 require the same environment when they are to be executed. For example, to execute Excel 97 applications, Excel 97 is necessary.

Adding Controls

Before BASIC codes can be written using Visual Basic, an additional component have to be added. These components enable Visual Basic to recognize PIA4800 Driver Objects, making the object library, extended syntax, and type information available.

When Visual Basic 5.0 is to be used, select [Components] from the [Project] menu. When Excel 97 is to be used, start up Visual Basic. In the [Tools] menu, select [Additional Controls]. The [Component] dialog box will appear as shown in Fig. 2-1. On the list in the [Components] dialog box, check Kikusui PIA4800 Driver or Pia4800 Class.

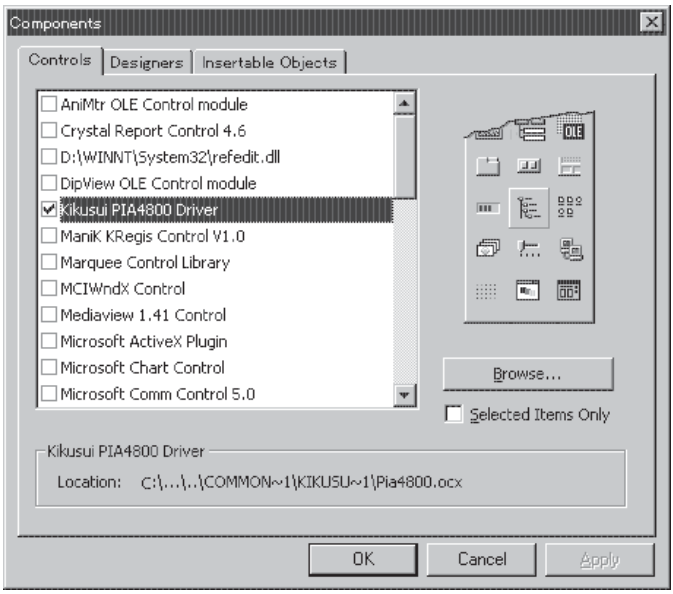


Fig. 2-1 Component dialog on Visual Basic 5.0

NOTE

- Component additions have to be made for each project when Visual Basic 5.0. Once set, the components do not require resetting. Settings have to be made only for a new project.

Chapter 3 Basic Programming Techniques

First Code Described in Visual Basic

With Visual Basic, event-driven programs such as button handlers or menu handlers can be created. The following is the easiest procedure for operating PIA4800 Driver Objects on Visual Basic:

- Paste the PIA4800 control on the Form.
- Add a command button to the Form.
- Write a handler code for the button event.

1. With the Form displayed in Visual Basic, click on PIA4800 in the [Tool box]. The cursor will change into a cross.
2. Put the cursor on the Form and drag it to specify an arbitrary-sized range. After the cursor is released, the PIA4800 control can now be pasted on the Form.
3. Using the same procedure as above, add a command button to the Form.
4. Double-click on the command button. The code editor will appear and the cursor will move to the position for writing a button handler.

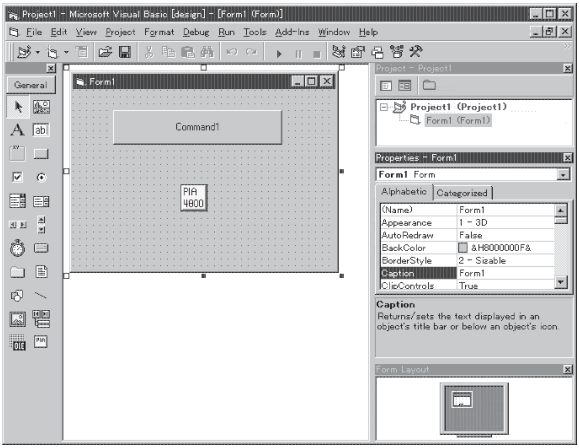


Fig. 3-1 Adding a button handler to the Form (VB5)

Pia4800 Objects

A name has to be assigned to each PIA4800 Control instance. Click on the PIA4800 Control and look at the Property window. The Name property will show “Pia48001” by default. The example below describes the pia instance after this default is changed to “pia.”

The pia instance of the object represents a PIA4810 or PIA4830, which is connected to the PC via GPIB or RS-232C. If this object is used, virtually all functions of the PIA4800 series are available from this program.

