Using ActiveX Instrument Driver Objects

Dec 2000　2nd Edition

Original Japanese Issue by :　Hiroshi Yamaguchi (Software Engineering Group)

English Edition Translation by:　Makoto Kondo (Software Engineering Group)

KIKUSUI ELECTRONICS CORP.

Getting Started

1. SOFTWARE LICENSE AGREEMENT

Each item of the following license agreement is exactly the same as shown in the Kikusui web's download page. You must have agreed the license agreement if you are reading this guidebook downloaded at the site. If you decline the agreement, please delete this guidebook.

1. Grant of License

KIKUSUI Electronics Corp. hereinafter referred to as ("KIKUSUI") grants to you the following rights concerning this software, which can be downloaded on the condition that you agree to the terms of this Agreement and related written materials (collectively referred to as "the SOFTWARE"):

(a) To use of the SOFTWARE for the purpose of using our products that correspond to the SOFTWARE

(b) To make copies of the SOFTWARE for use on one or more computers

2. Additional License

Files of various types that are created as a result of using the SOFTWARE in accordance with the specified purposes are considered as your copyrighted works.

3. Copyright

The SOFTWARE and its copies are owned by KIKUSUI or any party authorized by KIKUSUI, and are protected by Japanese Copyright Act and international treaty provisions. Any copies that you are permitted to make pursuant to this Agreement must contain the same copyright and other proprietary notices that are affixed to the downloaded SOFTWARE.

4. Restrictions

(a) When the SOFTWARE is supplied with source code, you shall not distribute the SOFTWARE which you modified to a third party.

(b) When the SOFTWARE is supplied with binary (compiled) object, you shall not reverse-engineer the SOFTWARE.

5. Limited Warranty

KIKUSUI does not guarantee that the SOFTWARE is suitable for a particular purpose, or that the SOFTWARE is free of defects, or any other matters relating to the SOFTWARE.

6. Exclusion of Liability

Under no circumstances shall KIKUSUI be liable for any damages whatsoever (including, but not limited to, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the SOFTWARE.

7. Termination

KIKUSUI reserves the right to terminate this Agreement if you fail to satisfy any of its terms and conditions, in which case you will not be permitted to use this SOFTWARE.

2. SUPPORT

Please be noted that this Server do not provide the User support. In case you have any question regarding to these software, please contact at our Service Dept.(service@kikusui.co.jp). And please be sure to read the document file after decompressing the file.

3. WARRANTY

Downloading and Installation of the driver shall be executed under the user's responsibility.

The contents of driver may be subject to change or to be modified for the improvement without notice.

4. COPYRIGHT HOLDER

The copyright of each driver belong to Kikusui Electronics Corp

5. ABOUT THIS GUIDEBOOK

This document will guide you through how to start using Visual Basic (ActiveX) instrument drivers downloaded at the Kikusui web download site with sample codes. It is recommended to read if you are in maze having no ideas after having drivers downloaded. This is a quick start guide.

The guidebook describes some example codes for use with Visual Basic 6.0, Excel 97, Visual C++ 6.0, and Delphi 5.0 in separate chapters. So you only need to read the chapter that matches with your development tool. Therefore you might feel there are some duplicate explanations if you strictly read every chapter.

Hereafter this guidebook calls Visual Basic (ActiveX) drivers simply "Instrument Drivers."

6. TRADEMARKS

Borland and Delphi are trademarks or registered trademarks of Inprise Corporation.

IBM is a registered trademark of International Business Machines Corporation.

Microsoft, Windows, Visual Basic, Visual C++, ActiveX are trademarks or registered trademarks of Microsoft Corporation in the U.S. or other countries.

National Instruments, NI-488.2, NI-VISA are trademarks or registered trademarks of National Instruments Corporation.

Other vendor names and product names are trademarks or registered trademarks of corresponding vendor.

Contents

Chapter 1- Overview

1-1 What Is Instrument Driver

Instrument drivers that can be downloaded at Kikusui web site are convenient development kits for writing application programs that control Kikusui products through GPIB or RS-232C.

If you write a GPIB-control app software in Visual Basic without instrument drivers, you will need link National Instruments gpib-32.dll looking up NI-GPIB API and command table of the Kikusui product. But using an instrument driver will allow you to put the control to your VB form as like as standard buttons or listboxes, and you can write easy-to-read codes rapidly.

The instrument drivers are built based on the COM (Component Object Model) technology. Therefore you can use them with various kinds of the language tools such as Visual Basic, Microsoft Office, and Delphi, which are all compatible with ActiveX feature.

1-2 Tools

This guidebook will show you concrete examples using Visual Basic 6.0, Excel 97, Visual C++ 6.0, and Delphi 5.0.

If you use other tools:

Any 32bit tools that support ActiveX (such as Visual Basic 4.0 / 5.0, C++ Builder, etc...) can also be used.

1-3 Operational Environment

You need VISA (Virtual Instrument Software Architecture) library ver2.0 or later for both GPIB and RS-232C operations, since the instrument drivers use VISA library. We did confirmation with NI-VISA but not with HP-VISA or other versions. VISA library may be included with your GPIB/HP-IB board or with LabVIEW CD-ROM. Otherwise you can download the NI-VISA2.0 at the following web site:

http://www.ni.com/visa/

GPIB users:

- Personal computer running Windows 9x/NT4/2000
- National Instruments NI-488.2M-compatible board, or Agilent Technologies HP-IB board
- GPIB cable

RS-232C users:

- Personal computer running Windows 9x/NT4/2000
- RS-232C Null-Modem (cross) cable

The following instrument driver versions are not compatible with VISA library. Please download the latest ones.

PCR-L/W series	Ver 2.70 or earlier
PLZ-3W/3WH series	Ver 1.00 or earlier
PIA3200	Ver 1.10 or earlier
TOS9000	Ver 1.01 or earlier
KJM6755(A)	Ver 1.00 or earlier

You need a different way when you import PCR-L/W series instrument driver Ver2.70 to your development tools. That driver comes with a guidebook detailer than this book. Please refer it.

PCR-L/W series driver Ver2.70 works only with National Instruments NI-488.2M-compatible boards.

Chapter 2- Setup

2-1 Interface Cards

If your instrument does not have GPIB or RS-232C interface as a standard equipment, you need install optional interface card to your instrument. Refer to the instruction manual about applicable option cards. After attaching the interface card, configure GPIB address or RS232 communication parameters. Mind that some RS-232C instruments require different communication parameters or cables.

2-2 GPIB Board

If you use GPIB, you need attach an NI-488.2M or compatible board (National Instruments) or an HP-IB board (Agilent Technologies) to your PC. Your PC also requires VISA library.

2-3 Connecting Interface Cable

When the interface cards on both instrument and PC are ready, connect an IEEE488 or RS-232C cable between them.

2-4 Installing Instrument Driver

The downloaded instrument driver (xxx.EXE) is a self-extract archive that works as a SETUP program. Run the program by typing the executable name or execute it from Explorer. As the dialogue box shown in Figure 2-1 appears, click the **Setup** button. After that you only need to follow the on-screen instruction.

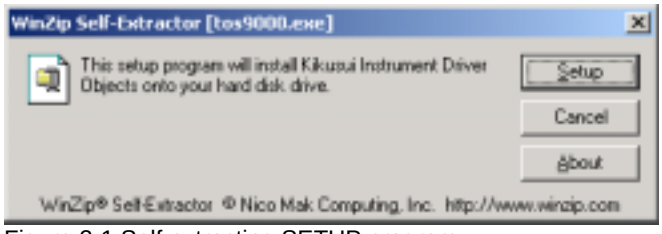


Figure 2-1 Self-extracting SETUP program

After the screen shows "Setup Complete !" message, click the **Close** button to finish the installation. The SETUP program copies necessary files to **C:\Program Files\Common Files\Kikusui Shared** folder and registers the driver to the registry database.

2-5 Tools

The guidebook assumes that you already installed your development tool such as Visual Basic 6.0 or Excel 97. The structure of this guidebook only requires you need to read necessary chapters concerning to your tool.

Chapter 3	Visual Basic 6.0	
Chapter 4	Excel 97	
Chapter 5	Visual C++ 6.0	(MFC Dialogue-based app, Late Binding)
Chapter 6	Visual C++ 6.0	(Early Binding)
Chapter 7	Delphi 5.0	

Chapter 3- Visual Basic 6.0

3-1 Adding A Component

Launch Visual Basic 6.0 and then select **Standard EXE** on the **New Project** dialogue. Choose **Project** | **Components** menu to open the **Components** dialogue shown in Figure 3-1. After installing the instrument driver, **Kikusui xxxx Driver Vx.x** must been seen in the component list. Now check the driver module you want to use. After closing the dialog with the **OK** button, the **Toolbox** window will show the driver component on it (Figure 3-2).

