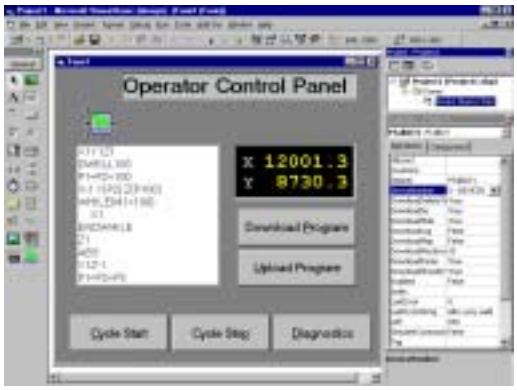
USER MANUAL

Accessory 9PTPRO



PTalkDTProOCX – ActiveX/Component/OCX

3Ax-OPTALK-xUxx Version 3.x

October 30, 2003

DELTA TAU
Data Systems, Inc.
NEW IDEAS IN MOTION ...

Single Source Machine Control *Power // Flexibility // Ease of Use*
21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are directly exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

INTRODUCTION **1**
What is PTalkDT? 1
What is an ActiveX Control? 1
What Can I use PTalkDT with? 1
What Can PTalkDT do for me? 1
What Built in Functions Does PTalkDT Have? 1
What You Will Need to use PTalkDT 2
How do I Get Support? 3

GETTING STARTED **5**
Installing PTalkDTPro 5
What Was Installed? *5*
Setting up Communications with PMAC 6
Plug & Play Device Installation *6*
Windows 98/ME Installation (Non Plug & Play Devices) *6*
Windows 2000 Installation (Non Plug & Play Devices) *9*
First Time PTalkDTPro Users 11
Serial Port Configuration *12*
Uninstalling PTalkDT OCX 13

HOW TO DESIGN WITH PTALKDT **15**
In Design Mode 15
Run Time Mode 16
Altering, Saving and Retrieving PTalkDT Settings at Run Time 18
Communication Settings *18*
General Settings *18*

YOUR FIRST VISUAL BASIC MMI WITH PTALKDT **19**
Overview 19
Instructions *19*

YOUR FIRST MICROSOFT VISUAL C++ MMI WITH PTALKDT **23**
Overview 23
Instructions *23*

PTALKDT REFERENCE **33**
Documentation Conventions 33
Overview 33
PTalkDT Properties 33
Enabled *33*
LastError *33*
LastErrorString *33*
Device Number *34*
DeviceNumber *34*
DownloadDeleteTemp *34*
DownloadDo *34*
DownloadHide *35*
DownloadLog *35*
DownloadMap *35*
DownloadMaxErrors *35*
DownloadParse *35*
DownloadShowErrors *36*
UploadAppend *36*
UploadHide *36*
UploadNoComments *36*
UploadShowProgress *36*
PTalkDT Methods 37

DPRAvailable() *37*
DownloadFile (file name) *37*
DPRDouble (LSB_word, MSB_word) *39*
DPRFixed (LSB_word, MSB_word) *39*
DPRDWordBit Set/Reset and BitSet Methods *40*
DPRGetDWord and DPRSetDWord Methods *40*
DPRGetFloat and DPRSetFloat Methods *41*
DPRGetMem and DPRSetMem Methods *41*
DPRGetWord and DPRSetWord Methods *42*
Flush ( ) *43*
GetControlResponse (Response, Control Char) *43*
GetLineAck (Response) *44*
GetLineCR (Response) *44*
GetResponse (Response, Command) *44*
IsLineWaiting ( ) *45*
LoadSettings ( ) *45*
LockPMAC ( ) *46*
ReleasePMAC( ) *46*
SaveSettings ( ) *46*
SelectDevice( ) *47*
SendChar (Character) *47*
SendLine (Command) *47*
ShowPropertyPage ( ) [OBSOLETE] *48*
UploadData (File Name, Command, Options, Expected Number of Lines) *48*
PTalkDT Events 49
OnError *49*
Trouble Shooting *49*
Dual Ported Ram Automatic Feature Example 50

DELTA TAU DRIVER CONFIGURATION 53
A Global View of the Driver 53
Device Configuration 53
Plug & Play Ports *53*
Non-Plug & Play Ports *53*
Device Configuration *53*
After Setting Up The Device Driver 57
Enhanced Features 57
Supported Operating Systems 57

GLOSSARY OF TERMS 59

INDEX 61

INTRODUCTION

What is PTalkDT?

PTalkDT is a user-friendly interface to Delta Tau's 32-bit driver, PComm32. It is designed to provide robust and efficient communication to PMAC®, Delta Tau's Motion Computer. Since PComm32 will continually evolve to include additional capabilities (i.e. VME PC's, PCI etc), PTalkDT has been designed so that your applications code will not be affected. Using PTalkDT ensures that your application will work for many future releases of Delta Tau's 32-bit driver (and as a result many future capabilities and versions of PMAC).

Unlike previous versions of communication libraries, PTalkDT is in the form of an ActiveX Control, a new and upcoming form of library that is taking Windows programming by storm. PTalkDT relieves you of the often-cumbersome task of writing your own communication routines. Experienced programmers know that communication functions play a critical role in creating reliable application software. We have taken all the pain out of writing communications software, and have provided what we feel is the best approach to creating a PMAC "MMI" (Man Machine Interface).

What is an ActiveX Control?

ActiveX controls are the latest addition to Microsoft's OLE (Object Linking and Embedding) family, providing unprecedented compatibility to almost any development geared application software. ActiveX controls, sometimes referred to as reusable components, give you, the programmer, the easiest way to incorporate advanced functionality into your applications with little or no programming. For those of you familiar with OCXs, ActiveX controls are the next generation; they have an added array of functions for networking ability.

What Can I use PTalkDT with?

PTalkDT can be used with the 32-bit version of Visual Basic, Visual C++ (4.x and beyond), 32-bit Delphi or C++ Builder, and just about any development package that supports ActiveX controls. In this manual, most of the examples and descriptions will pertain to Visual Basic (version 5.0) and Delphi (version 2.0).

What Can PTalkDT do for me?

PTalkDT provides you with a very stable and high-speed communications link to PMAC. Our intent is to allow you to focus on the functionality of your MMI (Man Machine Interface) by removing the burden of writing communication software to "talk" to Delta Tau's PMAC (hence, the name *PTalkDT*). PTalkDT gives your application instant communication capability to PMAC over the PC-bus, Dual Ported Ram or serial port with you writing little or no code. Furthermore, PTalkDT has been designed to quickly trap bugs in your code by centralizing the error handling (via an Event, discussed later on).

What Built in Functions Does PTalkDT Have?

Two classes of functions (or, more technically speaking, *methods*) are included, "Basic Communication" and "Extended" Functions. This manual only covers the Basic Communication methods, among them:

DownloadFile	This allows you to download a text file or multiple text files to PMAC. A powerful string substitution preprocessor is included.
Flush	A useful method to clear out PMAC's output string buffer before sending a new command.
GetControlResponse	Sends a single control character to PMAC and retrieves any pending string response from PMAC.
GetLineACK	Retrieves a string response from PMAC, stopping after receiving an ACK character (ASCII value of 6)
GetLineCR	Retrieves a response from PMAC, stopping after receiving a CR character

	(ASCII value of 13)
GetResponse	This allows you send commands to and receive string responses from PMAC in one convenient method.
LoadSettings	Retrieves the last saved communication settings.
SendChar	Send a single character to PMAC.
SelectDevice	Shows PTalkDT's Select Device dialog to allow end users to select, add, and configure PMAC devices.
SaveSettings	Stores PTalkDT communications settings to disk.
UploadData	This allows you to upload a series of string responses from PMAC—commonly used to obtain variables, motion, and PLC programs from PMAC.
DPR Read-Write	Numeric Read/Write. Enable use of DPR Automatic Features

All extended methods are prefixed with an "x" (i.e. xDPRRotBuf ()) and are detailed in Delta Tau's 32-bit driver manual (PComm32.DOC see Delta Tau's BBS or Web site WWW.DeltaTau.COM). Extended functions are rarely used.

What You Will Need to use PTalkDT

The minimum hardware and software requirements to install and support the use of PTalkDT are:

- IBM or compatible PC/AT (486, Pentium or higher CPU) with 8 MB of memory, one 3.25" floppy disk drive, and one hard disk drive with 3 MB of space
- VGA or SVGA display adapter
- Microsoft Windows 98, Windows NT, Windows 2000
- Development environment supporting 32-bit OCX controls such as Microsoft's Visual Basic (4.x or greater), Visual C++ (4.x or greater), or Delphi (2.x or greater).

How do I Get Support?

If you encounter problems your first troubleshooting steps should be to:

1. Review this manual and the Troubleshooting Guide in the Appendix of this manual-- doing this can save you time and money.
2. Get your **Serial/Registration** number from your diskettes or the back of your manual Contact our technical support for PTalkDT by faxing, sending E-mail or calling the following numbers (include serial number):

Fax:	**(818) 998-7807**
Web Page	WWW.DeltaTau.COM
E-mail:	**Support@DeltaTau.COM**
Voice Calls:	**(818) 998 2095**

We hope that PTalkDT's ease of use and this manual will provide all the help you need. (Hint: E-mail is the quickest. Include your Registration Number.).

GETTING STARTED

Installing PTalkDTPro

Before installing PTalkDTPro, read the license agreement included in this manual (behind title page). Also, please see the README.TXT file on the first installation disk. If there are corrections or additions to this manual, they will be listed in a file called README.TXT. This file can be displayed directly from the installation diskette using the Windows NOTEPAD utility. After the installation, this file can be read by double-clicking the PTalkDT README icon in the newly created program group.

Note:

Visual Basic users should install Visual Basic before PTalkDT.

To install PTalkDTPro from the Delta Tau Software CD, insert the CD into the CD drive. Auto install menu will popup. Click on the Pewin32 from the Suite to launch PTalkDTPro installation. To install PTalkDTPro from the floppy disks, put the PTalkDTPro distribution disk labeled "Disk #1" into a floppy drive and choose **File | Run** from the Program Manager. Enter **A:\SETUP.EXE** or substitute '**A**' for the letter of your floppy drive.

PComm32 drivers were rewritten when PCI, USB and Ethernet communication modes were added. It is therefore important to uninstall all old Delta Tau software products before installing Pewin32RPO.

The installation program will suggest a directory path where the program files should be copied. Use the suggested directory location for the installation for the purposes of uniformity among all PTalkDTPro users (and trouble shooting if need be).

Read the "readme.txt" file for last minute additions to this manual.

You will want to setup communication before running PTalkDTPro for the first time. For details on setting up communications see "Setting up Communications with PMAC".

What Was Installed?

The installation will create a new program group called PTalkDT. This group contains a README.TXT, and DIFFERENCES.TXT icons, three Visual Basic project, and one Visual C++ demo project icons.

The DIFFERENCES.TXT file shows the changes between one release and the next and will be useful for those upgrading to a new version of PTalkDT.

Be sure to see the "Setting up Communications with PMAC" section of this manual to "hook up" your operating system with the PMAC devices installed on your system. No communication to PMAC will occur before this is done.

We encourage you to run the Visual Basic and/or Visual C++ example projects. Please note that these will only work if you have the corresponding development environment.

After you have tried the example projects, try to make a simple application of your own by following the steps described in the section "Your First Visual Basic MMI with PTalkDT". Then you might want to look at the example program code that is provided.

Note

When these example programs were written, less than 5% of the development time was used for PMAC communications! Most of the effort went into making the various screens for these programs.

Setting up Communications with PMAC

No applications, including those created with PTalkDTPro, will be used to add, remove or configure PMAC devices in your system. Rather, communication settings have been centralized in your operating system, making the set up of each PMAC much like other devices in your computer (i.e. video card, sound card etc.) All setup is done through the Control Panels Add New Hardware Wizard. Following steps will help installing and registering the newly installed devices. Before using this application it is important that all applications that use PComm32 (the Delta Tau 32-bit communication driver) be shut down. This includes Pewin32Pro, NC for Windows, and any applications developed with PComm32 or PTalkDT.

Plug & Play Device Installation

Plug & Play are configured automatically at boot time or whenever plugged in (USB device). Devices can be reconfigured at any time for updated drivers as well.

1. Uninstall all old Delta Tau software packages
2. Install *PTalkDTPro*
3. Shutdown computer
4. Install PMAC-PCI hardware (USB UMAC can be plugged in at any time. Once computer restarts after PTalkDTPro installation, computer will detect a USB UMAC and install the driver automatically.
5. Restart computer.
6. Computer will recognize new hardware and configure the hardware. If prompted give the path of driver file(s). These file(s), depending on the operating systems, are in the following folders:
   a. Windows 98/ME c:\windows\system32\drivers
   b. Windows 2000 c:\winnt\system32\drivers

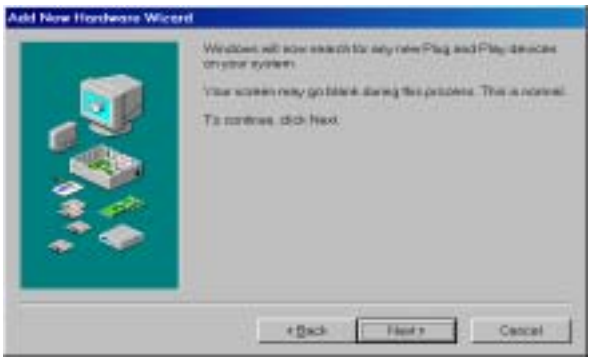
At this stage the Plug & play PMAC devices are configured and ready for use. Please see the First Time Pewin32 Users section for instructions on how to register the newly added devices.

Non-plug & play devices are configured through windows standard "hardware wizard." The steps involved in the installation of PComm32 driver under Windows 98/ME and Windows 2000 are slightly different. Next two sections describe all the necessary steps involved.

Windows 98/ME Installation (Non Plug & Play Devices)

1. Run **Add new hardware** from the control panel.

2. Continue through the auto plug and play device search wizard.

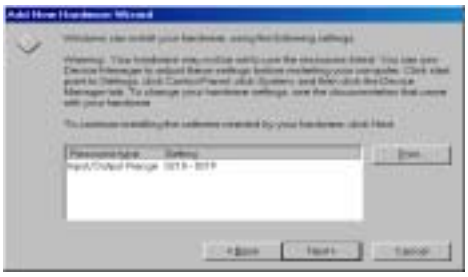


3. Continue until the following screen appears. Select **No** from the Windows auto search option.

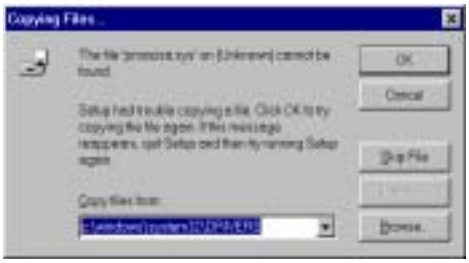
In the following screen select **other devices** from the hardware types (Needs to be done first time only.) Once device database is modified then Motion will be listed in the hardware types list and you will select the Motion type for future device additions.