Basically, an object has properties and methods. With PIA4800 Driver Objects, the functions of the PIA4800 series can be used through a sub-object in the Pia4800 object hierarchy. The object hierarchy is classified as follows:

Supply Object

Used to set basic elements, such as voltage, current, and output status, for the connected device.

Supply Collection Object

A collection object composed of four supply objects.

The effective number of supply objects is determined based on the structure of the Module object in the upper layer.

When the control boards OP01-PIA and OP02-PIA are used as Module objects, up to two Supply objects are effective.

For a DC power-supply unit with a digital remote-control function, the effective number of Supply objects is determined based on the number of output channels.

Module Object

An object composed of the control board OP01-PIA or OP02-PIA, or DC power-supply units with a digital remote-control function.

Module Collection Object

A collection object that incorporates 35 Module objects.

The PIA4800 series can control up to 32 DC power-supply units with an analog remote-control function, and 31 units with a digital remote-control function.

Only connected modules can be controlled.

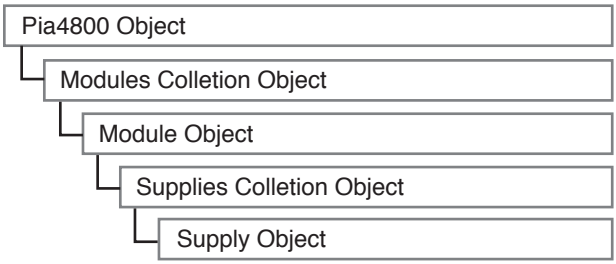


Fig. 3-2 PIA4800 Driver Object hierarchy

Access to Pia4800 Objects

Connecting GPIB/RS-232C

With the Pia4800 objects, it is necessary to first make connections to the PIA4810 or PIA4830 by the Connect method. Specify a device name for this method.

When a GPIB is to be used, specify device names GPIB::1::INSTR through GPIB::30::INSTR, which correspond to GPIB addresses 1 through 30. Specify a device name based on the GPIB address of the PIA4800 series.

With RS-232C, use ASRL::1::INSTR, ASRL::2::INSTR, or some other name that corresponds to the serial port used. Double-click on the button pasted on the Form. Then, enter the following code in the button handler:

```
pia.Connect "GPIB0::1::INSTR"
```

This connects to the PIA4800 series using a specified device name. Once connections are made, some information on the PIA4800 series will become available.

```
Dim a As String, b As Integer
a = pia.LegalModelName
b = pia.Version
```

The LegalModelName property returns the model name of the PIA4800 series. For the PIA4810, “PIA4810” is returned. The Version property returns the ROM version of the PIA4800 series. The upper byte and lower byte represent major version and minor version, respectively. If the version is 1.23, the Version property returns &H0117(&H17 = 23).

Voltage Setting

Voltage setting, which enables a voltage to be output, is one of the applications of PIA4800 Driver Objects.

The discussion to follow is based on the assumption that the control board OP01-PIA is equipped with Node address 1 of the PIA4800 series, with the DC power supply PAK35-10A electrically and logically connected to Channel 2.

The functions of PIA4800 Driver Objects depend to a large extent on the control board and the DC power-supply unit connected to the board. Even if properties of PIA4800 Driver Objects exist, its functions cannot operate to their full potential unless the control board and DC power-supply unit have corresponding functions to support them.

The Pia4800 object has two types of collection objects: Modules sub-object, which is the collection object of Module, and Supplies sub-object, which is the collection object of supply.

Access to the sub-objects of Pia4800 Driver Objects is extremely important. Without proper knowledge of how to deal with the sub-objects and the object layers, it will be difficult to control the DC power-supply unit.

```
With pia.Modules(1).Supplies(2)
        .Voltage = 5    'set voltage 5.0V
        .Current = 1    'set current 1.0A
        .Output = True  'set output on
End With
```

In the above example, the output is turned on after the output voltage is set to 5.0 V and the output current is set to 1.0 A for the DC power-supply unit, which is connected to Channel 2 on the OP01-PIA using Node Address 1.

Some properties can obtain a present set value as well as setting.

```
Dim v As Double, i As Double
With pia.Modules(1).Supplies(2)
  v = .Voltage    'get voltage
  i = .Current    'get current
End With
```

Obtaining Monitor Values

The voltmeter and ammeter can be used via OP01-PIA on multiple DC power-supply units connected to the PIA4800 series.