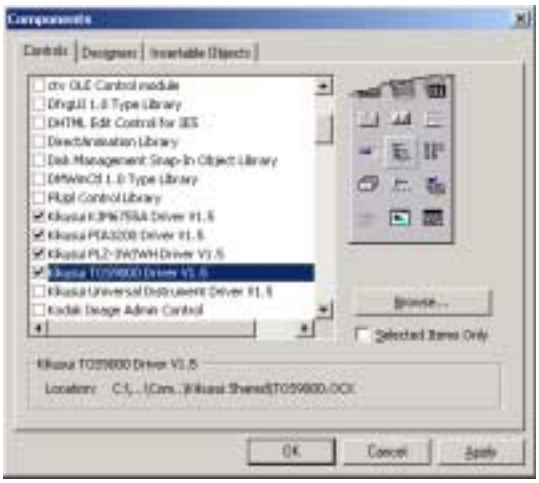


Figure 3-1 Visual Basic 6.0 Components dialogue

3-2 Putting Control Onto VB Form

Put **Tos9000** and **CommandButton** controls from the **Toolbox** onto your VB form. Although the given default name Tos90001 is nice, now rename it to simply *tos* on the **Properties** window.

> Note:
>
> Although the guidebook examples TOS9000 instrument driver here, it will be easy to replace with other instrument drivers.

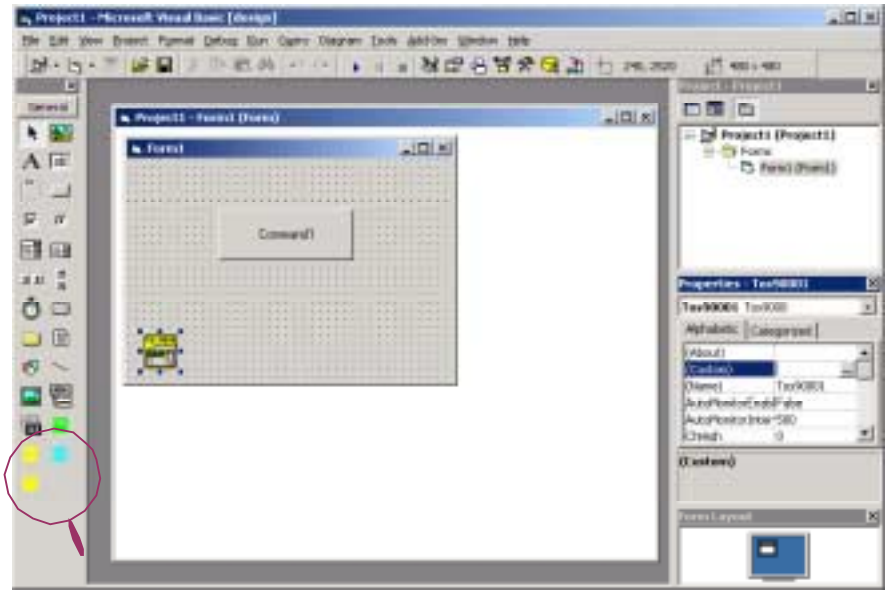


Figure 3-2 Visual Basic 6.0 Toolbox and Form

3-3 The First Code - Connect

Double-click the **Command1** button and you can write event handling code there. Type **tos.** and then you will see object members of Tos9000 object with a help of Auto List Member feature on Visual Basic (Figure 3-3). Highlight the **Connect** method and then press **CTRL** + **ENTER**. The word **Connect** will be automatically entered.

> Note:
>
> With **Tools**| **Options** menu and then **Editor** tab on the Visual Basic, you can enable/disable the Auto List Members feature.

Furthermore press the **SPACEBAR** then a guide Connect (DevName As String) will pop up. As the VB guides, type a device name **"GPIB::1"** as string type. The DevName "GPIB::1" means GPIB address 1. If using an RS-232C port, type **"ASRL1"**.

```
Private Sub Command1_Click()
   tos.Connect "GPIB::1"
End Sub
```

The **Connect** method actually connects the instrument as you think. You have to call this method first on your program codes. When the above one-line program is executed by clicking the **Command1** button, your TOS9000 must go to REMOTE state. In contrast the disconnection method is **Disconnect**.

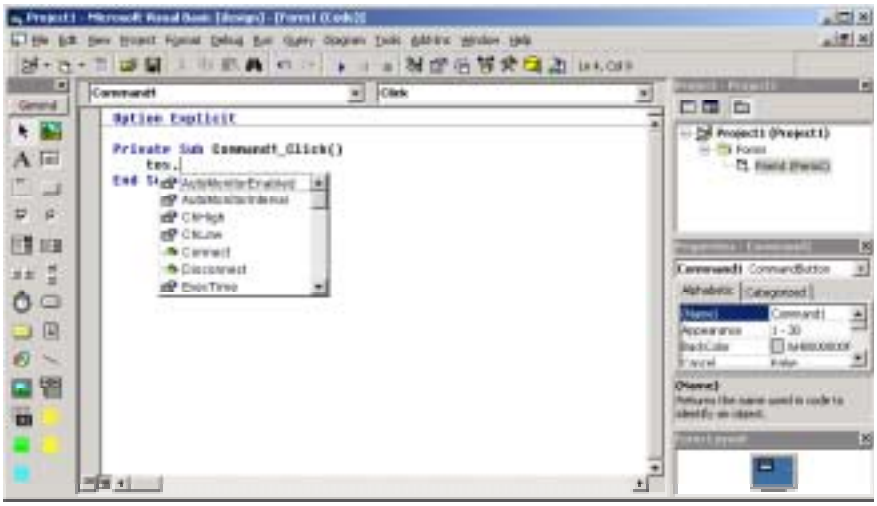


Figure 3-3 Auto List Members feature

It is also one of good ways that you Connect the instrument at form's Load operation and Disconnect at the Unload operation. Additionally, it will be also good to replace the button caption with **Connect** or **Disconnect**.

```
Private Sub Form_Load()
   tos.Connect "GPIB::1"
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
   tos.Disconnect
End Sub
```

If you succeeded until Connect operation, you will have understood basic concept. You can write your own application by referencing the online help.

3-4 Need A Help?

To look up the online help of instrument drivers, chose **View** | **Object Browser** menu to show the **Object Browser**, then select "**TOS9000LibCtl**" from the Project/Library

combobox. Highlight a member of the Tos9000 object and then click the **Help** button. Now you will see the online help for the TOS9000 Driver.

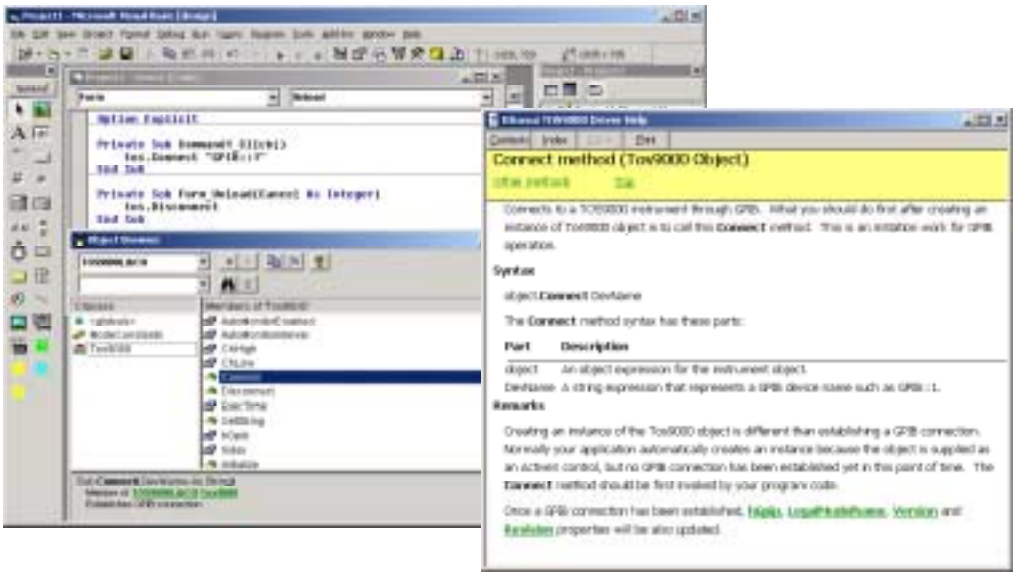


Figure 3-4 Referencing Help on Object Browser

3-5 Property Pages

Since the TOS9000 instrument driver has property pages for design-time setting, your program can easily control the instrument only with the **Start** and **Stop** actions. The **Auto Monitor** tab page is for setting event handling properties.