4. Once device database is compiled Delta Tau Data Systems Inc. will be added to the manufacturers list. Scroll through the manufacturer list and select Delta Tau Data Systems Inc.

5. Select the Model from the available list (PMAC ISA or PMAC Serial Port) controller. Base address, Memory configuration and/or IRQ assignments are re-configurable. Serial Port configurations are done at the application level.



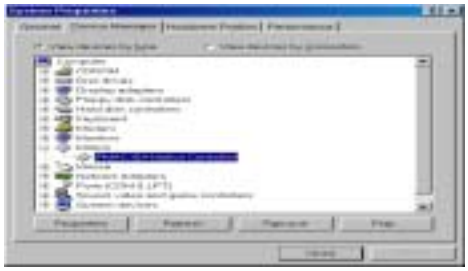
6. Select model specifies the required **driver file(s) PMACISA.SYS or PMACSER.SYS** for ISA or Serial configuration(s) respectively. If asked, specify the path of driver file(s). The required files are already in the **Windows\System32\Drivers** folder.



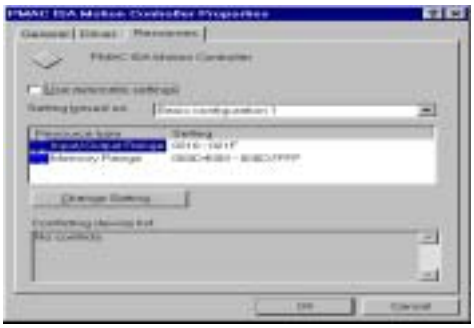
At this stage the driver is installed on your computer. A restart of computer is required after the driver installation before use.

Above steps are necessary for addition of a new device. Step (6) is different under Windows 98/ME as compared to Windows 2000. Therefore, following steps are necessary to assign appropriate resources to PMAC ISA configuration. Steps for resource reconfiguration are as follows:

1. There are essentially FOUR configurations available for ISA BUS. They are **I/O Port** only, **I/O Port w/DPRAM, I/O Port w/DPRAM & IRQ** and finally **I/O Port w/IRQ** only. One PMAC can be configured for only one of these configurations at a given time. Under Windows 98/ME. These resources can only be changed from Control Panel's device manager. Device manager can be launched from the Control Panel's System menu or directly by checking the properties of My Computer from the Desktop.



2. From the properties of PMAC ISA Motion Controller select the desired configuration and change the resources according to Jumper setting and available computer resources and respective jumper settings on PMAC controller. A computer restart may be required one the resources are altered. Once a device is configured successfully it is registered and available for use.

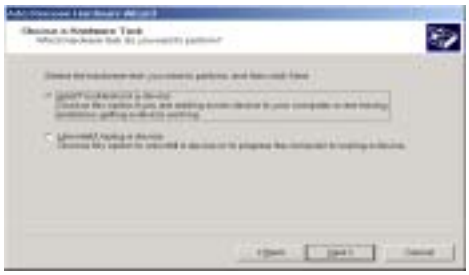


Serial Port Configuration such as, port number, baud rate, timeouts, handshake and parity options are done at the application level. Properties of a Serial Device are enabled in the PmacSelect dialog, which allows the selection of different options.

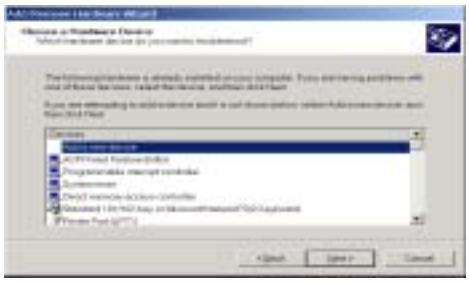
Windows 2000 Installation (Non Plug & Play Devices)

1. Run Add New Hardware from the control panel.

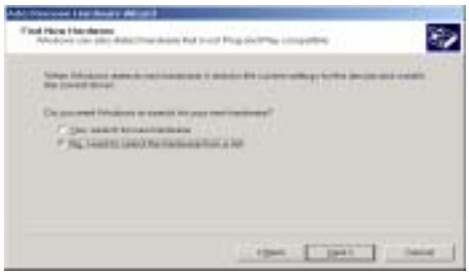
2. From choose a hardware task select Add/Troubleshoot a device



3. From choose a hardware device select add a new device.



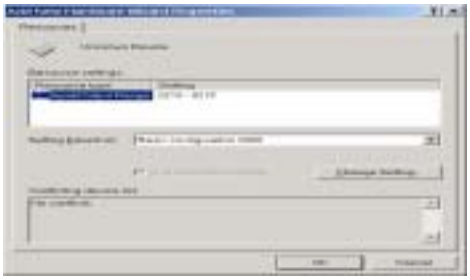
4. From find new hardware select No and continue.



5. From hardware types select other devices from the hardware types (*Needs to be done first time only*). Once device database is modified then Motion Controllers will be listed in the hardware types list and you will select the Motion Controllers type for future device additions.

6. Once device database is compiled Delta Tau Data Systems Inc. will be added to the manufacturers list. Scroll through the manufacturers list and select Delta Tau Data Systems Inc.

7. Select the Model from the available list (PMAC ISA or PMAC Serial Port) controller. Windows 2000 allows the resource configuration during installation. Therefore, at this stage, Base address, Memory configuration and/or IRQ assignments are configured. Select the appropriate configuration and after highlighting the resource press change settings to set the desired values. Confirm the “Create a forced configuration” message.



8. At this stage once you need to provide the path of driver file(s).

9. Select Model specifies **driver file(s) PMACISA.SYS or PMACSER.SYS** for ISA or the Serial configuration(s) respectively. If asked, specify the path of the driver file(s). The required files are located in the **Winnt\System32\Drivers** folder.

10. Finish the installation and restart your computer. You can review and reconfigure the resources before restarting the computer as well. These resources, however, can be changed any time by launching the device manager.

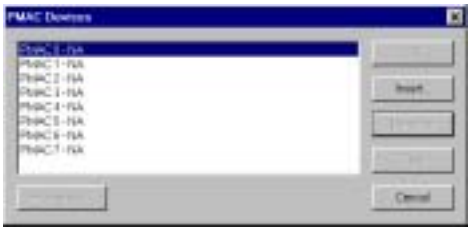
At this stage the driver is installed on your computer. A restart of computer is required after the driver installation before use.

Also at this stage your Non-Plug & play PMAC devices are configured and ready for use. Please see the First Time Pewin32 Users section for instructions on how to register the newly added devices.

First Time PTalkDTPro Users

Following steps are necessary to ensure proper startup of applications.

1. Once devices are configured, run PTalkDTPro. From the **Setup** menu, choose **General Setup and Options**.
2. If it's the first time no device will be listed in the registered device list. Press insert to register the available device(s).



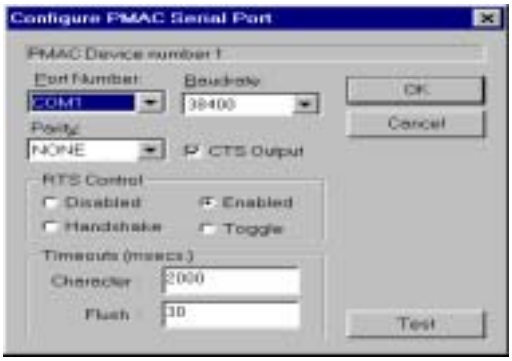
3. All configured devices are listed in the available PMAC devices list. Select the desired device to register. Repeat this procedure to register all available devices.

4. Once a device is registered, it can be selected to initiate communication.

5. Once a PMAC is listed in the PMAC Select window, it is registered and can be communicated with. It is highly recommended to test a device upon registering. At this time you should see a familiar screen and are ready to launch any window.

Serial Port Configuration

Serial port configurations are available at the application level. Properties button in the PmacSelect list allows the user to select the Port number, set the baud rate, set timeouts, handshake options and other selections as Odd/Even Parity checks.

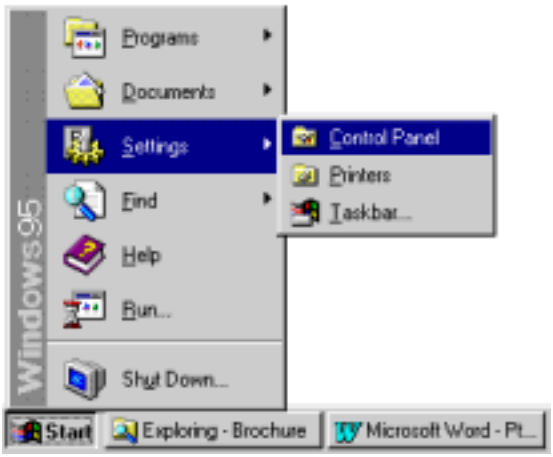


The next time the program is executed, it will start with the arrangement it had upon exiting.

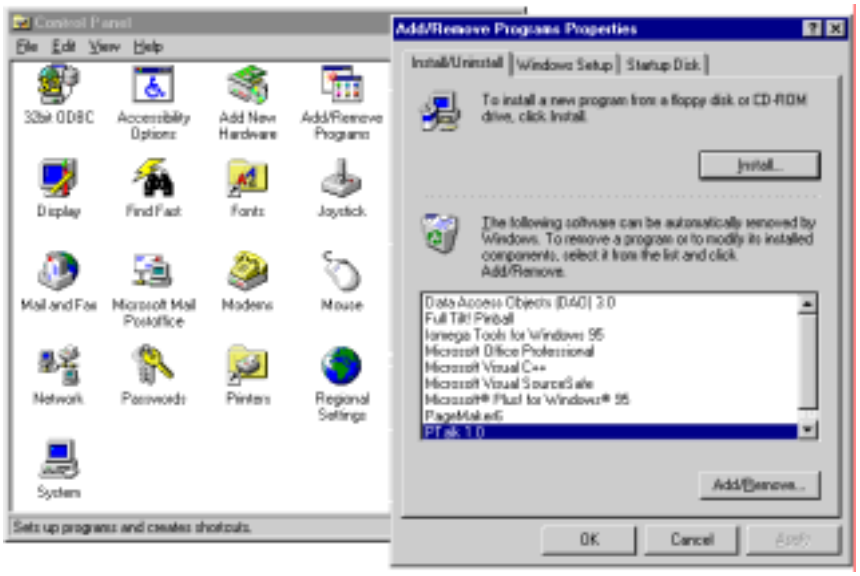
Uninstalling PTalkDT OCX

It is highly suggested that you uninstall PTalkDT before upgrading to a newer version of the product.

To uninstall PTalkDT, from Windows click the *Start* button from the taskbar and select Settings then Control Panel.



Within the control panel, select the Add/Remove Programs icon. Double click on the PTalkDT entry in the list box or push the Add/Remove button to uninstall.



All files copied during the installation will be removed (only if other programs are not currently dependent on them). Furthermore, if files have been added to the installation directory (i.e. program files you created) then the uninstall wizard will report that not all directories could be deleted. You will have to manually remove these files.

HOW TO DESIGN WITH PTALKDT

In Design Mode

First, configure your PMAC(s) in your system. See the Setting up Communications with PMAC section of this manual to "hook up" your operating system with the PMAC devices installed on your system. No communication to PMAC will occur before this is done.

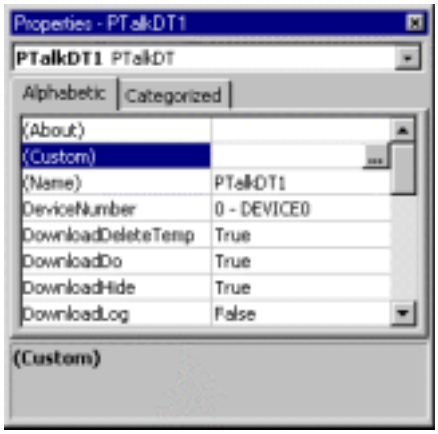
For most of the remainder of this manual, all examples will be described assuming you are using something similar to Visual Basic. If you are using a different development environment, the procedures described here will be analogous.

First add the PTalkDT control to your development environments toolbox. This is usually done by going to the "Tool" menu, and then selecting "Components". Now place a PTalkDT within the form that you are currently designing (Usually the main form of the application).

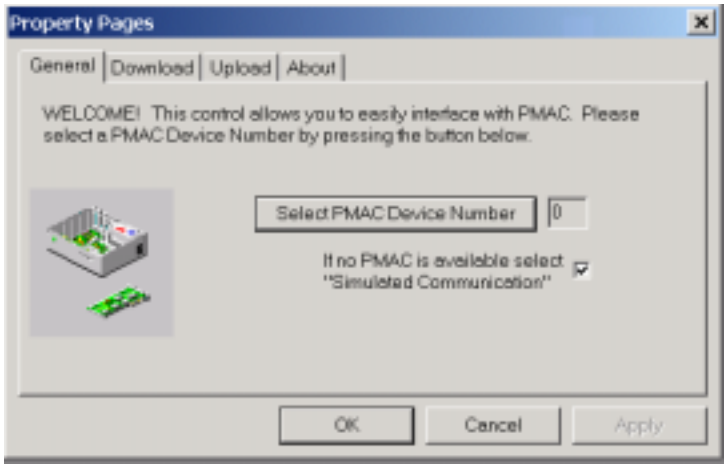
Note

PTalkDT uses Delta Tau's time tested 32-bit driver, PComm32 Pro.

The next thing most folks will want to do is configure the many properties of PTalkDT. This can be done by viewing the custom property page for a newly inserted PTalkDT. The custom property page can be viewed by double clicking on the "Custom" property (in other development environments you may double click the PTalkDT icon within the form).



The custom property page is shown below:



If you are developing without a PMAC be sure to set the **Simulate Communication** property to TRUE (check the box) and skip the next paragraph.

To choose from all functioning PMACs in your system, press the "Select PMAC Device Number button".

Each PTalkDT control you add to your project is intended to talk to a single PMAC. If your application is going to communicate with more than one PMAC, you will need to add a separate PTalkDT control for each PMAC. Within a single application, you are allowed to have a maximum of 8 PTalkDT controls. In general, it is a very good idea to use only one PTalkDT control per PMAC in your application's code.

Although the PTalkDT control has many important properties, here are a couple you should be familiar with to begin with:

Properties	**Description**
Enabled	Sets and returns an internal PTalkDT variable which enables or disables communications to the PMAC. Resets itself back to FALSE if communication can't be established. If the **Enabled** property resets itself back to FALSE, see the **LastErrorString** property for info and also see the **CONTROL PANEL's MOTION** applet.
Simulate Communication	Set to TRUE if developing without a PMAC in the system (DRY RUN)

Run Time Mode

Note

Communications can only be attempted during run time if the **Simulate Communication** property is set to FALSE AND the **Enabled** property has been successfully set to TRUE.

Upon executing your application, communications will be initialized when the **Enabled** property is or has been set to "True". This is not automatically done—you must set **Enabled** yourself (either in design mode or in your code).