Access to the monitor function enables monitoring of the output from the DC power-supply unit connected to the PIA4800.

As with the voltage setting and reading function, the monitor function can be activated by accessing Pia4800's sub-objects.

```
Dim v As Double, i As Double
With pia.Modules(1).Supplies(2)
  v = .MonVoltage    'get voltage monitor
  i = .MonCurrent    'get current monitor
End With
```

The Supply object's properties MonVoltage and MonCurrent return values that correspond to respective properties. They are read-only values, and do not allow settings to be made. The status of the power-supply unit and its peripherals can be determined in a similar manner as with the Status and DigialIn properties.

Handling Errors

The above example assumes that all operations terminate with no problems. In the event of a communication error, the program returns a run-time error. In such a case, Visual Basic displays a message box and stops the program on the line at which the error occurred.

Causes of run-time errors

The run-time errors attributable to PIA4800 Driver Objects are classified as follows:

- GPIB/RS-232C communication errors
- PIA4800 operation errors

Whenever any problem occurs in GPIB/RS-232C communication, PIA4800 Driver Objects issues an run-time error. It also issues a run-time error when an invalid operation is performed.

An operation error occurs when one of the following operations is performed:

- Accessing without mounting the control board on the PIA series.
- Accessing without turning ON the power to the expansion unit PIA4820
- Setting a value that exceeds the valid range
- Attempting to determine a monitored value (voltage and current) using the control board OP02-PIA dedicated to settings only.

The above operations cause PIA4800 Driver Objects to issue a run-time error. In the event of such an error, the cause of the error can be identified by accessing the Err object from the program. The Err object is a standard function of Visual Basic. The Number property indicates the error code, and the Description property provides details on the error. All error codes produced by PIA4800 Driver Objects have a value of over 1200.

Error trap

Normally, a run-time error forces the program to stop operation.

However, the On Error Goto statement can be used to disable the stop operation if an error handler is specified in advance. This statement allows the program to perform exceptional processing to restore operation.

Sample Program

The following describes the program blocks used to access the PIA4800 series. They perform fundamental settings, monitor outputs, and error handling.

```
Private Sub Command1_Click()
  Dim modl As PIA4800LibCtl.Module
                              'declear Module object
  Dim sup As PIA4800LibCtl.Supply
                              'declare Supply object

  On Error GoTo GPIB_EXP
                              'Support error handling

  pia.Connect "GPIB0::1::INSTR"
                 'Connect "GPIB0::1::INSTR" device
  Set modl = pia.Modules(1)    'Get Module object
  Set sup = modl.Supplies(2)   'Get Supply object

  sup.Voltage = 12.3           'Set 12.3V
  sup.Current = 1              'Set 1A
  sup.Output = True            'Output on

  Dim dVout As Double, dIout As Double

  dVout = sup.MonVoltage       'Get voltage output
  dIout = sup.MonCurrent       'Get current output

  pia.Disconnect               'disconnect
  EXIT Sub

GPIB_EXP:
  MsgBox Err.Description       'show error message
  pia.Disconnect               'disconnect

End Sub
```

Other Basic Functions

There are other basic functions which are accessible by Pia4800 and its sub-objects, including OVP settings and digital control output.

For the properties and methods, refer to Appendix C “Object Quick Reference” and Online Help.

NOTE

- In the code editor, enter “Pia4800” and press the F1 key. The online document useful in PIA4800 programming is displayed to provide detailed information on Pia4800, including its properties and the syntax for methods.

  To display the Object Browser, press the F2 key. The browser provides detailed descriptions of the syntax for each object.

- Limitations on the Apartment Threading Model

  With VB/VBA, object instances are created on the main apartment (the thread on the Form). Do not access objects from other thread contexts.

  Using the custom interface with VC++, an object interface can be created on any apartment. If it is necessary to access the object, be sure to do so from the thread context that has been used to create the interface.

Appendix

Appendix A Distribution of Applications

To distribute your applications using PIA4800 Driver Objects, it is necessary to have driver modules(DLL and OCX files) provided by Kikusui.

Kikusui's driver modules may be freely distributed, provided that it is used to control the PIA4800 series.