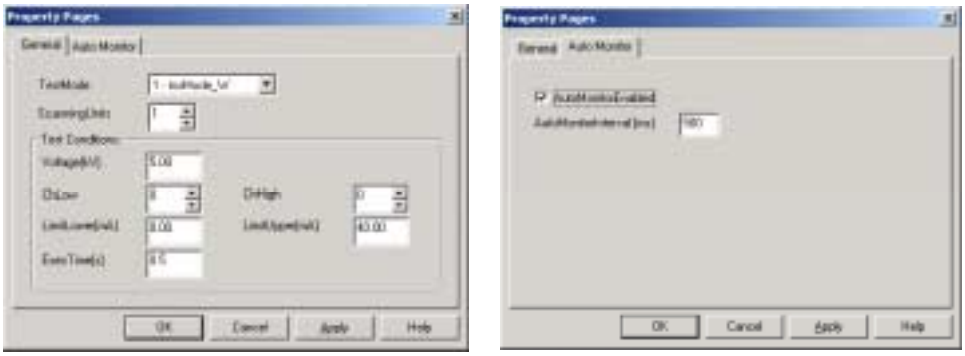


Figure 3-5 Property Pages for TOS9000 Driver

3-6 Event Handling

Double-click the TOS9000 icon on the form, or select **tos** on the Object combobox and **TestingW** on the Procedure combobox (Figure 3-6). Then a sub procedure in which you can write event-handling code for W-test will be generated.

```
Private Sub tos_TestingW(ByVal Status As Integer, ByVal Voltage As Double, ByVal Current As Double)

End Sub
```

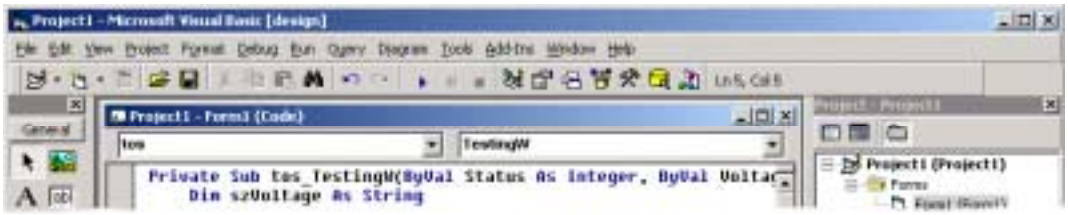


Figure 3-6 Object and procedure combo-boxes on Visual Basic

To enable the event the **AutoMonitorEnabled** property must be set to True. Also set **AutoMonitorInterval** to an appropriate value [ms]. We recommend 500ms. The example code shown in the section 3-7 performs a W-test and updates the monitored voltage and current display on the label control at every 500ms.

3-7 Example Codes

This example program sets Test Mode and Test Voltage with the **Setting** button, starts with the **Start** button, and stops with the **Stop** button. It connects the instrument when the Form is being loaded and disconnects when unloaded. If the connection is failed, then an error message will appear. Clicking the **Monitor** button also retrieves test voltage with the **MonVoltage** property and displays it on the label control. Although this example is similar to the "event-handling" example, the **Monitor** button is valid even if the test is stopped.

```
Private Sub Form_Load()             ' [ Form's loading process ]
   On Error GoTo GPIB_EXP           ' declares exception handler (GPIB_EXP)
   tos.Connect "GPIB::1"            ' connects instrument
   Exit Sub
GPIB_EXP:                           ' [ Exception handler ]
   MsgBox Err.Description           ' shows error message
   tos.Disconnect
   End                              ' halts the app
End Sub

Private Sub Form_Unload(Cancel As Integer)  ' [Form's unload process]
   tos.Disconnect                   ' disconnects instrument
End Sub

Private Sub Monitor_Click()         ' [ MONITOR button handler ]
   Label1.Caption = Format(tos.MonVoltage, "0.00kV")
                                    ' shows monitored voltage on Label1
End Sub

Private Sub Setting_Click()         ' [ SETTING button handler ]
   tos.TestMode = tosMode_W         ' W-test mode
   tos.ScanningUnits = 0            ' no scanning unit
   tos.Voltage = 5                  ' test voltage 5kV
   tos.LimitLower = 0               ' lower limit 0mA
   tos.LimitUpper = 40              ' upper limit 40mA
   tos.ExecTime = 10                ' test time 10sec
End Sub
Private Sub Start_Click()           ' [ START button handler ]
   tos.Start                        ' starts test
End Sub

Private Sub Stop_Click()            ' [ STOP button handler ]
   tos.Stop                         ' stops the test
End Sub

' [event handler for W-test ]
Private Sub tos_TestingW(ByVal Status As Integer, ByVal Voltage As Double,
ByVal Current As Double)
   Dim szVoltage As String
   Dim szCurrent As String

   If Status = 2 Then
       szVoltage = Format(Voltage, "0.00kV") ' formats volt value
       szCurrent = Format(Current, "0.00mA") ' formats ampere value

       ' shows volt & ampere on Label2 as a two-row display
       Label2.Caption = szVoltage + Chr(&HD) + Chr(&HA) + szCurrent
   End If
End Sub
```

Figure 3-7 Running Example Program

3-8 SetString And GetString Methods

The SetString method allows you to send a command as is. This is just like ibwrt/ibrd or PRINT@1;/INPUT@1; statements.

```
Private Sub Monitor_Click()
   tos.SetString "VDATA?"
   Label1.Caption = tos.GetString
End Sub
```

Chapter 4- Excel 97

4-1 Adding User Form and Control

Launch Excel 97, and then activate the Visual Basic toolbar by selecting **View** | **Toolbars** | **Visual Basic** menu. It is also good to embed the toolbar into the menu bar. On the Visual Basic toolbar, click the button that shows Visual Basic Editor tooltip. Now you will see a window captioned "Microsoft Visual Basic – Book1." (Hereafter we call it Excel VBA or simply VBA.) Then chose **Insert** | **UserForm** menu on the VBA to add a new form.

Choose **Tools** | **Additional Controls** menu, then the Additional Control dialogue appears as shown in Figure 4-1. If an instrument driver is installed, the list must have a **Kikusui xxxx Driver**. If you can find a target instrument on the list, check it and then click the **OK** button to close the dialogue. Now you will see a driver component on the Toolbox. (Figure 4-2)

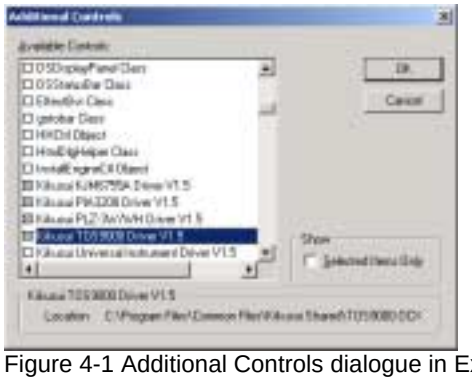
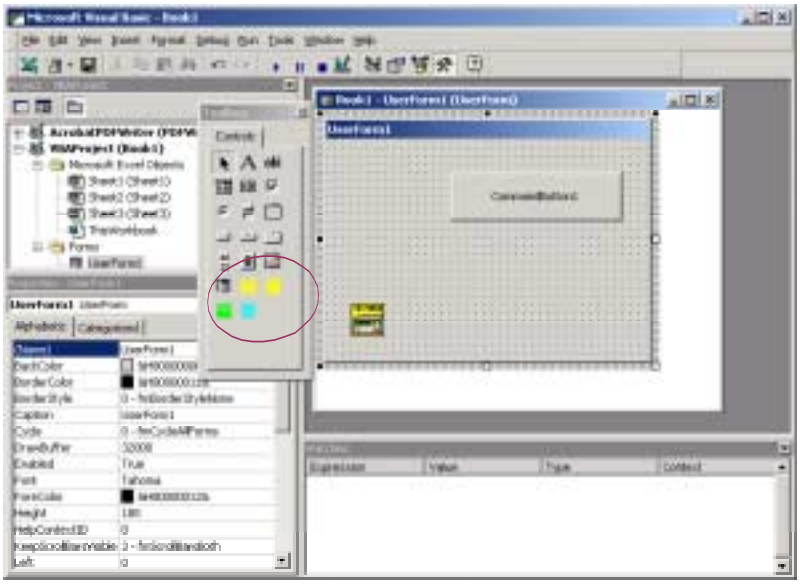


Figure 4-1 Additional Controls dialogue in Excel VBA

4-2 Putting Control

This example assumes that you put a button onto the form and add program codes, though it is also possible to put a button onto an Excel worksheet.

Figure 4-2 Excel VBA Toolbox and User Form



As shown in the Figure 4-2, put **CommandButton** and **Tos9000** objects on the UserForm from the Toolbox. Now change the default object name Tos90001 to simply **tos**, though the default name is not too bad.

Note:

Although the guidebook examples TOS9000 instrument driver here, it will be easy to replace with other instrument drivers.

4-3 The First Code - Connect

Double-click the **CommandButton1** button and you can write event handling code there. Type **tos.** and then you will see object members of Tos9000 object with a help of Auto List Members feature of Visual Basic (Figure 4-3). Highlight the Connect method and then press **CTRL** + **ENTER**. The word **Connect** will be automatically entered.

Note:

With **Tools**| **Options** menu and then **Editor** tab on the Visual Basic, you can enable/disable the Auto List Members feature.