Note

During run time, the PTalkDT control icon is *not* visible.

The PTalkDT methods in the table below are typically used for communication. Again, if the **Enabled** property is FALSE or **Simulate Communication** is "TRUE", no communications to PMAC will actually take place, and these methods will do nothing.

Methods	Description
DownLoadFile()	Download a file to PMAC.
Flush()	Empty out PMAC's input/output buffer.
GetControlResponse()	Send PMAC a control character and retrieve any pending response from PMAC.
GetResponse()	Send PMAC a command, and retrieve the subsequent response.
LoadSettings()	Restore the last stored communications configuration from disk.
SendChar()	Send a single character to PMAC.
SelectDevice()	Shows PTalkDT's Select Device dialog to allow end users to select, add, and configure PMAC devices.
SaveSettings()	Store PTalkDT's communications configuration to disk.
UploadData()	Upload a series of string responses to a file.
DPR Read-Write routines	Numeric Read/Write. Enable use of DPR Automatic Features

The following simple Visual Basic example shows how to establish basic PMAC communications via the PC Bus:

```
Private Sub Form_Load ()
        Dim response As String
        Dim return_value As Long

        PTalkDT1.Enabled = True
        ' test communications  by a query of motor status
        return_value = PTalkDT1.GetResponse(response, "?")
        if return_value = 0 then  ' if communications failed…
                '   An error occurred--, either handle here using use the
'  LastError and LastErrorString properties of PTalkDT or
'  have the OnError  event handle this.
        endif
End Sub
```

Debugging

The **OnError** event is used for trouble shooting and debugging. If you can't establish communications, or if you are timing out, or if a PMAC error was generated, then this event will be called. As a suggestion, your code associated with **OnError** may simply display the error message to you (while developing), or perhaps act on the error without the user ever knowing a problem occurred (good for release versions of your application). See the **OnError** event description for more details.

Altering, Saving and Retrieving PTalkDT Settings at Run Time

Communication Settings

After you've added PMAC devices to your operating system (see "Device Configuration" section of this manual) the communication settings are saved in the registry.

\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PMAC\DEVICE0
for PMAC device 0 and
\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\PMAC\DEVICE1
for PMAC device 1 and so on….

Three communication properties that aren't stored in the registry but rather in an initialization file are the **Enabled,** Simulate Communications and **DeviceNumber** properties. You may ensure that the state of these properties will persist by calling PTalkDT's **LoadSettings()** at the beginning of your application and **SaveSettings()** at the termination of your program.

General Settings

In addition to **SimulateCommunication** and **DeviceNumber**, the following properties may be saved/restored in PTalkDT's initialization file (via the **SaveSettings()/LoadSettings()** methods):

DownloadDo
DownloadParse
DownloadLog
DownloadMap
DownloadDeleteTemp
DownloadHide
DownloadShowErrors
DownloadMaxErrors
UploadHide
UploadShowProgress
UploadNoComments
UploadAppend

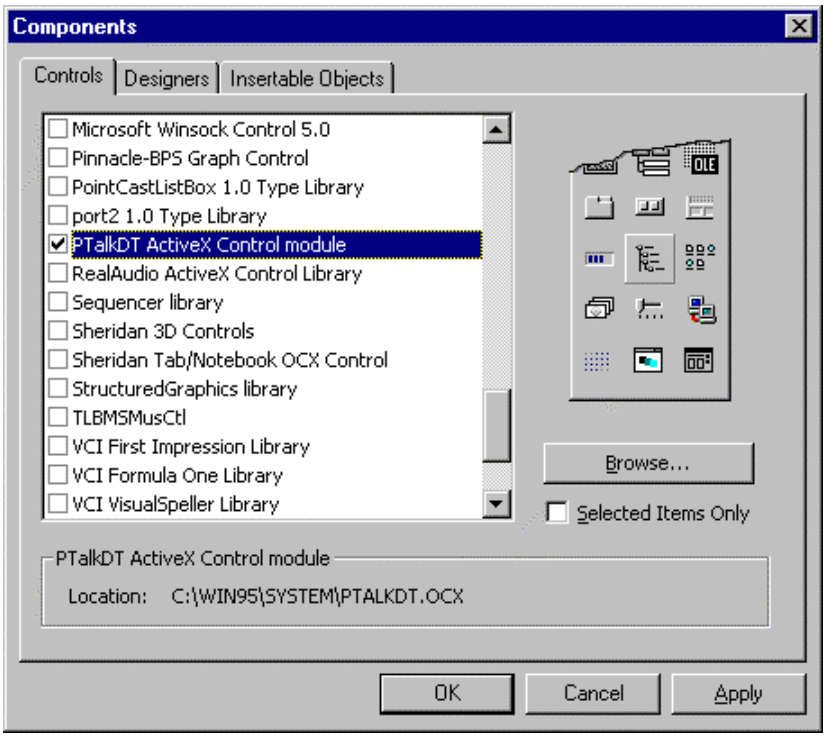
YOUR FIRST VISUAL BASIC MMI WITH PTALKDT

Overview

This section will guide you through building a simple Visual Basic 5.0 MMI (man-machine interface) application using PTalkDT. The resulting application displays the value of PMAC's constantly changing servo counter register. The code generated here can be similarly constructed with other development environments.

Instructions

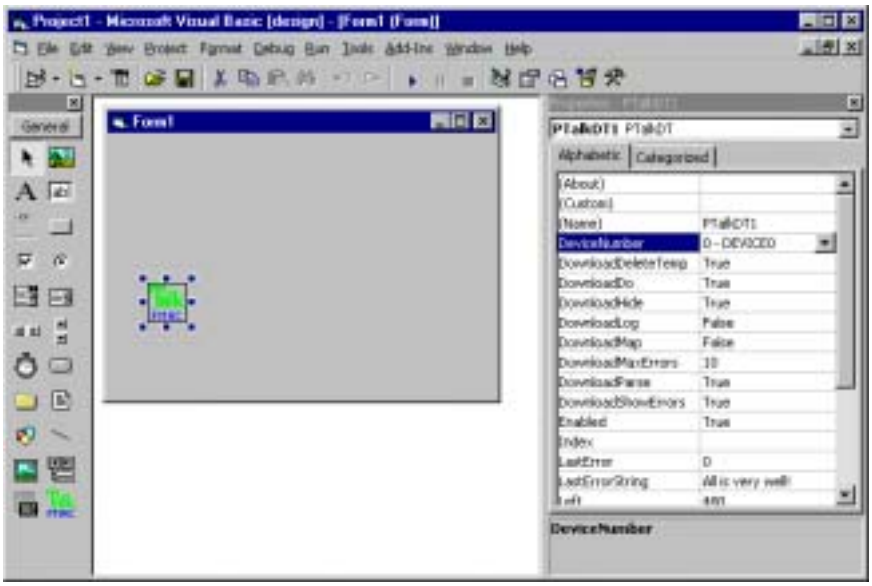
1. Start Visual Basic 6.0 and choose "Standard EXE" for project type.
2. Choose Project from the top menu bar and select Components. Select the "PTalkDT Control" module and then select the OK button.



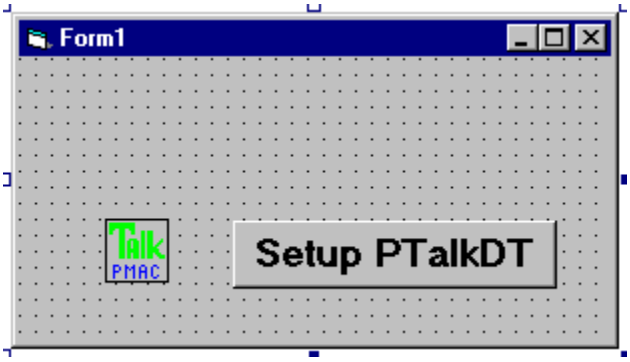
The PTalkDT icon 

should appear at the bottom of your tool palette:

3. Click on the PTalkDT icon and place it anywhere on a blank Visual Basic form.
4. With the PTalkDT icon on the form selected, press F4 to view the Visual Basic PTalkDT property window.



5. Now we will begin to form the user interface. To allow the user to select a PMAC in their system, and modify PTalkDT's properties, place a button on the form and set the **caption** property to "Setup PTalkDT".



6. Double click on the Setup PTalkDT button to associate code with the pressing of the button. Enter the following code

```
Private Sub Command1_Click()
    PTalkDT1.SelectDevice
    PTalkDT1.SaveSettings
End Sub
```

This code will call PTalkDT's **SelectDevice()** and **SaveSetings()** methods when the Configure button is pressed giving the user the ability to configure the appropriate communication settings at run time and making them persistent. **SaveSettings()** combined with the use of **LoadSettings()** ensures that the end users won't have to reconfigure PTalkDT settings every time the user runs the program.

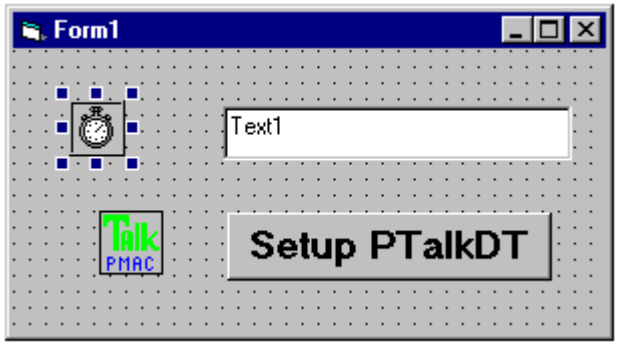
Setting the **Enabled** property to TRUE will reinitialize communication if required.

Now put the **LoadSettings()** method in the Form_Load() method of the form by double clicking on any "free" spot within the form. The routine should look like so when done:

```
Private Sub Form_Load()
    PTalkDT1.LoadSettings
    PTalkDT1.Enabled = True
End Sub
```

Setting the **Enabled** property to TRUE will guarantee that PTalkDT will at least attempt to establish communication with the PMAC **DeviceNumber** selected.

7. Next lets add real time display of PMAC's servo clock. Add a text control and a timer control to the form.
8. Press F4 to view the timer's property window.
9. Set the timer's property **Interval** to 10.



11. Double click on the timer and add the following code (shown below in bold):

```
Private Sub Timer1_Timer()
   Static Response As String
   Static return_value as Long

   return_value =PTalkDT1.GetResponse(Response,"RX0")
   Text1.Text = Response
End Sub
```

12. Press F5 to run your application. If all is well the servo clock is very quickly being updated in your newly created PTalkDT application. Try pressing the "Setup PTalkDT" button to setup. If you do have a PMAC be sure to uncheck the "SimulateCommunication" check box within the property page window. Notice that the PTalkDT icon is not visible during run time (neither is the timer control's icon).
13. For further examples, see the installation group box in your desktop's "Start\Programs" menu. Also check out Delta Tau's BBS/Website. Study the code and feel free to use it in your own applications.

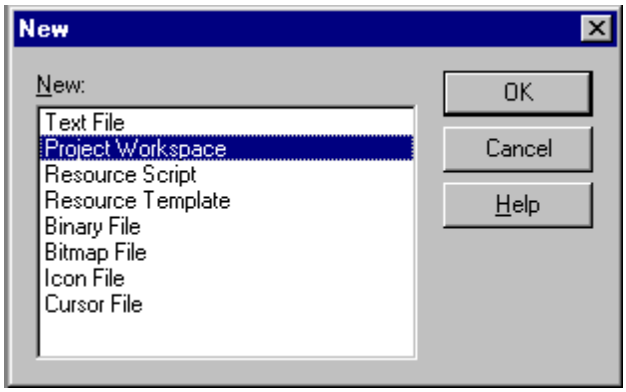
YOUR FIRST MICROSOFT VISUAL C++ MMI WITH PTALKDT

Overview

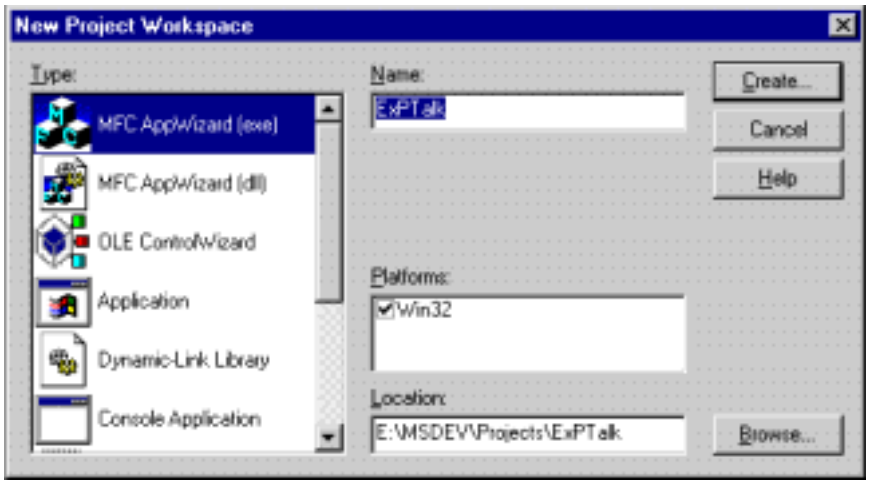
This section will guide you through building a simple Microsoft Visual C++ MMI (man-machine interface) application using PTalkDT. The resulting application displays the value of PMAC's constantly changing servo counter register. The code generated here can be similarly constructed with other development environments.

Instructions

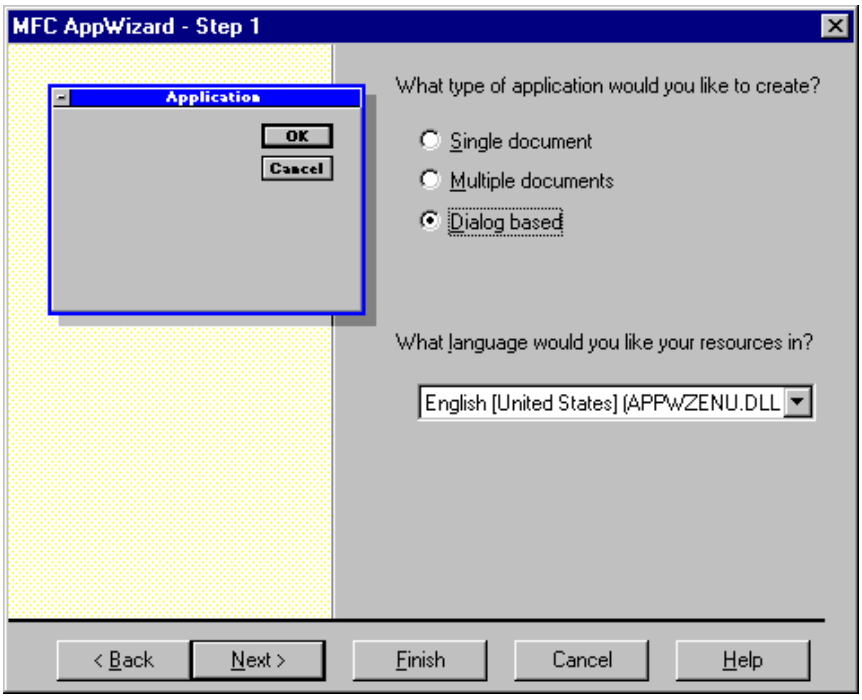
1. Start Visual C++.
2. Choose FILE from the top menu bar and select New. Highlight Project Workspace from the list box and then select the OK button.