Distributing Driver Modules

Necessary libraries can be distributed and registered following the installation of PIA4800 Utilities, which accompanies the PIA4800 series.

Appendix B Use of Other Programming Languages

This Guidebook presents examples based on Visual Basic 5.0 and Excel 97/VBA. This appendix describes the use of other programming languages and versions available for PIA4800 Driver Objects.

Visual Basic Family

Visual Basic 6.0

Can be used in the same way as Visual Basic 5.0.

Access 97 and Word 97 or later

All VBA-related tools bundled in Microsoft Office 97 support VBA Version 5.0. As with Visual Basic 5.0 and Excel 97, Access 97 and Word 97 can be used for PIA4800 Driver Objects.

Access 7.0 (95) and Visual Basic 4.0 (32-bit version)

Both tools have the same VB/VBA syntax as other relevant tools, except that they require slightly different reference setting procedures and Visual Basic startup procedures. Program codes can be written in virtually the same manner as other relevant tools.

Excel 7.0 (95)

Excel 7.0/VBA does not provide the New keyword used for the application of ActiveX control and object programming. To create object instances, use the CreateObject statement.

```
Dim Pia As Object
Set Pia = CreateObject("Pia4800.Pia4800")
Pia.Connect "DEV1"
With Pia.Modules(1).Supplies(2)
   .Voltage = 5            'set voltage 5.0V
   .Current = 1            'set current 1.0A
   .Output = True          'set output on
End With
```

The program ID string "Pia4800.Pia4800" can be used to identify the ActiveX control object PIA4800 Driver. The Object type does not provide a syntax checking mechanism for compilation time. Instead, all syntax checks are delayed until the program is executed. (Pia4800 type is operable, but New is not.)

Other 16-bit versions

PIA4800 Driver Objects is incompatible with all 16-bit VB/VBA tools, including Visual Basic 1.0/2.0/3.0/4.0 (16 bit) and Microsoft Office 4.x (Excel 5.0 and Access 2.0).

Borland Delphi Family

Delphi 3.0 or later

With Delphi 3.x, first select ActiveX Control in Components. The dialog will appear. From the registered components, select Kikusui PIA4800 Driver and click on the Install button. The PIA icon is then added to the ActiveX page on the Component Pallet.

Click on the PIA icon, and paste the PIA4800 components on the Form. In addition, paste a button on the Standard page of the Form. Click on the Button 1 to display the code editor.

For objects other than those for Pia48001, declare in a Variant type.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   m  : Variant;
   s  : Variant;
bigin
   Pia48001.Connect('GPIB0::1::INSTR');
   m  := Pia48001.Modules[1];
   s  := m.Supplies[1];
   s.Voltage  := 16;
   s.Output   := True;

end;
```

Delphi 2.0

The compatibility of Delphi 2.0 has not yet been verified.

Delphi

The first version of 16-bit Delphi is incompatible.

Visual C++ 4.x

With Visual C++ 4.x, the applications have to be Automation-enabled.

Creating a new project

To create a new project, specify OLE Automation using AppWizard.

Previously created projects

To add an OLE-related code to a previously created project, add the following line to the end of Stdafx.h:

```
#include <afxdisp.h>
```

In addition, add the following call code to the InitInstance function of the application class:

```
AfxOleInit();
```

NOTE • The application has to use MFC.

Upon establishment of the necessary environment, create a PIA4800 controller class.

To do so, click on the [Add a Class] button in ClassWizard. Then, from the pull-down menu, select [From OLE typelib ...].

To add an OLE type library, select "\Program Files\Common Files\KIKUSUI Sheard\Pia4800.ocx."

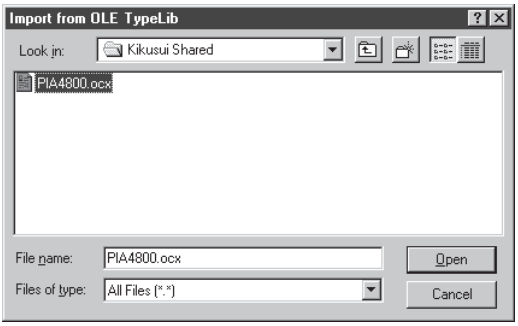


Fig. A-1 Import from OLE Typelib

Select File and open ClassWizard to display the dialog shown below.

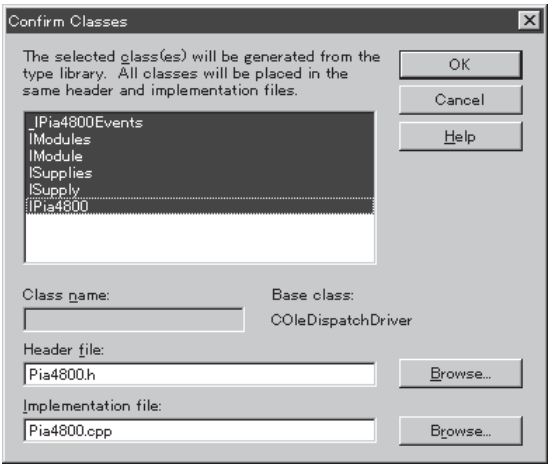


Fig. A-2 Confirm Classes

Through this function, the control class (Pia4800.h/Pia4800.cpp) is created.

Include the “Pia4800.h” file (default file name), and write the code for execution shown below.

```
try{
  IPia4800 pia;
  pia.CreateDispatch( "Pia4800.Pia4800");
  pia.Connect("GPIB0::1::INSTR");

  VARIANT vEmpty;
  vEmpty.vt = VT_ERROR;
  vEmpty.scode = DISP_E_PARAMNOTFOUND;

  IModules ms = pia.GetModules(vEmpty);

  IModule m = ms.GetItem(2);
  CString strModuleName = m.GetModuleName();
  ::AfxMessageBox( strModuleName);

  ISupplies sups = m.GetSupplies(vEmpty);
  ISupply   sup = sups.GetItem(1);

  CString strModelName = sup.GetModelName();
  ::AfxMessageBox( strModelName);

  sup.SetRemote( TRUE);
  sup.SetVoltage( 10.0);
  sup.SetCurrent( 5.0);
  sup.SetOutput( TRUE);

  double dVout, dIout;
  dVout = sup.GetMonVoltage();
  dIout = sup.GetMonCurrent();

  pia.Disconnect();
}
catch(COleDispatchException* pE){
  CString strError = pE->m_strDescription;
  pE->Delete();
  ::AfxMessageBox(strError);
}
```

Visual C++ 5.0/6.0

Visual C++ 5.0 and 6.0 can use the same programming maner as Visual C++ 4.0. However, Visual C++ 5.0 and later offer the COM smart pointer, which enables objects to be created using C++ as easily as with Visual Basic.

A description of the use of the COM smart pointer is given below.

Import of a type library

First, the #import directive has to be used to allow the compiler to access the type library.

```
#import "c:\Program Files\Common Files\Kikusui Shared\Pia4800.ocx" named_guids
```

Write the code for execution as follows:

```
try {
  PIA4800Lib::IPia4800Ptr spPia( __uuidof( PIA4800Lib::Pia4800));
  PIA4800Lib::IModulesPtr spModls;
  PIA4800Lib::IModulePtr spModl;
  PIA4800Lib::ISuppliesPtr spSups;
  PIA4800Lib::ISupplyPtr spSup;

  spPia->Connect( L"DEV1");
  spModls = spPia->GetModules();
  spModl = spModls->GetItem(1);
  spSups = spModl->GetSupplies();
  spSup = spSups->GetItem(1);

  spSup->PutRemote( VARIANT_TRUE);
  spSup->PutCurrent( 1.0);
  spSup->PutVoltage( 12.5);
  spSup->PutOutput( VARIANT_TRUE);

  double dIout, dVout;
  dIout = spSup->GetMonCurrent();
  dVout = spSup->GetMonVoltage();