Furthermore press the **SPACEBAR** then a guide Connect (DevName As String) will appear. As the VB guides, type a device name **"GPIB::1"** as string type. The DevName "GPIB::1" means GPIB address 1. If using an RS-232C port, type **"ASRL1"**.

```
Private Sub CommandButton1_Click()
    tos.Connect "GPIB::1"
End Sub
```

The **Connect** method actually connects the instrument as you think. You have to call this method first on your program codes. When the above one-line program is executed by clicking the **CommandButton1** button, your TOS9000 must goes to REMOTE state. In contrast the disconnection method is **Disconnect**.

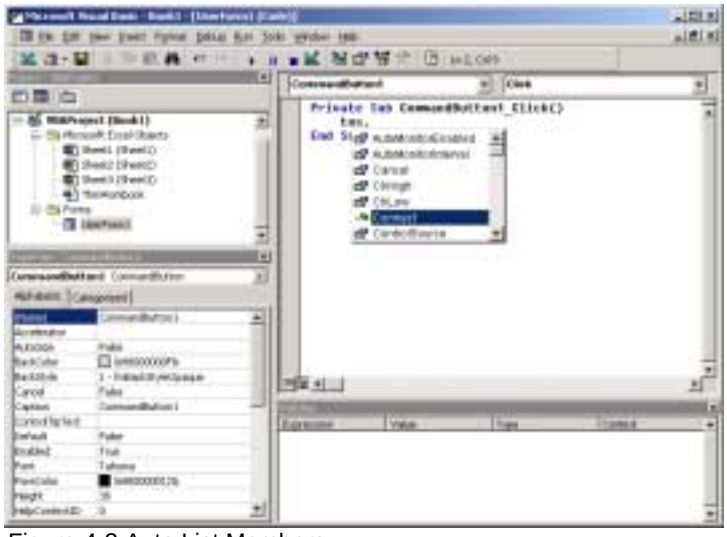


Figure 4-3 Auto List Members

Therefore, it is one of good ways that you Connect the instrument at form's Load operation and Disconnect at the Unload operation. Additionally, it will be also good to replace the button caption with **Connect** or **Disconnect**.

```
Private Sub UserForm_Initialize()
    tos.Connect "GPIB::1"
End Sub
```

```
Private Sub UserForm_Terminate()
    tos.Disconnect
End Sub
```

If you succeeded until Connect operation, you will have understood basic concept. You can write your own application by referencing the online help.

4-4 Need A Help?

To look up the online help of instrument drivers, chose **View** | **Object Browser** menu to show the **Object Browser**, then select "**TOS9000Lib**" from the Project/Library combobox. Highlight a member of the Tos9000 object and then click the **Help** button. Now you will see the online help for the TOS9000 Driver.

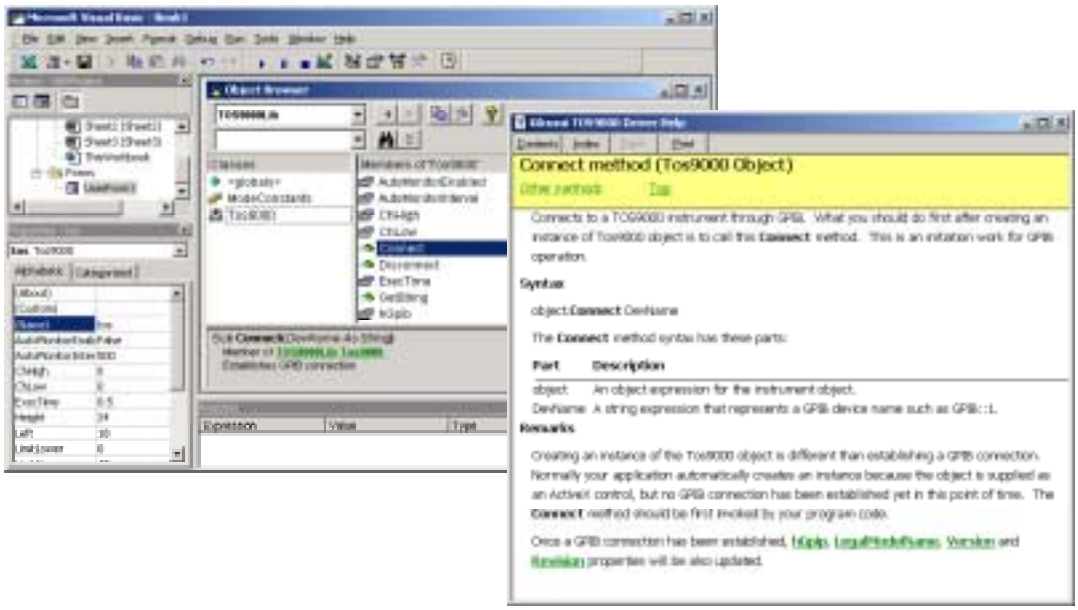


Figure 4-4 Referencing on-line help with Object Browser

4-5 Property Pages

Since the TOS9000 instrument driver has property pages for design-time setting, your program can easily control the instrument only with the **Start** and **Stop** actions. The **Auto Monitor** tab page is for setting event handling properties.

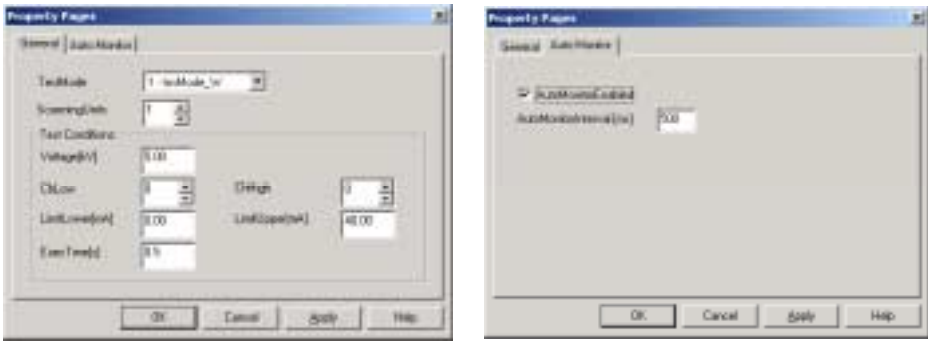


Figure 4-5 Property Pages for TOS9000 Driver

4-6 Event Handling

Double-click the TOS9000 icon on the form, or select **tos** on the Object combobox and **TestingW** on the Procedure combobox (Figure 4-6). Then a sub procedure in which you can write the event-handling for W-test will be generated.

```
Private Sub tos_TestingW(ByVal Status As Integer, ByVal Voltage As Double, ByVal Current As Double)

End Sub
```

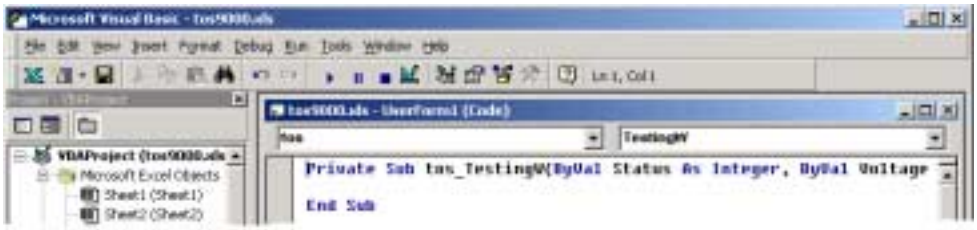


Figure 4-6 Object and procedure combo-boxes on VBA

To enable the event the **AutoMonitorEnabled** property must be set to True. Also set **AutoMonitorInterval** to an appropriate value [ms]. We recommend 500ms. The example code shown in the section 4-7 performs a W-test and updates the on-test voltage and current display on the label control at every 500ms.

4-7 Example Codes

To have an Excel-specific feature, you write here an example that sets up test conditions looking up the worksheet. The example locates option-buttons that can select each of A/B/C columns on the worksheet which describe test conditions. Clicking the **Start** button will start a test with given test conditions. Leakage current and measured resistance will be shown on the Label during withstanding and insulation test respectively.