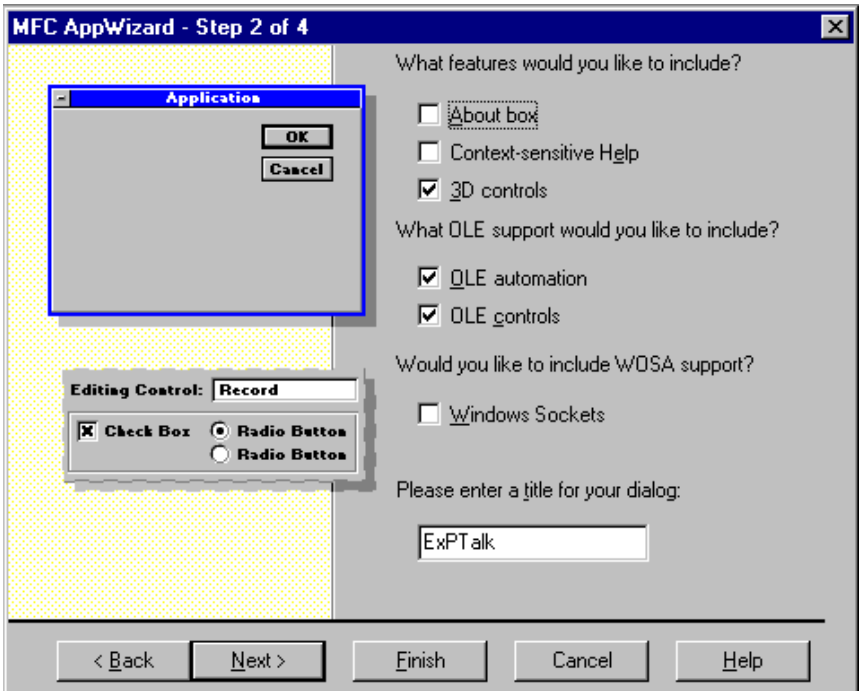
3. In the next dialog box, select MFC AppWizard (exe) from the list box, type in a project name (such as **`ExPTalk`**), and click on Create:



4. On the next dialog box, select the Dialog Based radio button and click on Next >:



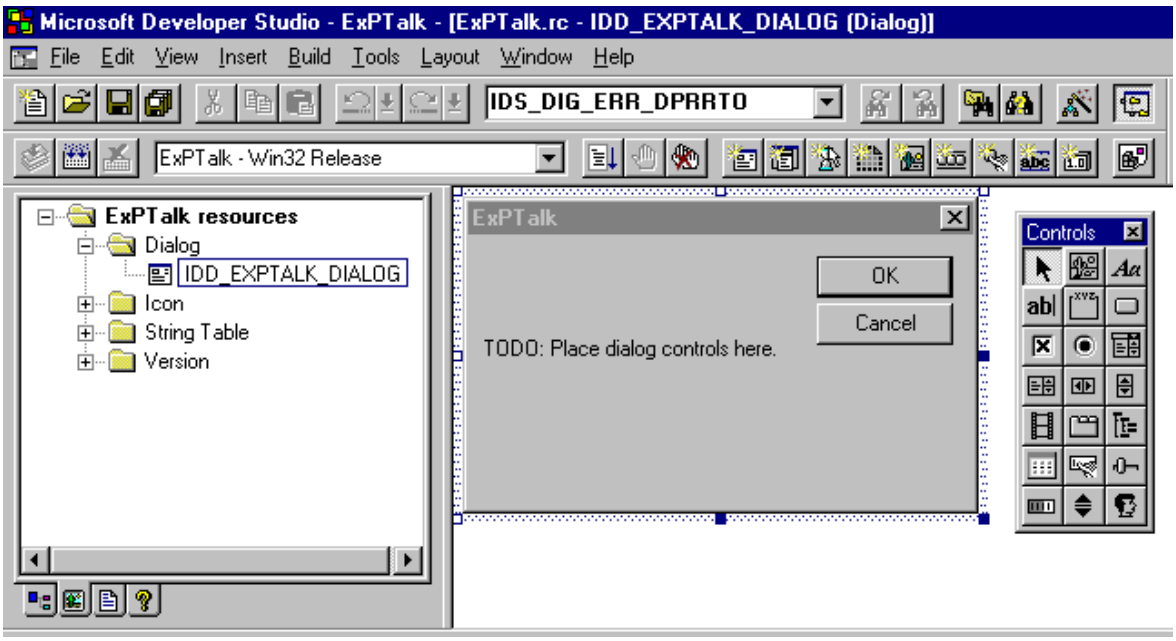
5. On the last dialog box, place a check mark for 3D controls, OLE automation, and OLE controls and click on Finish:



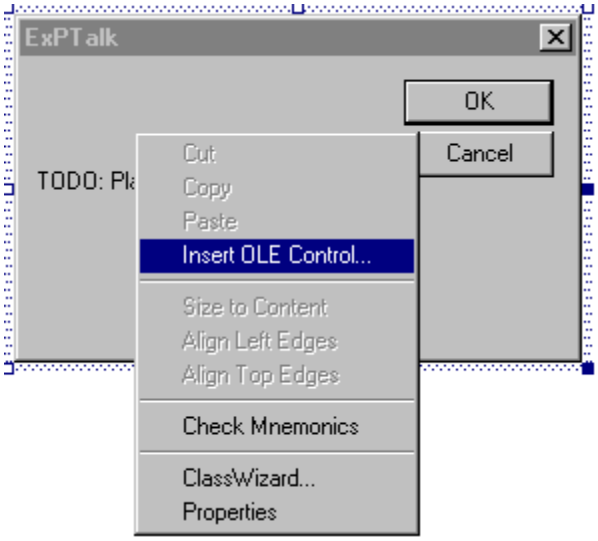
At this point, a set of C++ files have been generated in a directory with the same name as the project name you selected. Go ahead and compile this newly created project and run it to verify it works correctly. When you execute this program, a blank dialog box with an OK and Cancel button should appear:

Now, let us go back and add the PTalkDT control to this dialog box.

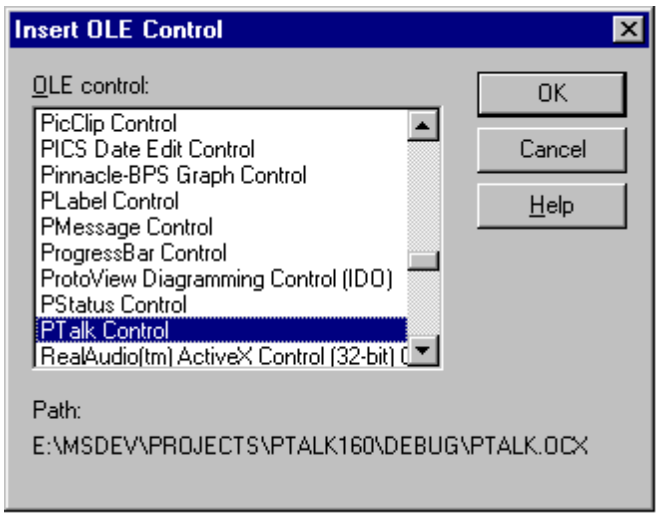
From within the Visual C++ workspace environment, select to view the existing resources (which were created by the AppWizard in the previous steps) and click on the Dialog resource. Your screen should look like this:



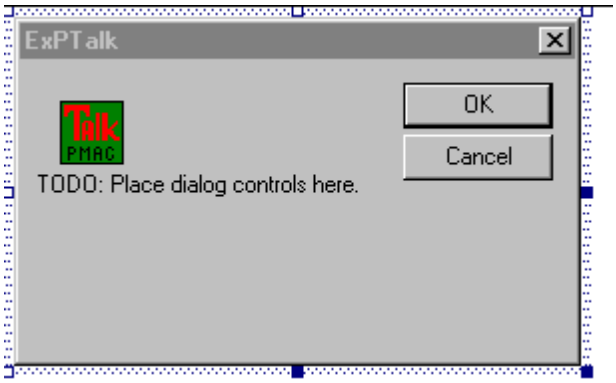
6. With your mouse pointing to the dialog box (on the right, called "ExPTalk"), click the *right* mouse button to expose the following pop-up menu and select Insert OLE Control.



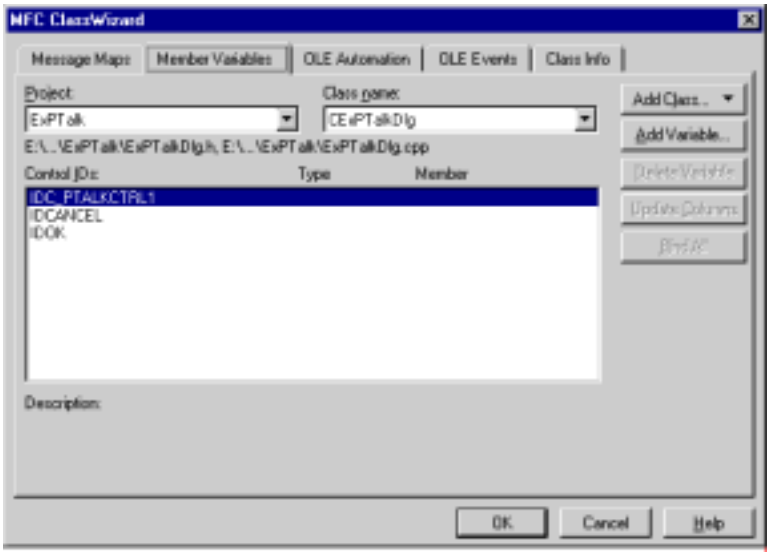
7. A new dialog box will appear containing a list of available controls. Scroll down and choose the control called PTalkDT Control and then click OK:



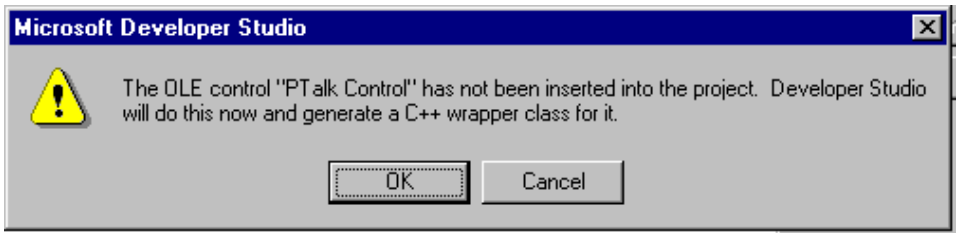
The PTalkDT control should now be visible in your dialog box:



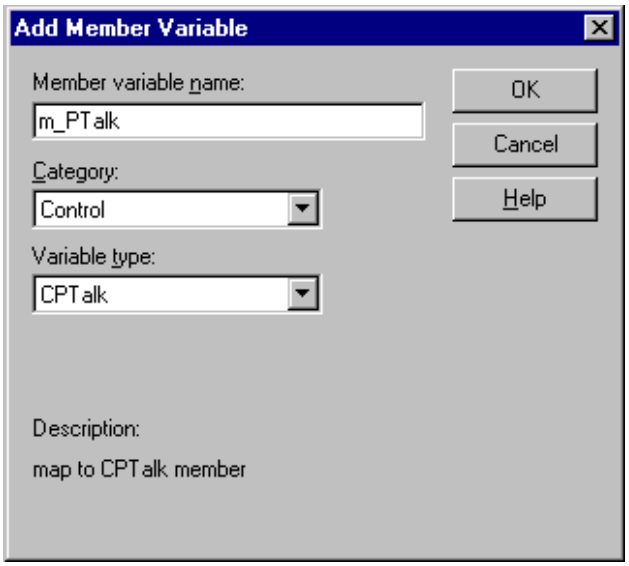
8. Our next step is to use the MFC Class Wizard within Visual C++ to generate code that will create a control class for this newly added PTalkDT control. To do this, select the View menu and then Class Wizard. The MFC Class Wizard dialog box will appear. Select the Member Variables tab. Your screen should look like this:



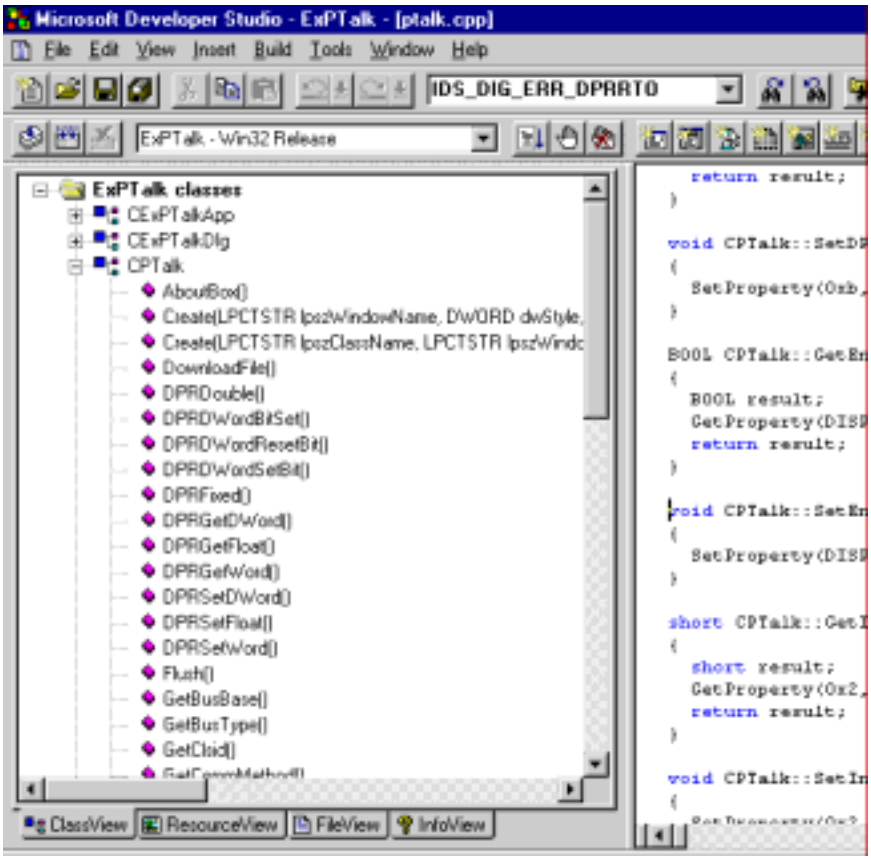
Highlight `IDC_PTALKCTRL1` and press Add Variable. When you do this, the following dialog box will appear:



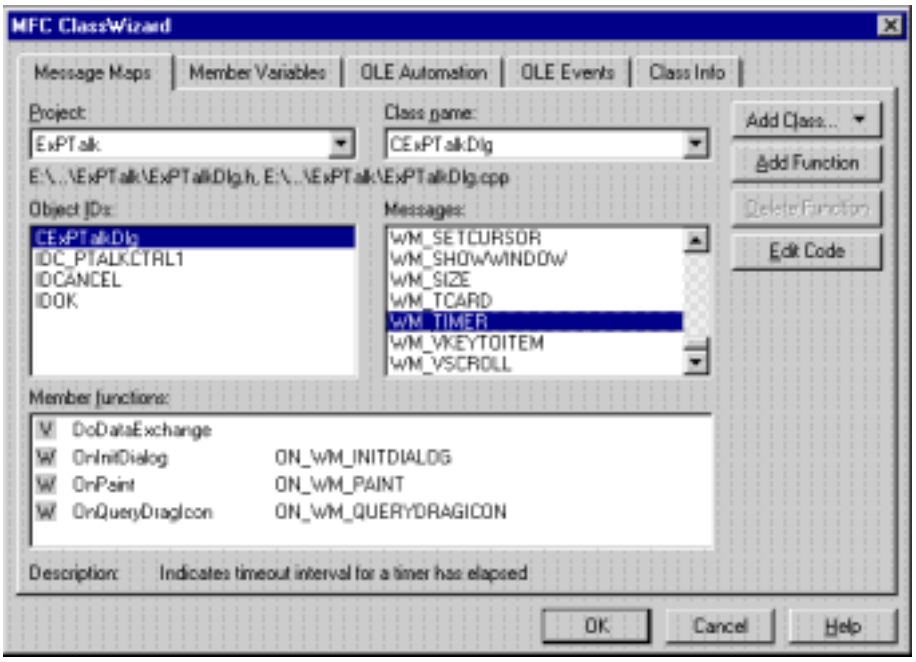
9. Select *OK*. On the next dialog box, select OK again.
10. The next dialog box will ask you to type in a name for the variable that will be used to access all of PTalkDT's properties and methods in your C++ code. Use the name shown on below and click on OK:



Click on OK again. At this point, the MFC Class Wizard has generated a new C++ file and header file, which contains the code to allow you're to access all the functionality of PTalkDT! For each property, a specific function has been created, making it easy to read or set the various PTalkDT properties. To see these new functions created, select to view the classes in your project. When you do this, your screen should look like this:

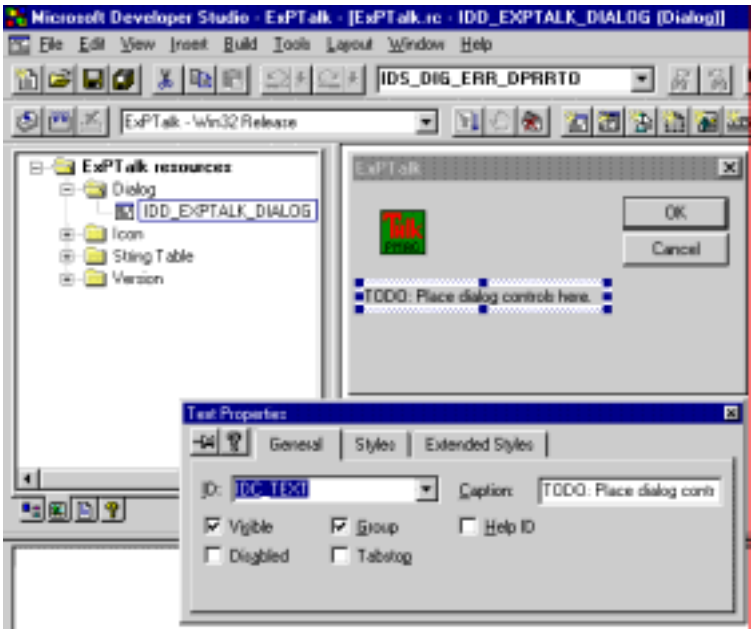


11. We will now add a timer function to our dialog box, which will use PTalkDT to continuously query PMAC for information. We will use the MFC Class Wizard again to do this. Select the View menu and then Class Wizard. The MFC Class Wizard dialog box will appear. Select the Message Maps tab, locate, and highlight the item called WM_TIMER in the Messages list box. Click on Add Function and then OK.

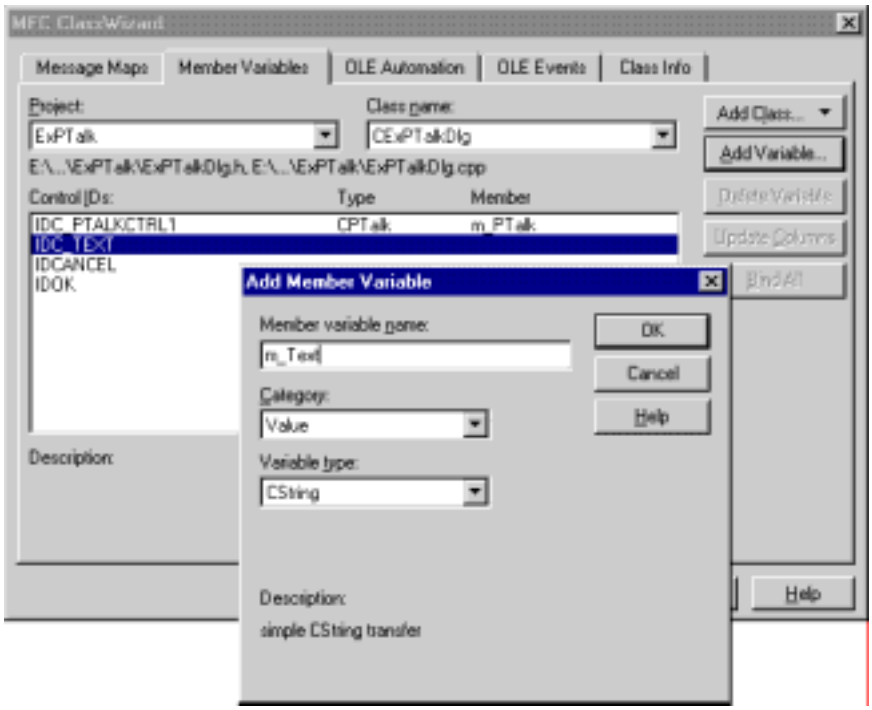


A new function for the timer is has now been created. We will add code to this function later on.

12. We must now change the name of the static text that was automatically placed there by the AppWizard when the project was first created. We will be using this text to display the response from PMAC in our dialog box. Bring up the dialog box in the resource editor, double -click on the static text and modify its variable name as shown on the next page. The name used here is `IDC_TEXT`.



Now bring up the Class Wizard again to create a usable variable so that we may access this static text in our code. Select the View menu and then Class Wizard. The MFC Class Wizard dialog box will appear. Select the Member Variables tab, locate, and highlight the item called IDC_TEXT in the Control ID list box. Click on Add Variable type in m_Text for the variable name and then OK twice to back out of all the dialog boxes.



13. We now need to add code to setup the properties of PTalkDT to correspond to how you will be communicating with PMAC. In the file **`ExPTalkDlg.CPP`**, locate the function `CExPTalkDlg::OnInitDialog` and add the following code shown in bold:

```
BOOL CExPTalkDlg::OnInitDialog()
  {
      CDialog::OnInitDialog();

      SetIcon(m_hIcon, TRUE);              // Set big icon
      SetIcon(m_hIcon, FALSE);         // Set small icon

      m_PTalkDT.SetEnabled(TRUE);
      SetTimer(1, 50, NULL);
      return TRUE
  }
```

14. Now locate the code for the `CExPTalkDlg::OnTimer` function. This function will be called on a repeated basis about every 50 milliseconds. In this function we will place the code to query PMAC for the contents of its servo clock register and copy this number to the static text variable `m_Text`. Add the code shown in bold:

```
 void CExPTalkDlg::OnTimer(UINT nIDEvent)
{
      // TODO: Add your message handler code here
      TCHAR buf[255];
      BSTR response = SysAllocString(L"");
```

```
        m_PTalkDT.GetResponse(&response,"RX0");
        USES_CONVERSION;
        strcpy(buf,OLE2T(response));
        m_Text = buf;
        UpdateData (FALSE);
        SysFreeString(response);

        CDialog::OnTimer(nIDEvent);

}
```

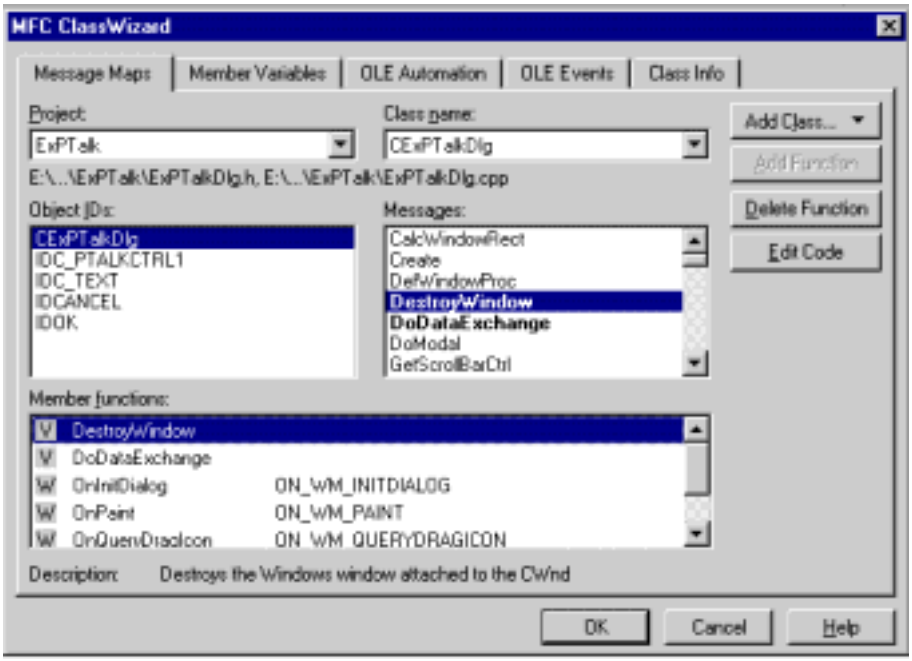
Also, add this #include statement after the

```
#include <afxpriv.h>
```

It should look like this after:

```
#include "stdafx.h"
#include "ExPtalk.h"
#include "ExPtalkDlg.h"
#include <afxpriv.h>
```

15. We must use the MFC Class Wizard one last time to created one last function. Select the *View* menu and then Class Wizard. The MFC Class Wizard dialog box will appear. Select the Message Maps tab and locate and highlight the item called DestroyWindow in the Messages list box. Click on Add Function and then OK.



Locate this newly added function `CExPTalkDlg::DestroyWindow` and add the code shown in bold:

```
BOOL CExPTalkDlg::DestroyWindow()
{

        KillTimer (1);

        return CDialog::DestroyWindow();
}
```

You are now ready to run your program. Press F5 to run the program. If your PMAC has been configured appropriately in the **CONTROL PANELs MOTION** applet, you should see a number in the label which is continually counting upwards. Notice that the PTalkDT icon is not visible during run time.

PTALKDT REFERENCE

Documentation Conventions

This manual uses the following notational conventions:

```
Source code and data structures are displayed in a monospaced
typeface.
```

Note

Warnings or important information are bounded on top and bottom with single lines.

Overview

As mentioned before, PTalkDT is a 32-bit ActiveX control designed to handle all communications between your application and Delta Tau's PMAC. It is meant to be used as a PMAC application development tool. You may use PTalkDT in any 32-bit OLE container application such as Visual Basic, Delphi , etc. PTalkDT's built-in features make most communications tasks as easy as calling a simple method (function).

Note

PTalkDT will force PMAC's I-variable I3=2 at all times to ensure high speed and efficient communications.

PTalkDT Properties

Enabled

Data Type Boolean or Long Integer
Default Value Zero (for "False")
Description Enables or disables PtalkDT from communicating with PMAC.
Remarks Used to specify or determin is PtalkDT is allowed to communicate with PMAC. You must set this property to "True" and **SimulateCommunication** to "False to allow PtalkDT to communicate to PMAC.

Note

At end of the **SelectDevice()** method the **Enabled** property is set to True internally. If communication was successful, the Enabled property retains the True value.

LastError

Data Type Long Integer
Default Value 0
Description Used in the debugging of an application using PTalkDT
Remarks Used to read the state of PtalkDT's most recent communications error. This property is usually used in the debugging of an application. You may want to set this property to 0 just before calling a PtalkDT method. Then recheck **LastError** for a non-zero code. The error may be due to a PMAC reported error (i.e. invalid command) or bad parameters passed to a PtalkDT method.
See Also **LastErrorString, OnError**

LastErrorString

Data Type String
Default Value NULL

Description Used in the debugging of an application using PTalkDT.
Remarks Returns the last error string generated. The error may be due to a PMAC reported error (i.e. invalid command) or bad parameters passed to a PTalkDT method. See also the **OnError()** event..
See Also **LastError**, **OnError**

Device Number

Data Type Long Integer
Default Value 0
Description Used to uniquely identify which PMAC device the PtalkDT will use to communicate to.
Remarks The **CONTROL PANEL'S "MOTION"** applet may be used to add/remove or set up PMAC's in your operating system. A device number (starting from 0) will be associated with each PMAC you add. Use this same device number when specifying which PMAC you want your PtalkDT Active X control to communicate to.
See Also **Enabled, SimulateCommunication**

DeviceNumber

Data Type Long Integer
Default Value 0
Description Used to uniquely identify which PMAC device the PTalkDT will use to communicate to.
Remarks The **CONTROL PANEL's "MOTION"** applet may be used to add/remove or setup PMAC's in your operating system. A device number (starting from 0) will be associated with each PMAC you add. Use this same device number when specifying which PMAC you want your PTalkDT ActiveX control to communicate to.
See Also **Enabled**, **SimulateCommunication**

DownloadDeleteTemp

Data Type Boolean or Long Integer
Default Value >0 True
Description For use with the **DownloadFile()** method. To eliminate any intermediary files that are created after downloading, set this property to True.
Remarks Intermediary files will be created if the **DownloadParse** method is set to true. The files created will have the same name as the original argument to **DownloadFile()**, but the extensions will be "PMA", "LOG", and "56K".
See Also **DownloadDo, DownloadHide, DownloadLog, DownloadParse, DownloadMap, DownloadShowErrors, DownloadMaxErrors**

DownloadDo

Data Type Boolean or Long Integer
Default Value >0 True
Description Used when the **DownloadFile()** method is called. To only to Macro parsing and compiling of PLCC's set this property to False and the end resulting file (*.56K) will not get downloaded to PMAC.
Remarks Rarely used
See Also **DownloadDeleteTemp, DownloadHide, DownloadLog, DownloadParse, DownloadMap, DownloadShowErrors, DownloadMaxErrors**