}
catch( _com_error e){
  _bstr_t desc = e.Description();
  HRESULT hr = e.Error();
  ::MessageBox( NULL, desc, "Error", MB_ICONEXCLAMATION);
}
```

NOTE

- With the MFC application, write the #import sentence after #include “standafx.h” or within stdafx.h.

  With the MFC application, if Automation is not specified, override InitInstance and ExitInstance in the Application Class and call CoInitialize(NULL) and CoUninitialize( ), respectively. These calls are not necessary when Automation is specified with the MFC application (that is, when AfxOleInit() has already been called with InitInstance).

  In a non-MFC application, call CoInitialize(NULL) at the beginning of the program, and CoUninitialize() immediately before its termination.

  In the process of accessing a COM object using #import directive, to pass strings such as those for the Connect method, they have to be converted to Unicode.

Appendix C Object Quick Reference

Pia4800 Object

Property	Type	Description
Vendor (Read Only)	String	Returns the device vendor name "KIKUSUI"
LegalModelName (Read Only)	String	Returns the formal model name "PIA4810" or "PIA4830"
Version (Read Only)	Integer	Returns the ROM version of a device
hGpib (hGpib)	Long	Returns the VISA handle. If not connected or after disconnection, 0 is returned. Programs that use NI-GPIB functions in Version 1.52 must be rewritten using VISA functions.
Err (Read Only)	Integer	Returns the ERR register value specific to a device
Stb (Read Only)	Integer	Returns a status register value
SRQUnmask	Integer	Sets a mask for each SRQ event
SRQInterval	Integer	Sets a regular monitoring interval for SRQ Turns off monitoring operation when "0" (unit: ms) is set. Default: 1,000 ms; maximum: 32,767 ms
SRQEnable	Boolean	Makes settings to specify whether to perform regular monitoring of SRQ In the case of TRUE, serial polling is performed at a preset interval (SRQ interval) during connection. When the SRQ bit is active, an SRQ event is generated.
Alloutput	Integer, Boolean	Turns on/off the outputs from each channel for all power-supply units connected
Modules (Index As VARIANT)	Object	Returns a Modules Collection object or a Module

Method	Return value/ Argument	Description
Connect	Return value: None Argument: DevName As String	Specifies a device name and makes connections to the device
Disconnect	Return value: None Argument: None	Disconnects the device
Initialize	Return value: None Argument: None	Makes initial settings for a device
SetString	Return value: None Argument: strWrt As String	Writes general character strings This method is applied to use a function not supported by the driver.
GetString	Return value: String Argument: None	Reads general character strings This method is applied to read information not supported by the driver.
ClearRegs	Return value: None Argument: None	Initializes the register function and register state of a device
Shutdown	Return value: None Argument: None	Turns off the outputs from all power-supply units, and sets the reference to "0"

Event	Argument	Description
SRQ	Byval stb As Integer	Valid when the SRQ Enabled property is set to TRUE. Called when a service request is issued by the device.

Modules Collection Object

Property	Type	Description
Count	Long	Returns the number of Module objects in the collection
Item	Return value: Module Argument: Long	Returns a specified Module object

Module Object

Property	Type	Description
Supplies (Index As VARIANT)	Object	Returns a Supplies Collection object or Supply object
ModuleId (Read Only)	Integer	Returns a Module ID
ModuleName (Read Only)	String	Returns a Module name

Supplies Collection Object

Property	Type	Description
Count	Long	Returns the number of Supply objects in the collection
Item	Return value: Supply Argument: Long	Returns a specified Supply object

Supply Object

Property	Type	Description
ModelName (Read Only)	String	Returns the model name of the power supply connected
ModelId (Read Only)	Integer	Returns the model ID of the power supply connected
MaxVoltage (Read Only)	Double	Returns the maximum voltage value of the power supply connected
MaxCurrent (Read Only)	Double	Returns the maximum current value of the power supply connected
Voltage	Double	Sets a voltage value for the power supply connected
Current	Double	Sets a current value for the power supply connected
DigitalOut	Integer	Makes digital settings on the power supply connected
Remote	Boolean	Makes remote/local settings on the power supply connected
Output	Boolean	Turns ON/OFF outputs from the power supply connected
Load	Boolean	Turns ON/OFF a PLZ load connected
Ovp	Double	Sets an OVP value on the power supply connected
MonVoltage (Read Only)	Double	Returns the output-voltage value of the power supply connected
MonCurrent (Read Only)	Double	Returns the output-current value of the power supply connected
DigitalIn (Read Only)	Integer	Returns the output digital value of the power supply connected
Status (Read Only)	Integer	Returns status information on the power supply (including alarm information)
Fau (Read Only)	Integer	Returns fault register information
Funmask	Integer	Sets a mask bit for the fault register

Method	Return value/ Argument	Description
PowerOff	Return value: None Argument: None	Turns OFF the AC input to the power supply connected