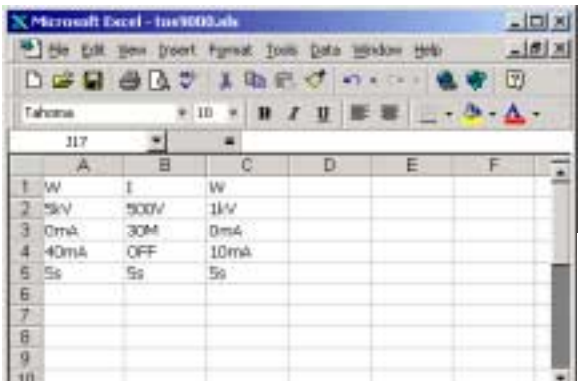


Figure 4-7 Excel worksheet

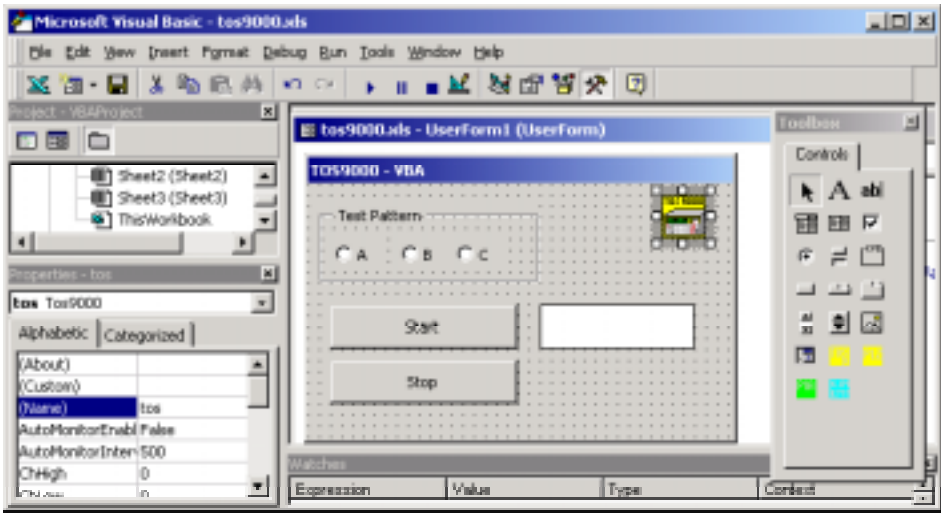


Figure 4-8 Designing UserForm

```
Private Sub CommandButtonStart_Click()       ' [START button handler]
   Dim szTestPat As String
   Dim szCmd As String
   Dim nSts As Integer

   If OptionButton1.Value = True Then
       szTestPat = "A"                       ' tests with A condition
   ElseIf OptionButton2.Value = True Then
       szTestPat = "B"                       ' tests with B condition
   Else
       szTestPat = "C"                       ' tests with C condition
   End If

   ' Check the first row text on Excel worksheet
   If Worksheets("Sheet1").Range(szTestPat + "1") = "W" Then
       ' W for Withstanding Test
       tos.TestMode     = tosMode_W
       ' References 2nd~5th rows to set voltage/lower/upper/time
       tos.Voltage      = Val(Worksheets("Sheet1").Range(szTestPat + "2"))
       tos.LimitLower   = Val(Worksheets("Sheet1").Range(szTestPat + "3"))
       tos.LimitUpper   = Val(Worksheets("Sheet1").Range(szTestPat + "4"))
       tos.ExecTime     = Val(Worksheets("Sheet1").Range(szTestPat + "5"))

   ElseIf Worksheets("Sheet1").Range(szTestPat + "1") = "I" Then
       ' I for Insulation Test
       tos.TestMode = tosMode_I
       ' References 2nd~5th rows to set voltage/lower/upper/time
       tos.Voltage  = Val(Worksheets("Sheet1").Range(szTestPat + "2"))/
1000
       tos.LimitLower   = Val(Worksheets("Sheet1").Range(szTestPat + "3"))
       tos.LimitUpper   = Val(Worksheets("Sheet1").Range(szTestPat + "4"))
       tos.ExecTime     = Val(Worksheets("Sheet1").Range(szTestPat + "5"))
   Else
       Exit Sub
           End If
   tos.Start                                 ' starts test
End Sub

Private Sub CommandButtonStop_Click()        ' [STOP button handler ]
   tos.Stop
End Sub



' [event handler for I-test ]
Private Sub tos_TestingI(ByVal Status As Integer, ByVal Resistance As
Double)  Dim dMomR As Double

   dMonR = Resistance
   If dMonR > 99900 Then
       Label1.Caption = "OVER"               ' shows OVER if exceeding 99.9Gohm
   Else
       Label1.Caption = Str(dMonR) + "Mohm"' shows resistance value on Label1
   End If
End Sub

' [event handler for W-test ]
Private Sub tos_TestingW(ByVal Status As Integer, ByVal Voltage As Double,
ByVal Current As Double)
   Label1.Caption = Format(Current, "0.00mA")
End Sub
```

```
Private Sub UserForm_Initialize()              ' Form's initialization process
   On Error GoTo GPIB_EXP                      ' declares exp handler (GPIB_EXP)

   tos.Connect "GPIB::1"                       ' connects instrument
   Exit Sub

GPIB_EXP:
   MsgBox Err.Description                      ' shows err msg when exp raised
   tos.Disconnect
   End                                         ' halts the app
End Sub

Private Sub UserForm_Terminate()               ' Form's termination process
   tos.Disconnect                              ' disconnects the instrument
End Sub
```

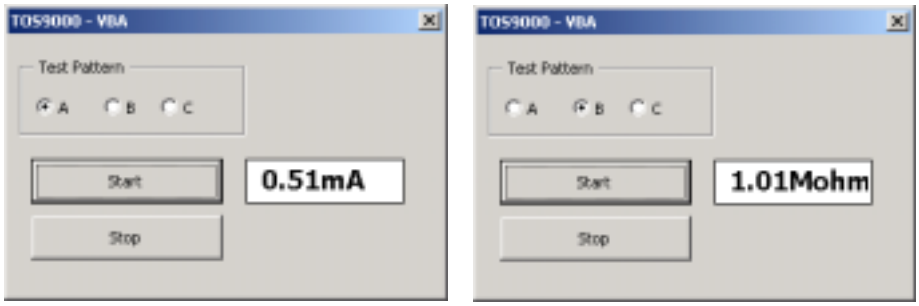


Figure 4-9 Running example

4-8 SetString And GetString Methods

The SetString method allows you to send a command as is. This is just like ibwrt/ibrd or PRINT@1;/INPUT@1; statements.

```
Private Sub CommandButtonMonitor_Click()
   tos.SetString "VDATA?"
   Label1.Caption = tos.GetString
End Sub
```

Chapter 5- Visual C++ 6.0 (Dialogue-based App)

5-1 Creating A Project

Choose **File** | **New** menu to create a dialogue-based app project. Then select the **Project** tab with **MFC AppWizard(exe)**. Enter **TOS9000** as the project name. (Figure 5-1)

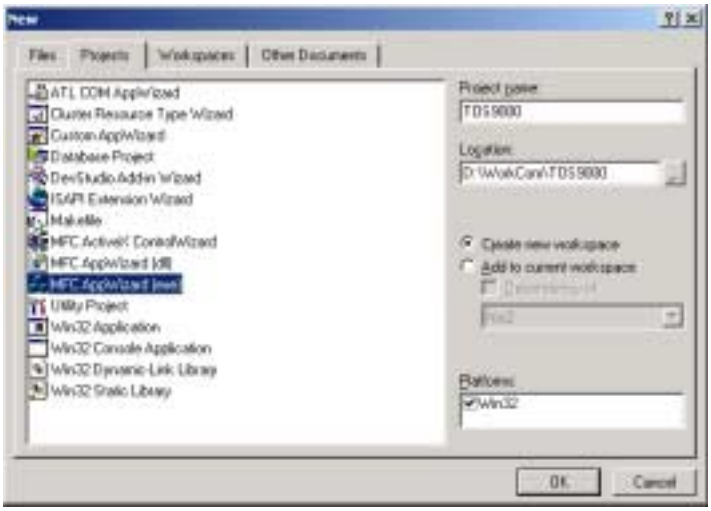
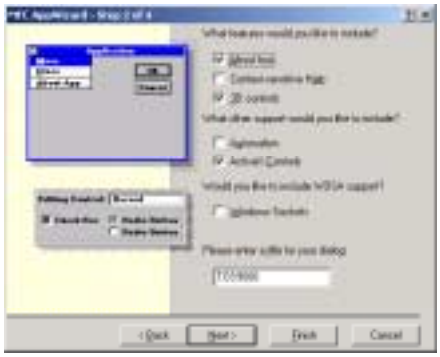


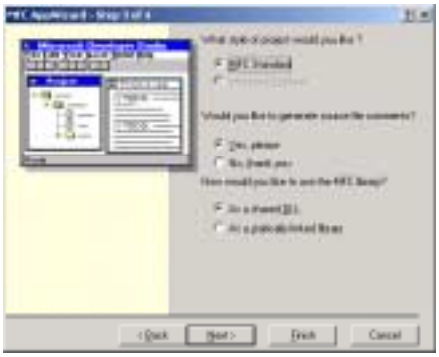
Figure 5-1 Visual C++ 6.0 "New" dialogue

In the MFC AppWizard - Step1, choose **Dialog based**. Steps 2 through 4 can be default (Figure 5-2).