DownloadHide

Data Type Boolean or Long Integer
Default Value True
Description Used when the **DownloadFile()** method is called. To hide the **DownloadFile()** dialog set this value to True.
Remarks Can be set in the property page.
See Also **DownloadDeleteTemp,DownloadDo, DownloadLog, DownloadParse, DownloadMap, DownloadShowErrors, DownloadMaxErrors**

DownloadLog

Data Type Boolean or Long Integer
Default Value False
Description Used when the **DownloadFile()** method is called. To have the event log of the **DowloadFile()** method recorded, set this property to True. The file created will have the same name as the argument to **DownloadFile()** method but have the "LOG" file extension (i.e. "MYFILE.LOG").
Remarks Can be set in the property page.
See Also **DownloadDeleteTemp, DownloadDo, DownloadHide, DownloadParse, DownloadMap, DownloadShowErrors, DownloadMaxErrors**

DownloadMap

Data Type Boolean or Long Integer
Default Value False
Description Used when the **DownloadFile()** method is called. To create a cross referencing of MACROS used set this property to True. The file created will have the same name as the argument to **DownloadFile()** but with the "MAP" extension.
Remarks To be of any use, the **DownloadParse** property must be set to True.
See Also **DownloadDeleteTemp,DownloadDo, DownloadHide, DownloadLog, DownloadParse,, DownloadShowErrors, DownloadMaxErrors**

DownloadMaxErrors

Data Type Long Integer
Default Value 10
Description Used when the **DownloadFile()** method is called. This property limits the number of errors before the **DownloadFile()** method aborts.
Remarks Can be set in the property page.
See Also **DownloadDeleteTemp, DownloadDo, DownloadHide, DownloadLog, DownloadParse, DownloadMap, DownloadShowErrors**

DownloadParse

Data Type Boolean or Long Integer
Default Value True
Desctription Used when the **DownloadFile()** method is called. If the file you are downloading has PLCC's or macro definitions, then you'll want to set this property to True. Otherwise, if the file is strictly PMAC native code with no PLCC's feel free to set **DownloadParse** to False.
Remarks Can be set in the property page.
See Also **DownloadDeleteTemp, DownloadDo, DownloadHide, DownloadLog, DownloadMap, DownloadShowErrors, DownloadMaxErrors**

DownloadShowErrors

Data Type Boolean or Long Integer
Default Value False
Description Used when the **DownloadFile()** method is called. If errors occurred in the downloading of a file and this property is set to True, the log file that was created will be shown in NotePad.EXE.
Remarks If the **DownloadLog** property is False no Errors will be shown.
See Also **DownloadDeleteTemp, DownloadDo, DownloadHide, DownloadLog, DownloadParse, DownloadMap, DownloadMaxErrors**

UploadAppend

Data Type Boolean or Long Integer
Default Value False
Description Used in the **UploadData()** method. When uploading data to a file, you have the option of overwriting the existing file (**UploadAppend** = False) or appending to the existing one (**UploadAppend** = True)
Remarks Can be set in the Property Page
See Also **UploadHide, UploadNoComments, UploadShowProgress**

UploadHide

Data Type Boolean or Long Integer
Default Value True
Description Used in the **UploadData()** method. To have the **UploadData()** methods dialog box hide itself, set this property to True.
Remarks Can be set in the Property Page
See Also **UploadAppend, UploadNoComments, UploadShowProgress**

UploadNoComments

Data Type Boolean or Long Integer

Default Value False

Description Used in the **UploadData()** method. The specified file that will be created (or appended to—see the other options), will contain no comments, i.e. only the actual uploaded responses will be written into the file.

Remarks Can be set in the Property Page

See Also **UploadAppend, UploadHide, UploadShowProgress**

UploadShowProgress

Data Type Boolean or Long Integer
Default Value True
Description During the upload process (if the dialog box is not hidden), a progress bar will be shown, indicating the upload status if this property is set to True. To use this option correctly, you must specify a positive value for *num_lines* argument to the **UploadData()** method. Also, *num_lines* should be as close as possible to the expected number of responses to be received.
Remarks Can be set in the Property Page
See Also **UploadAppend, UploadHide, UploadNoComments**

PTalkDT Methods

DPRAvailable()

Description Used to check to see that Dual Ported Ram is available for use with PTalkDT.

Return Value A Boolean value indicating whether or not PTalkDT was able to access PMAC's Dual Ported Ram.

Visual Basic & Delphi
```
[form].controlname.ConfigureDriver
value = Mainform.PTalk1.ConfigureDriver
```

C++
```
BOOL controlname->ConfigureDriver()
value = PTalkDT->ConfigureDriver()
```

Remarks This method is useful for those applications that will use PMAC's Dual Ported Ram. You may disable that portion of your application that uses DPR if this function returns False.

DownloadFile (*file name*)

Description Downloads a text file (or a series of files) to PMAC and checks for errors.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi
```
[form].ctrlname.DownloadFile (filename$, options As Long)
Mainform.PTalkDT1.Downloadfile ("c:\files\main.pmc")
```

C++
```
BOOL controlname->DownloadFile (char *filename,long options)
PTalkDT1->Downloadfile ("c:\\files\\main.pmc")
```

Remarks This method is useful for downloading commands and programs to PMAC. A full preprocessor is built in and is invoked if the **DownloadParse** property has been set to TRUE. The only parameter *filename* is a string containing the full path of any valid ASCII text file that contains preprocessor or PMAC compatible code. The following properties should be set up before this method is called:

Property	What it does
DownloadDo	Used when the **DownloadFile()** method is called. To only to Macro parsing and compiling of PLCC's set this property to False and the end resulting file (*.56K) will not get downloaded to PMAC.
DownloadDeleteTemp	Intermediary files will be created if the **DownloadParse** method is set to true. The files created will have the same name as the original argument to **DownloadFile()**, but the extensions will be "PMA", "LOG", and "56K".
DownloadHide	Used when the **DownloadFile()** method is called. To hide the **DownloadFile()** dialog set this value to True.
DownloadLog	Used when the **DownloadFile()** method is called. To have the event log of the **DowloadFile()** method recorded, set this property to True. The file created will have the same name as the argument to **DownloadFile()** method but have the "LOG" file extension (i.e. "MYFILE.LOG").
DownloadMap	Used when the **DownloadFile()** method is called. To create a cross referencing of MACROS used set this property to True. The file created will have the same name as the argument to **DownloadFile()** but with the "MAP" extension.
DownloadMaxErrors	Used when the **DownloadFile()** method is called. This property limits the number of errors before the **DownloadFile()** method aborts.
DownloadParse	Used when the **DownloadFile()** method is called. If the file you are downloading has PLCC's or macro definitions, then

	you'll want to set this property to True. Otherwise, if the file is strictly PMAC native code with no PLCC's feel free to set **DownloadParse** to False.
DownloadShowErrors	Used when the **DownloadFile()** method is called. If errors occurred in the downloading of a file and this property is set to True, the log file that was created will be shown in NotePad.EXE.

About the preprocessor	The preprocessor provides the ability to use *#include* file statements and macro string substitution in your code just like in the C and C++ languages. Delta Tau's PMAC Executive Program supports this same use of *#include* file and macro string substitution.

Directive	Example	Description
#define *name* {*command or variable*}	`#define COUNTER P1`	Declares the name of a macro string substitution. For every occurrence of *name*, the preprocessor will substitute in {*command or variable*}.
#define *name*	`#define DEBUG_MODE`	Declares a variable name that can be used for compiler directives.
#include *"filename"*	`#include "macros.txt"` `#include "C:\\PE\\macros.txt"`	Preprocess and download the specified file from the current directory or given path. This is useful for including multiple files as part of the download.

#ifdef *name*	`#ifdef DEBUG_MODE`
	`…`
	`#else`
	`…; (this code ignored)`
	`#endif`
		Tests to see if *name* has been previously declared. If so, the subsequent lines of code are included in the download.

#ifndef *name*	`#ifndef DEBUG_MODE`
	`…`
	`#else`
	`…; (this code ignored)`
	`#endif`
		Tests to see if *name* has NOT been previously declared. If *name* has NOT been declared, the subsequent lines of code (until the next **#else** or **#endif**) are included in the download.

#else	`#ifdef DEBUG_MODE`
	`…`
	`#else`
	`…`
	`#endif`
		In the example, if DEBUG_MODE has not been declared, the lines of code following the **#else** are included in the download. This directive provides a means to alternate lines of code when the **#ifdef** or **#ifndef** conditions are false.

#endif	`#ifdef DEBUG_MODE`
	`…`
	`#else`
	`…`
	`#endif`
		For every **#ifdef** or **#ifndef**, you must include a matching **#endif**.

DPRDouble (LSB_word, MSB_word)

Description Converts a PMAC 48 bit floating point data value (as found in PMAC's Dual Port RAM) to a 64 bit floating point value compatible with Visual Basic, C++, Delphi, etc.

Return Value A 64-bit floating-point value (of type double) converted from the passed in parameters.

Visual Basic & Delphi
```
[form].controlname.DPRDouble (lo_val as Long,hi_val As Long)
value = Mainform.PTalk1.DPRDouble (lo_val,hi_val)
```

C++
```
double controlname->DPRDouble (long lo_val,long hi_val)
value = PTalkDT->DPRDouble (lo_val,hi_val)
```

Remarks Floating-point values within PMAC's internal memory are stored as 48-bit numbers. Floating-point values in your PC's memory are typically stored as 32-bit values (*float* or *single*) and 64-bit values (*double*). These formats are not directly compatible. When accessing various floating point registers in PMAC's Dual Port RAM, they can be accessed by reading two 32-bit integers (or "words") and combining them to form a PC-compatible 64-bit number. For this function, the first word, *LSB_word*, specified in the parameters is treated as the least significant word. And the second word, *MSB_word*, is the most significant word. This function will prove very useful when reading the many floating point registers in the Real Time Buffer section of PMAC's Dual Port RAM.

DPRFixed (*LSB_word, MSB_word*)

Description Converts a PMAC 48 bit integer data value (as found in PMAC's Dual Port RAM) to a 64 bit floating point value compatible with Visual Basic, C++, Delphi, etc.

Return Value A 64-bit floating-point value (of type double) converted from the passed in parameters.

Visual Basic & Delphi
```
[form].controlname.DPRFixed (lo_val as Long,hi_val As Long)
value = Mainform.PTalk1.DPRFixed (lo_val,hi_val)
```

C++
```
double controlname->DPRFixed (long lo_val,long hi_val)
value = PTalkDT->DPRFixed (lo_val,hi_val)
```

Remarks Integer values within PMAC's internal memory are stored as 48-bit numbers. Floating-point values in your PC's memory are typically stored as 32-bit values (*float* or *single*) and 64-bit values (*double*). These formats are not directly compatible. When accessing various integer based registers in PMAC's Dual Port RAM, they can be accessed by reading two 32-bit integers (or "words") and combining them to form a PC-compatible 64-bit number. For this function, the first word, *LSB_word*, specified in the parameters is treated as the least significant word. And the second word, *MSB_word*, is the most significant word. This function will prove very useful when reading the many integer based registers in the Real Time Buffer section of PMAC's Dual Port RAM such as motor position.

DPRDWordBit Set/Reset and BitSet Methods

DPRDWordSetBit (offset, bit_position)
DPRDWordResetBit (offset, bit_position)
DPRDWordBitSet (offset, bit_position)

Description These functions can be used to set (assign a bit value of 1), reset (assign a bit value of 0), or query, respectively, the state of an individual bit within a 32 bit integer located in the address space of PMAC's Dual Ported Ram.

Return Value **DPRDWordSetBit** and **DPRDWordResetBit** return "True" if successful, otherwise "False". **DPRDWordBitSet** returns the value of the bit being queried, either a 1 or 0.

Visual Basic & Delphi
```
[form].ctrlname.DPRDWordSetBit (offset as long, bit As long)
[form].ctrlname.DPRDWordResetBit (offset As long, bit As long)
[form].ctrlname.DPRDWordBitSet (offset As long, bit As long)
Call Mainform.PTalk1.DPRWordSetBit (&H0800&,2)
```

C++
```
BOOL controlname-> DPRDWordSetBit (long offset, long bit)
BOOL controlname-> DPRDWordResetBit (long offset, long bit)
BOOL controlname-> DPRDWordBitSet (long offset, long bit)
PTalkDT->DPRFixed (0x800,2)
```

Remarks The *offset* parameter is the number of PMAC addresses from the base address of the DPR within the PMAC address space. PMAC's Dual Ported Ram base address is always $D000 (the last DPR address is $DFFF). For example to specify address $D200 in the DPR use a value of $200 (that is hex 200, or 512 decimal)

The *bit* parameter specifies the bit within the double word. Valid ranges for *bit* are from 0 to 31.

DPRGetDWord and DPRSetDWord Methods

DPRGetDWord (base_address_offset)
DPRSetDWord (base_address_offset, value)

Description These functions can be used to read and write 32 bit integers from and to PMAC's Dual Ported RAM.

Return Value **DPRGetDWord** returns the 32-bit integer read from PMAC's Dual Ported Ram. **DPRSetDWord** returns "True" if successful, "False" if a failure occurred.

Visual Basic & Delphi
```
[form].ctrlname.DPRDGetDWord (offset As long) As long
[form].ctrlname.DPRDSetDWord (offset As long,value As long)
value = Mainform.PTalk1.DPRGetWord (&H0800&)
```

C++
```
long controlname-> DPRDGetDWord (long offset)
BOOL controlname-> DPRDSetDWord (long offset, long value)
value = PTalkDT->DPRGetWord (0x800)
```

Remarks The *base_addr_offset* parameter is the number of PMAC addresses from the base address of the DPR within the PMAC address space. PMAC's Dual Ported Ram base address is always $D000 (the last DPR address is $DFFF). For example to specify address $D200 in the DPR use a value of $200 (that is hex 200, or 512 decimal)

Example
```
Var
    aBool  : Bool;
      aLong :  LongInt;
      offset : LongInt ;
      aString: string[11];
begin
// Assign offset of 512  from DPR Base Address (PMAC
Address $D200)
```

```
  offset := 512;
  aLong := Form1.PTalkCtrl1.DPRGetDWord(offset);
  Str(aShort, aString);  // Convert to a string
  Edit8.Text := aString; // Write to an edit box
// Write to first 4 bytes of DPR
  aBool := Form1.PTalkCtrl1.DPRSetDWord(0,aShort);
end;
```

DPRGetFloat and DPRSetFloat Methods

DPRGetFloat (offset)
DPRSetFloat (offset, value)

Description These functions can be used to read and write 32 floating-point values from and to PMAC's Dual Ported Ram.

Return Value **DPRGetFloat** returns the 32-bit floating-point value read from PMAC's Dual Ported RAM. **DPRSetFloat** returns "True" if successful, "False" if a failure occurred.