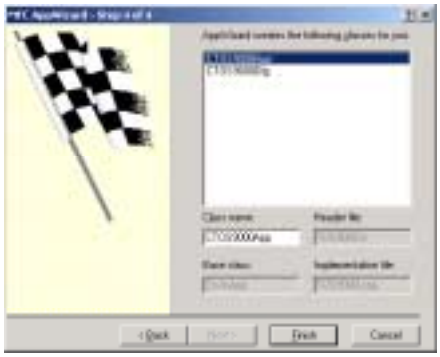
Step1/4



Step2/4



Step3/4



Step4/4

Figure 5-2 MFC AppWizard Step1 to 4

5-2 Adding Component

From the Visual C++ environment, choose **Project** | **Add To Project** | **Components and Controls** menu. The **Components and Controls Gallery** dialogue will appear. (Figure 5-3)

Double-click the **Registered ActiveX Controls** folder to show the ActiveX control list. If an instrument driver is installed, the list must display its name such as **Kikusui xxxx Driver**. Highlight a driver you want and then click the **Insert** button. As you close the dialogue, the driver component will be added onto the **Control** palette (Figure 5-4).

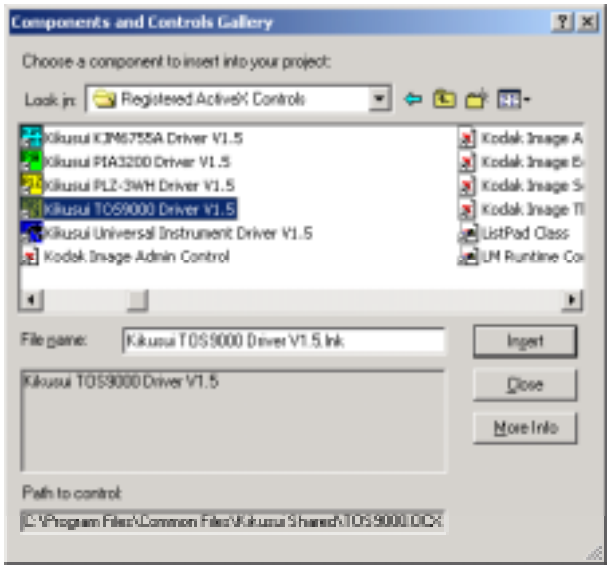


Figure 5-3 Components and Controls Gallery dialogue

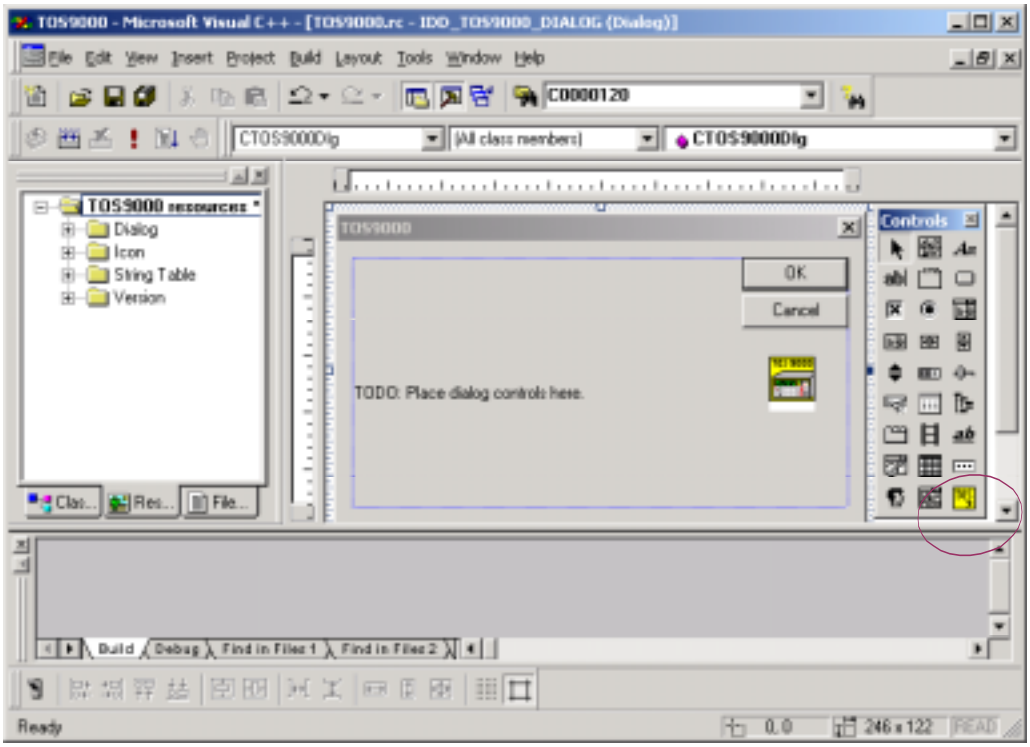


Figure 5-4 ActiveX controls for instrument driver

5-3 Locating Controls

As shown in Figure 5-4, select the Tos9000 object from the Control palette and put it onto the dialogue box.

Note:

Although the guidebook examples TOS9000 instrument driver here, it will be easy to replace with other instrument drivers.

Delete the default **OK** and **Cancel** buttons. Instead add newly 4 buttons and 2 edit controls. Also add some static controls that describes each control as needed. (Figure 5-5)

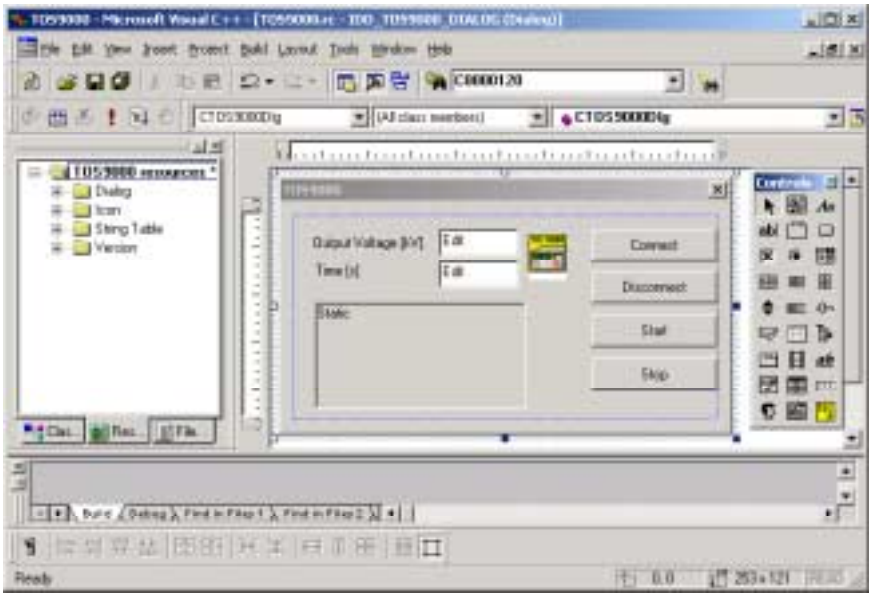


Figure 5-5 Designing dialogue box

Modify resource ID and caption for each of buttons with the property dialogue. (Figure 5-6)

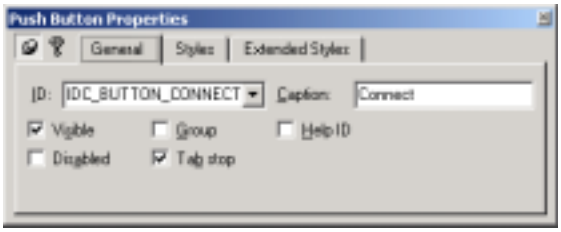


Figure 5-6 Control's Property

Set the resource ID for the Tos9000 to **IDC_TOS9000**. Also add a data member of the IDC_TOS9000 giving the name **m_tos9000**. (Figure 5-7)

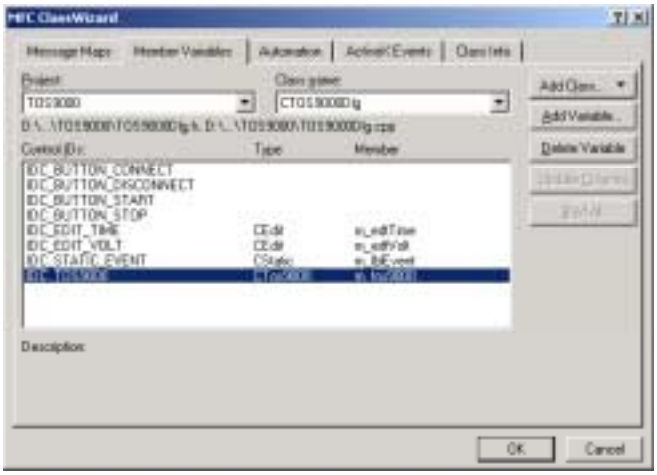


Figure 5-7 Adding member variables with Class Wizard

5-4 The First Code – Connect

Double-click the **Connect** button to add the button handler. In the handler, as you type **m_tos9000.** then Visual C++6.0's auto-completion feature will help you by showing available class members. Then just typing **co** will make you find the Connect method. Also do not forget add **try/catch** exception handling code.

```
void CTOS9000Dlg::OnButtonConnect()
{
    CString strVolt;

    try{
        m_tos9000.Connect("GPIB::1");
    }
    catch( COleDispatchException* pE){
    CHAR szMsg[ 64];
        pE->GetErrorMessage( szMsg, sizeof( szMsg), NULL);
        AfxMessageBox( szMsg);
        pE->Delete();
    }
}
```

If the connection has failed, an error message shown in (Figure 5-8) will appear.

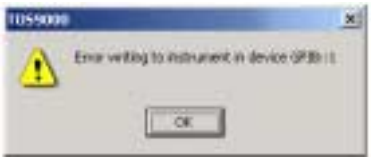


Figure 5-8 Error Message

5-5 Event Handling

Right-click the TOS9000 icon on the dialogue and select **Events**. Then the **New Windows Message and Event Handlers** dialogue will appear. Highlight **TestingW** and then click the **Add Handler** button (Figure 5-9). Now a procedure on which you can write event-handling code will be generated.

```
void CTOS9000Dlg::OnTestingWTos9000(short Status, double Voltage, double Current)
{
    // TODO: Add your control notification handler code here
}
```

Figure 5-9 Creating Event Handler

To enable the event the **AutoMonitorEnabled** property must be set to True. Also set **AutoMonitorInterval** to an appropriate value [ms]. We recommend 500ms. The example code shown in the section 5-6 performs a W-test and updates the on-test voltage and current display on the static text at every 500ms.

5-6 Example Code

In the same manner, add button handlers for **Disconnect**, **Start**, and **Stop** buttons.

```
void CTOS9000Dlg::OnButtonDisconnect()        //[ DISCONNECT button handler ]
{
    m_tos9000.Disconnect();                   //disconnects the instr
}

void CTOS9000Dlg::OnButtonStart()             //[ START button handler ]
{
    CString strVolt;
    CString strTime;

    try{
        m_edtVolt.GetWindowText( strVolt);        //retrieves editbox contents
        m_edtTime.GetWindowText( strTime);

        //Sets test mode 1=Withstanding, 2=Insulation, 3=LowOhm
        m_tos9000.SetTestMode( 1);
        m_tos9000.SetVoltage( atof( strVolt));    //sets test voltage
        m_tos9000.SetLimitLower( 0);              //sets lower limit
        m_tos9000.SetLimitUpper( 40);             //sets upper limit
        m_tos9000.SetExecTime( atof( strTime));   //sets test time
        m_tos9000.Start();                        //starts test
    }
    catch( COleDispatchException* pE){            //exception handling
        ' shows err msg
        CHAR szMsg[ 64];
        pE->GetErrorMessage( szMsg, sizeof( szMsg), NULL);
        AfxMessageBox( szMsg);
        pE->Delete();                             //deletes exception object
    }
}

void CTOS9000Dlg::OnButtonStop()              //[ STOP button handler ]
{
    m_tos9000.Stop();                         //stops the test
}

//[event handler for I-test ]
void CTOS9000Dlg::OnTestingWTos9000(short Status, double Voltage, double
Current)
{
    CString  strEvent;
    //shows volt & ampere on the static text
    strEvent.Format( "%3.2fkV\n%3.2fmA", Voltage, Current);
    m_lblEvent.SetWindowText( strEvent);
}
```

5-7 Build And Run

Figure 5-10 Build and run

5-8 Need A Help?

For detail information about methods, properties, and events, access the on-line help for each of instrument drivers.

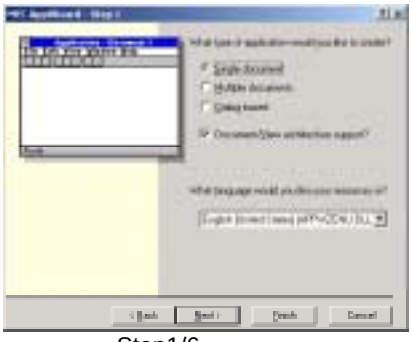
C:\Program Files\Common Files\Kikusui Shared\xxxx.hlp

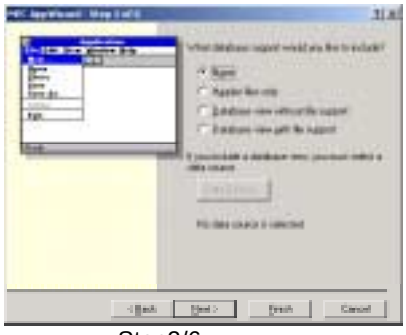
Chapter 6- Visual C++ 6.0 (With Early Binding)

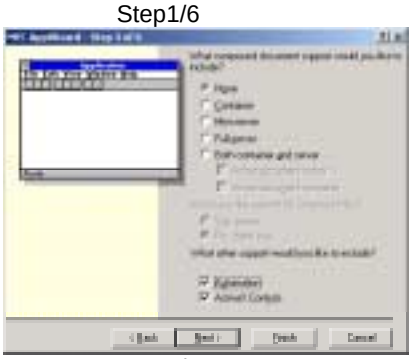
6-1 Creating Project

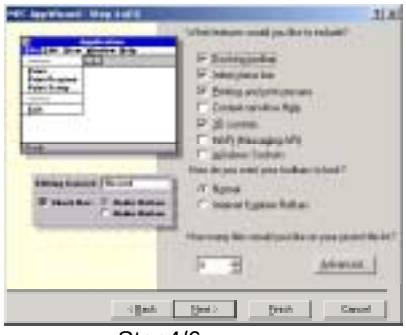
This chapter describes how to use instrument drivers with MFC Doc/View architecture app using early binding. The example is SDI app but MDI app is also okay. Instead of putting controls on a dialogue, the example adds a menu item that handles device controlling.

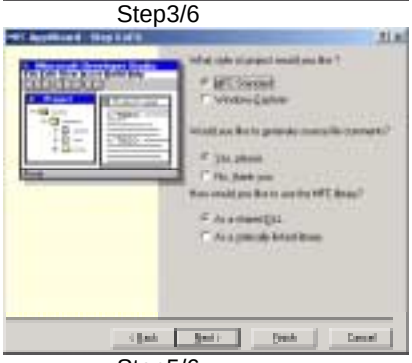
First Create an MFC AppWizard(exe) project named **Tos9000SDI**. The app type is SDI (Figure 6-1).


Step1/6


Step2/6


Step3/6


Step4/6


Step5/6


Step6/6

Figure 6-1 MFC AppWizard Step 1 to 6

6-2 Importing Type Library

Open **StdAfx.h** and then add the following **#import** statement after other includes:

#import "C:\Program Files\Common Files\Kikusui Shared\TOS9000.OCX" named_guids no_namespace

```
#include <afxcmn.h>          ' MFC support for Windows Common Controls

#import "C:\Program Files\Common Files\Kikusui Shared\TOS9000.OCX"
        named_guids no_namespace

#endif ' _AFX_NO_AFXCMN_SUPPORT
```

Note:

If there is a possibility that the instrument driver class name conflicts with others, delete the last keyword **no_namespace**. For instance, PIA3200 and PIA4800 drivers both have the same interface name **ISupply**, which will conflict each other. To solve this problem, you need avoid using the **no_namespace** keyword. In this situation, however, your app code will have to write explicit name-space designation for all the CLSID, interface names (such as ITos9000), and interface ID (such as IID_ITos9000).

Open the **InitInstance** member function on the **CTos9000SDIApp** class to check if the **AfxOleInit()** call is contained. If not add the call on the member function. If you have checked the **Automation** option on the AppWizard phase, the call must have been generated. (Figure 6-1 Step3)

6-3 Adding Menu

Activate the **ResourceView** tab on the Workspace window and then double-click the **IDR_MAINFRAME** menu resource. Add **Instrument** menu at the right side of the existing View menu and add an item **Connect**. The resource ID should be **ID_INSTR_CONNECT**. From the ClassWizard, generate the event handler for this menu item by selecting COMMAND message in the Doc class then clicking the **Add Function** button. (Figure 6-2)

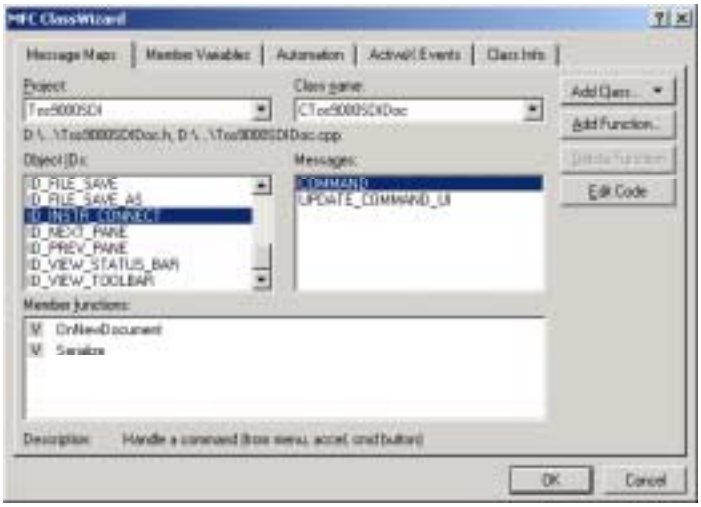


Figure 6-2 Adding Menu Handler with ClassWizard

6-4 Adding Data Members

Declare a data member **m_spTos** as **ITos9000Ptr** type in the Doc class header file. (Bold letters in the code below)

```
class CTos9000SDIDoc : public CDocument
{
protected: ' create from serialization only
    ITos9000Ptr m_spTos;

    CTos9000SDIDoc();
    DECLARE_DYNCREATE(CTos9000SDIDoc)
        :
```

Note:

Once you have built the program, you can see the Type Library header (extension **.tlh**) automatically generated by the compiler into the **Debug** folder having ITos9000 interface type which is also typedefed as a smart-pointer. Similarly, you can see the implementation file (**.tli**).