Visual Basic & Delphi
```
[form].ctrlname.DPRDGetFloat (offset As long) As long
[form].ctrlname.DPRDSetFloat (offset As long,value As Single)
value = Mainform.PTalk1.DPRGetFloat (&H0800&)
```

C++
```
float controlname-> DPRDGetFLoat (long offset)
BOOL controlname-> DPRDSetFloat (long offset, float value)
value = PTalkDT->DPRGetFloat (0x800);
```

Remarks The *offset* parameter is the number of PMAC addresses from the base address of the DPR within the PMAC address space. PMAC's Dual Ported Ram base address is always $D000 (the last DPR address is $DFFF). For example to specify address $D200 in the DPR use a value of $200 (that is hex 200, or 512 decimal)

PMAC's special m-variable format "F" may be used to easily assign 32 bit floating-point values to Dual Ported RAM.

Example
```
Var
     aBool  : Bool;
    aFloat : Single;
     offset : LongInt ;
begin
     offset := 100;   // Assign offset from PMAC's base address
     aFloat := 1.2345;      // Assign float
     aBool := Form1.PTalkCtrl1.DPRSetFloat(offset,aFloat);
     aFloat := Form1.PTalkCtrl1.DPRGetFloat(offset);
end;
```

DPRGetMem and DPRSetMem Methods

DPRGetMem
DPRSetMem (long Offset, long NumLongWords, long FAR* LongArray)

Description These functions can be used to read and write a user defined number of 32 bit integers from and to PMAC's Dual Ported RAM.

Return Value Both these functions return TRUE if successful, otherwise FALSE (0).

Visual Basic & Delphi
```
[form].ctrlname.DPRGetMem(Offset As Long, NumLongWords As
long,LongArray as Long) As long
[form].ctrlname.DPRSetMem(Offset As Long, NumLongWords As
long,LongArray as Long) As long
Private Sub cmdDPRGetMem_Click()
Dim mylongarray(0 To 9) As Long
```

```
' BOOL DPRGetMem(long Offset, long NumLongWords, long FAR*
LongArray)
' Note OFFSET is in PMAC Words
If (PTalkDT1.DPRGetMem(0, 10, mylongarray(0))) Then
    MsgBox ("Cool")
Els eMsgBox ("Un cool")
End If
End Sub
Private Sub cmdSetMemTest_Click()
Dim mylongarray(0 To 9) As Long
  mylongarray(0) = 1
  mylongarray(1) = 2
  mylongarray(2) = 3
  mylongarray(3) = 4
  mylongarray(4) = 5
  mylongarray(5) = 6
  mylongarray(6) = 7
  mylongarray(7) = 8
  mylongarray(8) = 9
  mylongarray(9) = 10
  ' BOOL DPRSetMem(long Offset, long NumLongWords, long
FAR* LongArray)
  ' Note OFFSET is in PMAC Words
  If (PTalkDT1.DPRSetMem(0, 10, mylongarray(0))) Then
    MsgBox ("Cool")
  Else
    MsgBox ("Un cool")
  End If
End Sub
```

C++

`long` *`controlname`*`->` **`DPRDGetWord`** (long Offset, long NumLongWords, long FAR* LongArray);

`BOOL` *`controlname`*`->` **`DPRDSetWord`** (long Offset, long NumLongWords, long FAR* LongArray);

```
value = PTalkDT->DPRGetWord (0,10,LongArray[0]);
```

Remarks

The *offset* parameter is the number of PMAC addresses from the base address of the DPR within the PMAC address space. PMAC's Dual Ported Ram base address is always $D000 ($60000 for Turbo). For example to specify address $D200 in the DPR use a value of $200 (that is hex 200, or 512 decimal). The *NumLongWords* parameter should be the size of the array passed in.

DPRGetWord and DPRSetWord Methods

DPRGetWord(bank, offset)

DPRSetWord(bank, offset, value)

Description

These functions can be used to read and write 16 bit integers from and to PMAC's Dual Ported RAM.

Return Value

DPRGetWord returns the 16-bit integer read from PMAC's Dual Ported am. **DPRSetWord** returns “True” if successful, “False” if a failure occurred.

Visual Basic & Delphi

`[form].ctrlname``.`**`DPRGetWord`** `(`*`bank`* `As Long,`*`offset`* `As long) As long`

`[form].ctrlname``.`**`DPRSetWord`** `(`*`bank`* `As Long,`*`offset`* `As Long,`*`value`* `As integer)`

Visual Basic

```
value = Mainform.PTalk1.DPRGetWord ('X',&H0800&)
```

Delphi

```
// 88 = 'X' in ASCII
value = Mainform.PTalk1.DPRGetWord (88,&H0800&)
```

C++

```
long controlname-> DPRDGetWord (long bank,long offset)
BOOL controlname-> DPRDSetWord (long bank,long offset, int value)
value = PTalkDT->DPRGetWord ('X',0x800);
```

Remarks

The *bank* parameter specifies PMAC's X or Y address space. Use a value of 24 for X or 25 for Y (or more intuitively an ASCII character "x", "X", or "y", "Y").

The *offset* parameter is the number of PMAC addresses from the base address of the DPR within the PMAC address space. PMAC's Dual Ported Ram base address is always $D000 (the last DPR address is $DFFF). For example to specify address $D200 in the DPR use a value of $200 (that is hex 200, or 512 decimal).

```
PMAC's m-variable formats "X" and “Y” may be used to
easily assign 16 bit integers to Dual Ported RAM
(i.e. m1->X:$D200,0,16,s).
```

Example

```
Var
  aBool  : Bool;
  aShort : short;
  offset : LongInt ;
  aString: string[100];
begin
// Read from PMAC DPR Address X$D200
  offset := 512;
  aShort := Form1.PTalkCtrl1.DPRGetWord('X',offset);
  Str(aShort, aString);    // Convert to a string
  Edit8.Text := aString;   // Write to an edit box
// Write to first two bytes of DPR
aBool:=Form1.PTalkCtrl1.DPRSetWord('X',offset,aShort);
end;
```

Flush ( )

Description Empties PMAC’s response buffer and character I/O port.

Return Value “True” for success else “False”

Visual Basic & Delphi

```
[form].controlname.Flush
Call Mainform.PTalk1.Flush
```

C++

```
BOOL controlname->Flush ();
PTalkDT->Flush();
```

Remarks Empties the contents of PMAC’s output buffer queue and strips out any remaining characters in PMAC’s ASCII queue. The characters that get “Flushed” cannot be read. Note that this method has no parameters.

GetControlResponse (Response, Control Char)

Description Sends a control character to PMAC and waits for PMAC’s response.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi

```
[form].ctrlname.GetControlResponse (Response As String,
controlChar As Integer)
Mainform.PTalk1.GetControlResponse (Response, 16)
```

C++ `BOOL controlname->GetControlResponse (char *response, char control);`
`result = PTalkDT->GetControlResponse (response,'P');`

Remarks Sends a control character to PMAC and waits up to **Timeout** iterations for PMAC's response.

Note

This function will not send control-T. This is to avoid putting PMAC in a full-duplex mode. Doing so will keep PTalkDT from re-establishing communications the next time the application is run.

GetLineAck (Response)

Description Gets a string from PMAC up to the terminating <ACK> character.

Return Value Number of characters retrieved.

Visual Basic & Delphi `[form].controlname.GetLineAck (Response As String)`
`Mainform.PTalk1.GetLineAck (Response)`

C++ `long controlname->GetLineAck (char *response);`
`result = PTalkDT->GetLineAck (response);`

Remarks Communications routine for receiving a response from PMAC. Certain commands can cause PMAC's response to contain multiple <CR> characters. This will receive the entire response up to the terminating <ACK> character or timeout condition. This response string can be as large as 16000 characters. For most applications the **GetResponse** method should be used instead of **GetLineAck**. Exceptions would be when you want to receive something from PMAC without sending a command as in a terminal program.

GetLineCR (Response)

Description Gets a string from PMAC up to the terminating <CR> character.

Return Value Number of characters retrieved

Visual Basic & Delphi `[form].controlname.GetLineCr (Response As String)`
`Mainform.PTalk1.GetLineAck (Response)`

C++ `long controlname->GetLineAck (char *response);`
`result = PTalkDT->GetLineAck (response);`

Remarks Communications routine for receiving a response from PMAC. This routine will read a pending response up to the next <CR> or <ACK> character.
Although PMAC will respond to commands with a terminating <ACK> character, sometimes only the part of PMAC's response up to the next <CR> is desired at the moment. In this situation the **GetLineCR** method can be used. For most applications the **GetResponse** method should be used instead of **GetLineCR**. Exceptions would be when you want to receive something from PMAC without sending a command as in a terminal program.
Response: Response string will never be greater than 255 characters.

GetResponse (Response, Command)

Description Sends a string to PMAC and waits for PMAC's response.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi `[form].controlname.GetResponse (Response As String, command As String)`
`Mainform.PTalk1.GetResponse (Response,"#1P")`

C++ `BOOL controlname->GetResponse (char *response,char *command);`
`result = PTalkDT->GetResponse (response,"#1P");`

Remarks General-purpose communications routine for sending a command, and receiving a consequential response from PMAC. *Response* will never be greater than 16,000 characters. *Command* should not be greater than 250 characters if using Bus or Serial Port, and should not exceed 150 characters if using the Dual Ported Ram.

IsLineWaiting ( )

Description Used to determine if PMAC is waiting to say something to the host.

Return Value non-zero : PMAC has an ASCII response pending for the host
zero : PMAC does not have an ASCII response pending for host

Visual Basic & Delphi
```
[form].controlname.IsLineWaiting
result = Mainform.PTalk1.IsLineWaiting
```

C++
```
BOOL controlname->IsLineWaiting ();
result = PTalkDT->IsLineWaiting();
```

Remarks This method is excellent for creating applications which will periodically check to see if PMAC has an ASCII response for the Host computer. Instead of calling **GetResponse** to see if a response is pending use **IsLineWaiting** instead. **IsLineWaiting** will not remove any contents of PMAC's output buffer, and will not timeout. Note that this method does not have parameters.

LoadSettings ( )

Description Loads the last stored PTalkDT settings.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi
```
[form].controlname. LoadSettings
result = Mainform.PTalk1. LoadSettings
```

C++
```
BOOL controlname-> LoadSettings();
result = PTalkDT-> LoadSettings();
```

Remarks Loads the last stored parameters via the **SaveSettings** method. If the **Enabled** property is set to TRUE before this method is called, communication will be re-attempted after the settings have been loaded.

Settings include the following properties:

DeviceNumber
SimulateCommunication
DownloadDo
DownloadParse
DownloadLog
DownloadMap
DownloadDeleteTemp
DownloadHide
DownloadShowErrors
DownloadMaxErrors
UploadHide
UploadShowProgress
UploadNoComments
UploadAppend

LockPMAC ( )

Description Locks the PMAC resource from other threads and processes.

Return Value None

Visual Basic & Delphi
```
[form].controlname. LockPMAC
Mainform.PTalk1.LockPMAC
```

C++
```
void controlname-> LockPMAC();
PTalkDT-> LockPMAC();
```

Remarks To be used in conjunction with **ReleasePMAC()**. These two methods lock and release the PMAC resource respectively. This should only be used very sparingly to ensure that no cross talk occurs when using the **SendChar()**, **SendLine()** and any **GetLine()** methods. All other communication methods are thread safe.

For Example:

```
LockPmac() // Hold off any other processes or threads
SendLine("?") // Send the line
GetLineACK(response)  // Get the response
ReleasePMAC() //Let other threads have access to PMAC
```

ReleasePMAC( )

Description Releases the PMAC resource for other threads and processes

Return Value None

Visual Basic & Delphi
```
[form].controlname.ReleasePMAC
Mainform.PTalk1.ReleasePMAC
```

C++
```
void controlname-> ReleasePMAC();
PTalkDT->ReleasePMAC();
```

Remarks To be used in conjunction with **LockPMAC()**. These two methods lock and release the PMAC resource. This should only be used very sparingly to ensure that no cross talk occurs when using the **SendChar()**, **SendLine()** and any **GetLine()** methods. All other communication methods are thread safe.

For Example:

```
LockPmac() // Hold off any other processes or threads
SendLine("?") // Send the line
GetLineACK(response)  // Get the response
ReleasePMAC() //Let other threads have access to PMAC
```

SaveSettings ( )

Description Saves the current communications settings.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi
```
[form].controlname.SaveSettings
Mainform.PTalk1. SaveSettings
```

C++
```
BOOL controlname->SaveSettings();
result = PTalkDT->SaveSettings();
```

Remarks Stores the following properties to an initialization file whose name is the same as PTalkDT's **name** property (i.e. **PTalkDT1.ini**)

DeviceNumber
SimulateCommunication
DownloadDo
DownloadParse
DownloadLog
DownloadMap

DownloadDeleteTemp
DownloadHide
DownloadShowErrors
DownloadMaxErrors
UploadHide
UploadShowProgress
UploadNoComments

SelectDevice( )

Description Shows PTalkDT's Select Device dialog to allow end users to select, add, and configure PMAC devices.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi
```
[form].controlname.SelectDevice()
result = Mainform.PTalk1.SelectDevice
```

C++
```
BOOL controlname->SelectDevice();
result = PTalkDT->SelectDevice();
```

Remarks Calling this function will give your end users the ability to remove, add, and reconfigure PMAC devices. Consider your end users capability before calling this routine.

SendChar (Character)

Description Sends a single ASCII character, *aChar*, to PMAC.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi
```
[form].controlname.SendChar (character As Long)
Mainform.PTalk1. SendChar(Asc("P"))
```

C++
```
BOOL controlname-> SendChar(long character);
result = PTalkDT->SendChar('P');
```

Remarks Sends a single ASCII character to PMAC without waiting for PMAC to respond. This will come in handy when you need to send characters one at a time either in a terminal or when sending control characters.

SendLine (Command)

Description Sends a string to PMAC.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi
```
[form].controlname.SendLine (command As String)
Mainform.PTalk1.GetResponse ("ListProg1")
```

C++
```
BOOL controlname->SendLine (char *command);
result = PTalkDT->GetResponse ("ListProg1");
```

Remarks This function is here only for backward compatibility. Use GetResponse() instead. If you find that you have to use this function follow these instructions very carefully.
SendLine() sends PMAC a command string. PMAC WILL HAVE A RESPONSE TO THE SENT COMMAND. If PMAC has two or more pending responses for the host computer, the PMAC will suspend the running of all PLC's and motion programs, as well as any incoming ASCII commands. Therefore, always call GetLineACK() after using SendLine() to purge any pending response from PMAC.

One last very important thing. Use the LockPMAC() method before the SendLine() and the ReleasePMAC() method after the GetResponse() call to ensure that your program won't cause any "CROSS TALK" amongst other threads or processes that are using Delta Tau's 32 bit driver, PComm32.