6-5 Constructor

The Doc class constructor has to have a generation code for the Tos9000 object instance. This allows Tos9000 object to be instantiated when the Doc object is created at the same time. (Bold letters in the code below)

```
CTos9000EBDoc::CTos9000EBDoc()
{
    ' TODO: add one-time construction code here
    HRESULT hr =::CoCreateInstance( CLSID_Tos9000, NULL,
            CLSCTX_INPROC_SERVER, IID_ITos9000, (void**)&m_spTos);
    ASSERT( SUCCEEDED(hr));

    EnableAutomation();
    AfxOleLockApp();
}
```

6-6 Handler For Connect Menu

Open the menu handler code **OnInstrConnect** generated by the ClassWizard and then add the following code. If the declaration of the smart-pointer went well, typing **m_spots->** will invoke the auto-completion feature. Also do not forget the exception handler. (Bold letters in the code below)

```
void CTos9000SDIDoc::OnInstrConnect()
{
    ' TODO: Add your command handler code here
    try {
        m_spTos->Connect(L"GPIB::1");
    }
    catch( _com_error e) {
        _bstr_t strDesc = e.Description();
        AfxMessageBox( strDesc);
    }
}
```

Note:

When you use early bindings in Visual C++, every string passed/returned to/from the object must be UNICODE. The **_bstr_t** is a convenient class that converts UNICODE and ANSI strings automatically.

6-7 Build and Run

Now build the executable and run it. Clicking the **Instrument** | **Connect** menu will make your instrument REMOTE state.

Figurfe 6-3 Running SDI app

After connection has succeeded, you can imagine what you should do for remaining. For detail information about methods, properties, and events, access the on-line help for each of instrument drivers.

C:\Program Files\Common Files\Kikusui Shared\xxxx.hlp

Chapter 7- Delphi 5.0

Note:

Although this chapter introduces how to use instrument drivers for use with Delphi5, you can use Delphi3 and Delphi4 in the same manner.

7-1 Importing Control

As you choose **Component** | **Import ActiveX Library** menu, you will see a dialogue shown in the Figure 7-1. If an instrument driver is installed, the list of registered controls must show the driver name such as "Kikusui xxxx Driver Vxx." Highlight the driver item and then click the **Install** button.

Figure 7-1 Installing ActiveX Control

In the **Install** dialogue, select **Into Existing Package** tab and then click the **OK** button.

Figure 7-2 Adding to existing package

Figure 7-3 Recompiling and registering component

Delphi now compiles its own component package and updates the component palette.

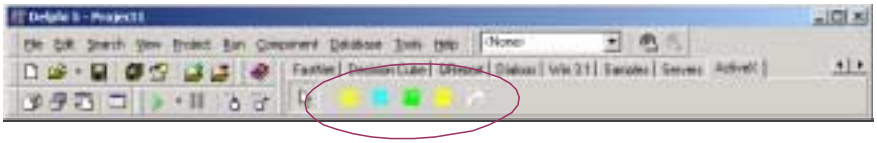


Figure 7-4 ActiveX page on Component Palette

The Component Palette will now add the instrument driver controls on the ActiveX page.

7-2 Putting Control onto Form

Put a **Tos9000** control (from ActiveX page on the Component Palette) and a **BitBtn** button (from Additional page) both onto the form. Then replace the button caption with **Connect** as well as renaming the Tos9000 control to simply **tos**.

Figure 7-5 Putting Controls

7-3 The First Code – Connect

Double-click the **Connect** button to add event handler code. Typing **tos.** will pop up available object members with a help of the Code Completion feature. Highlight the **Connect** method then press the **ENTER** key. After that type **('GPIB::1')** as a device name parameter. 'GPIB::1' means GPIB address 1. When using RS232, type **('ASRL1')** instead.

```
procedure TForm1.BtnConnectClick(Sender: TObject);
begin
   tos.Connect('GPIB::1');
end;
```

7-4 Exception Handling with try/except

In the except block, normally you need an exception object that is concerning to ActiveX/OLE objects. This is represented with **EOleException** object. To use this object type, your program code must add **ComObj** on the **Uses** section.

```
uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
   Dialogs, StdCtrls, Buttons, OleCtrls, TOS9000Lib_TLB, ComObj;
```

The following code shows a message box with a yellow exclamation mark and an **OK** button when an exception has been raised.

```
{ CONNECT button handler }
procedure TForm1.BtnConectClick(Sender: TObject);
begin
   try
       tos.Connect('GPIB::1');
   except                                                    {exception block}
       on e: EOleException do begin
            MessageDlg( e.Message, mtWarning, [mbOK], 0);   { shows err msg }
       end;
   end;
end;
```

Now let's try running the program. Clicking the **Connect** button will make the instrument REMOTE state. Also try turning the power off and then click the **Connect** button. This case Delphi debugger causes a runtime error and shows a message box (Figure 7-6).

Figure 7-6 Error Message

If you succeeded until Connect operation, you will understand basic concept. You can write your own application by referencing the online help.

C:\Program Files\Common Files\Kikusui Shared\xxxx.hlp

7-5 Event Handling

Chose **tos** on the Object Inspector, and then double-click the cell placed right-hand to the OnTestingW on the **Events** tab.

Figure 7-7 Generating event

Now an event procedure on which you can write handler code for W-test will appear.

```
procedure TForm1.tosTestingW(Sender: TObject; Status: Smallint;   Voltage, Current: Double);
begin

end;
```

To enable the event the **AutoMonitorEnabled** property must be set to True. Also set **AutoMonitorInterval** to an appropriate value [ms]. We recommend 500ms. The example code shown in the section 7-6 performs a W-test and updates the on-test voltage and current display on the label control at every 500ms.

7-6 Example Code

As shown in the Figure 7-8, the Form deign of this example has Edit controls for Test Voltage and Test Time, **Start** button, and **Stop** button. Other properties such as TestMode, LimitLower, or LimitUpper can be set at design time with the Object Inspector.

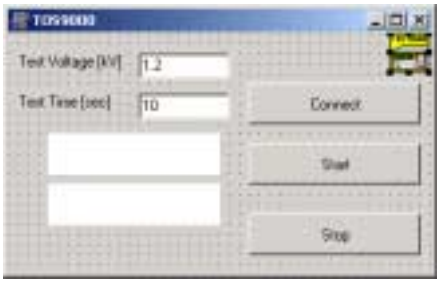


Figure 7-8 Designing Form

Clicking the **Start** button will start a test with given test conditions. The program displays output voltage and leakage current values at 500ms interval during the test running..

```
procedure TForm1.BtnConnectClick(Sender: TObject); {CONNECT button handler}
begin
   try
       tos.Connect('GPIB::1');                          { connects instrument }
   except
       on e: EOleException do begin                     { exception block }
            MessageDlg( e.Message, mtWarning, [mbOK], 0);
                                                        { shows err msg }
       end;
   end;
end;

procedure TForm1.BtnStartClick(Sender: TObject); { [START button handler]
begin
   tos.Voltage:= StrToFloat(EditVoltage.Text);    { sets test voltage  }
   tos.ExecTime:= StrToFloat(EditTime.Text);      { sets test time     }
   tos.Start();                                   { starts test        }
end;

procedure TForm1.BtnStopClick(Sender: TObject);   { [STOP button handler] }
begin
   tos.Stop();                                    { stops the test     }
end;

{ [Form closing ]  }
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
   tos.Disconnect();                              { disconnect the instr      }
end;

{ [event handler for W-test ] }
procedure TForm1.tosTestingW(Sender: TObject; Status: Smallint;
  Voltage, Current: Double);
begin
    { Formats monitored volt & ampere and shows them on Label1 }
   LabelMonitor1.Caption := Format('%2.2fkV', [Voltage]);
   LabelMonitor2.Caption := Format('%2.2fmA', [Current]);
end;
```

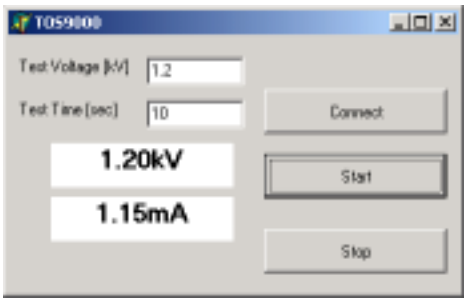


Figure 7-9 Running Example