For Example:

```
LockPmac() // Hold off any other processes or threads
SendLine("?") // Send the line
GetLineACK(response)  // Get the response
ReleasePMAC() //Let other threads have access to PMAC
```

ShowPropertyPage ( ) [OBSOLETE]

Description This method is available for backward compatibility only. Use the new SelectDevice() method instead. Both ShowPropertyPage() and SelectDevice() do the same thing.
Shows PTalkDT's Select Device dialog to allow end users to select, add, and configure PMAC devices.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi
```
[form].controlname.ShowPropertyPage
result = Mainform.PTalk1.ShowPropertyPage
```

C++
```
BOOL controlname->ShowPropertyPage();
result = PTalkDT->ShowPropertyPage();
```

Remarks Calling this function will give your end users the ability to remove, add, and reconfigure PMAC devices. Consider your end users capability before calling this routine.

UploadData (File Name, Command, Options, Expected Number of Lines)

Description Uploads a series of responses from a PMAC command to a text file.

Return Value Non-zero if successful, zero when a failure occurred.

Visual Basic & Delphi
```
[form].ctrlname.UploadData (filename As String, command As
String, number_of_lines As Long)
Mainform.PTalkDT1.UploadData
("c:\files\main.pmc","i0..1023",1023)
Mainform.PTalkDT1.UploadData ("c:\files\plc1.pmc","list plc
1",0)
```

C++
```
BOOL controlname->UploadData (char *filename,char
*command,long number_of_lines)
result = PTalk1->UploadData
("c:\\files\\main.pmc","i0..1023",1023)
```

Remarks This method is useful for receiving a series of responses from PMAC and writing them to a file. With this method you can upload items such as motion and PLC programs, I-, P-, Q- and M- variables, and gathered data to a data file. By default, helpful comments are also written into the file, including a time and date stamp. The first parameter *filename* is the full path of any valid ASCII text file that will contain the upload data. The second parameter *command* is the actual command string that will be sent to PMAC to generate the upload data. The third parameter *number_of_lines* specifies the number of expected lines so that the optional progress bar can show the correct progress status during the upload. For example, if the command was **I0..1023** (which uploads the values of I-variables I0 through I1023), you expect to receive 1024 responses and you would set *number_of_lines* equal to 1024. The following PTalkDT properties summarizes the available options:

Name of Option	Description
UploadNoComments	The specified file that will be created (or appended to—see the other options), will contain no comments, i.e. only the actual uploaded responses will be written into the file.
UploadHide	The usual dialog box that appears showing the progress of the upload is not shown. As a result, you will not be able to cancel the upload process before it completes.
UploadAppend	If the specified file already exists, the newly uploaded data will be appended to the end of the specified file. If the specified file does not exist, it will be created.
UploadShowProgress	During the upload process (if the dialog box is not hidden), a progress bar will be shown, indicating the upload status. To use this option correctly, you must specify a positive value for *number_of_lines*. In addition, this value should be as close as possible to the expected number of responses to be received.

PTalkDT Events

OnError

Description Signals when a PTalkDT initialization or communications error has occurred.

Visual Basic
```
Private Sub PTalk1_OnError(ByVal ErrorNumber As Long, ErrorString
As String)
FormDebug.Text1.Text = Str(ErrorNumber)
FormDebug.Text2.Text = ErrorString
ErrorCount = ErrorCount + 1
FormDebug.Text3 = Val(ErrorCount)
End Sub
```

Remarks The **OnError** event was meant to be used for troubleshooting. If you can't establish communications, if you are timing out, if a PMAC error was generated etc. then this event will notify you. Your code in this routine may simply display the message, *ErrorString*, to the user (good for developing), or perhaps act on the *ErrorCode* without the end user ever knowing a problem occurred (good for releases). The ErrorCode and ErrorString parameters passed in this event represent the **LastError** and **LastErrorString** properties just modified state.

See Also PMAC Software Reference Manual \ On line commands \ I6 for an explanation of PMAC Errors.

Trouble Shooting

To see if the problem you are encountering is communications related, try disabling the communications via the **SimulateCommunication** property.

Symptom PTalkDT can't seem to load or fails unpredictably.
Cause Visual Basic users should be sure to install Visual Basic first then PTalkDT second.

Symptom You can't establish serial communications but everything works O.K. once you run the PMAC Executive Program.
Cause Some PMAC firmware versions (before 1.16A) set the hardware handshaking lines incorrectly on power up or reset. To get around this problem short pins 4 & 5 (CTS & RTS, clear to send and request to send) on the PC's serial port connector.

Symptom You can't establish serial communications period.

Cause Are you using a known working serial cable? You may just want to see exactly what your PMAC's baud rate is and use that.

If your PMAC has been put in full-duplex mode (by sending it a control-t) communications with PTalkDT will not occur. Putting a jumper on the board to put it in a factory default state (E51 on PMAC1, E3 on PMAC2) should eliminate this problem.

Look at the port setup from the operating systems control panel . Also, try the supplied "HyperTerminal" application.

Symptom Serial communications is losing characters.
Cause Setup your COM port from the Control Panel of the operating system. Make sure that you are NOT using a FIFO, and that HARDWARE FLOW CONTROL is being used.

Symptom In Microsoft Visual C++ after inserting a PTalkDT control, you can't see any of the member variables displayed in the class wizard.
Cause The problem may be that the operating system's language may not be set to English(US). Try switching to this.

Symptom Communications routines return "True", but don't really work.
Cause **SimulateCommunications** may be set to "True"

Symptom Unable to register **PTALKDT.OCX**.
Cause **PTALKDT.OCX** cannot access some DLL's or DLL's of the correct version.

a. Make sure PMAC.DLL is in the SYSTEM directory
b. Look at the supplied installation script, and check it's accuracy

Dual Ported Ram Automatic Feature Example

The example below illustrates how to make use of PMAC's automatic Dual Ported Ram features. In this case were using the "Fixed Real Time Data Buffer" which has motor specific information. All 8 motor actual positions are being displayed using a timer procedure. The example was done in Delphi.

```
  procedure TForm1.Timer2Timer(Sender: TObject);
var
   aBool  : Bool;
   aShort : short;
   aString: string[100];
   LongLow: LongInt;
   LongHigh: LongInt;
   position: double;
begin
    // Tell PMAC we are busy reading, Y:$D009,  89 = "Y" in ASCII
   aBool := Form1.PTalkDTCtrl1.DPRSetWord(89,9,1);
   // Read in servo timer,  X:$D009,  88 = "X" in ASCII
   aShort := Form1.PTalkDTCtrl1.DPRGetWord(88,9);
   aShort := aShort and $7FFF;// Bit 15 is a handshake bit, mask off
   Str(aShort, aString);
   Edit13.Text := aString;
   // Read in Motor Actual Positions, 2 long words that need to be
   // converted to a float via a special method
   LongLow := Form1.PTalkDTCtrl1.DPRGetDWord(20);
   LongHigh := Form1.PTalkDTCtrl1.DPRGetDWord(21);
   position := Form1.PTalkDTCtrl1.DPRFixed(LongLow,LongHigh);
   position := position/(32*96); // Ix08 *32 scale factor
   eM1.Text := FloatToStr(position);
```

```
    LongLow := Form1.PTalkDTCtrl1.DPRGetDWord(35);
    LongHigh := Form1.PTalkDTCtrl1.DPRGetDWord(36);
    position := Form1.PTalkDTCtrl1.DPRFixed(LongLow,LongHigh);
    position := position/(32*96); // Ix08 *32 scale factor
    eM2.Text := FloatToStr(position);
    LongLow := Form1.PTalkDTCtrl1.DPRGetDWord(50);
    LongHigh := Form1.PTalkDTCtrl1.DPRGetDWord(51);
    position := Form1.PTalkDTCtrl1.DPRFixed(LongLow,LongHigh);
    position := position/(32*96); // Ix08 *32 scale factor
    eM3.Text := FloatToStr(position);
    LongLow := Form1.PTalkDTCtrl1.DPRGetDWord(65);
    LongHigh := Form1.PTalkDTCtrl1.DPRGetDWord(66);
    position := Form1.PTalkDTCtrl1.DPRFixed(LongLow,LongHigh);
    position := position/(32*96); // Ix08 *32 scale factor
    eM4.Text := FloatToStr(position);
    LongLow := Form1.PTalkDTCtrl1.DPRGetDWord(80);
    LongHigh := Form1.PTalkDTCtrl1.DPRGetDWord(81);
    position := Form1.PTalkDTCtrl1.DPRFixed(LongLow,LongHigh);
    position := position/(32*96); // Ix08 *32 scale factor
    eM5.Text := FloatToStr(position);
    LongLow := Form1.PTalkDTCtrl1.DPRGetDWord(95);
    LongHigh := Form1.PTalkDTCtrl1.DPRGetDWord(96);
    position := Form1.PTalkDTCtrl1.DPRFixed(LongLow,LongHigh);
    position := position/(32*96); // Ix08 *32 scale factor
    eM6.Text := FloatToStr(position);
    LongLow := Form1.PTalkDTCtrl1.DPRGetDWord(110);
    LongHigh := Form1.PTalkDTCtrl1.DPRGetDWord(111);
    position := Form1.PTalkDTCtrl1.DPRFixed(LongLow,LongHigh);
    position := position/(32*96); // Ix08 *32 scale factor
    eM7.Text := FloatToStr(position);
    LongLow := Form1.PTalkDTCtrl1.DPRGetDWord(125);
    LongHigh := Form1.PTalkDTCtrl1.DPRGetDWord(126);
    position := Form1.PTalkDTCtrl1.DPRFixed(LongLow,LongHigh);
    position := position/(32*96); // Ix08 *32 scale factor
    eM8.Text := FloatToStr(position);

     // Tell PMAC we are not busy anymore
    aBool := Form1.PTalkDTCtrl1.DPRSetWord(89,9,0);
end;
```

DELTA TAU DRIVER CONFIGURATION

A Global View of the Driver

Delta Tau's 32-bit hardware driver itself consists of three files.

- PCOMM32.DLL - A 32-bit DLL.
- PMAC**ISA**(**SER**, **PCI**, or **USB**).SYS – Windows 98/ME or NT 2000 kernel drivers.
- PMAC**ISA**(**SER**, **PCI**, or **USB**).INF - Windows Setup Information files.

The illustration below shows how these modules are related.

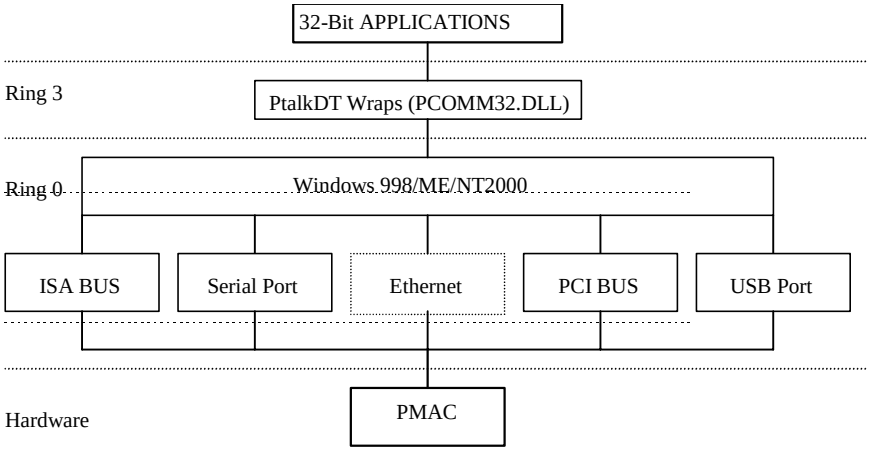


Figure 1-1. PComm32 Driver Structure

Device Configuration

Plug & Play Ports

- PCI BUS PMAC
- USB Port PMAC

Non-Plug & Play Ports

- ISA Bus PMAC
- Serial Port PMAC
- Ethernet port PMAC (expected in 2nd trimester of 2001)

Device Configuration

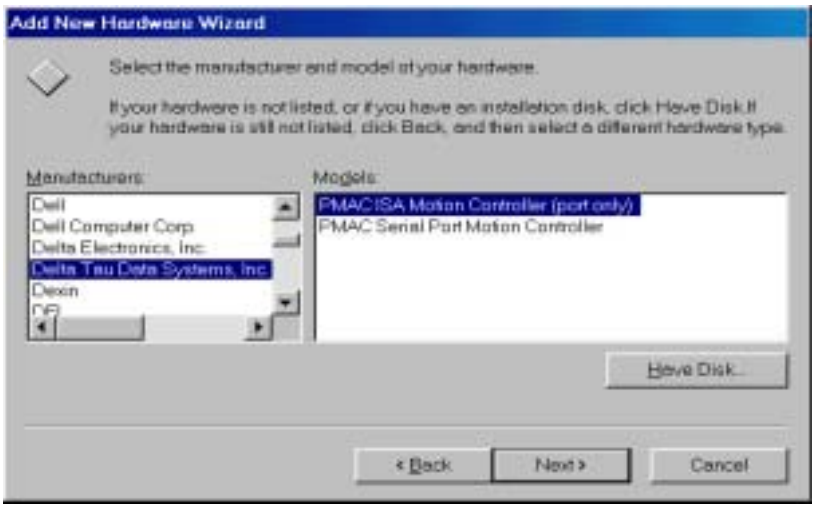
- **Plug & play devices** are configured automatically at boot time or whenever plugged in (USB device.) Devices can be reconfigured at any time for updated drivers as well.
- **Non-plug & play devices** are configured through windows standard "hardware wizard" as follows:

- In Windows 98/ME/2000, run **Add new hardware** from the control panel. Hardware wizard in listed under System in Windows 2000.

- Continue for a couple of screens until following screen appears. Select **No** from the Windows auto search option.

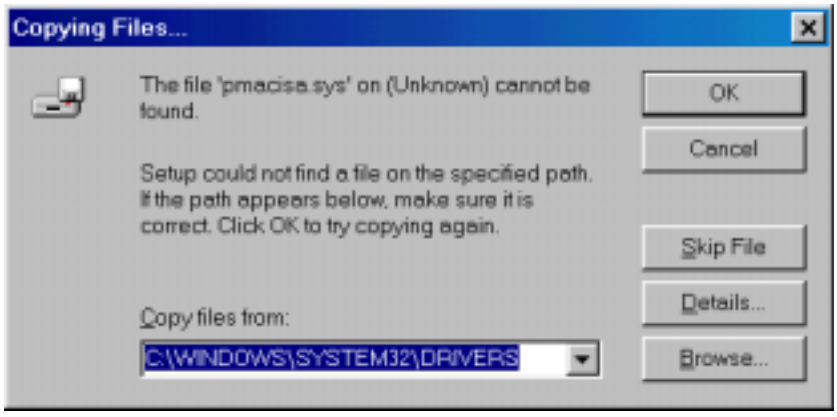
- In the following screen, select **other devices** from the hardware types (First time only.) Next time, Motion (Motion Controllers under Windows 2000) will be listed in the hardware types list.

- Select **have disk** from the following wizard and specify the path of **Setup information** file(s) **PMACISA.Inf** or **PMACSER.Inf** depending upon the desired configuration, ISA bus or Serial respectively. Setup information file(s) are located in **Windows\INF** folder (**Winnt\INF** folder under Windows 2000.)



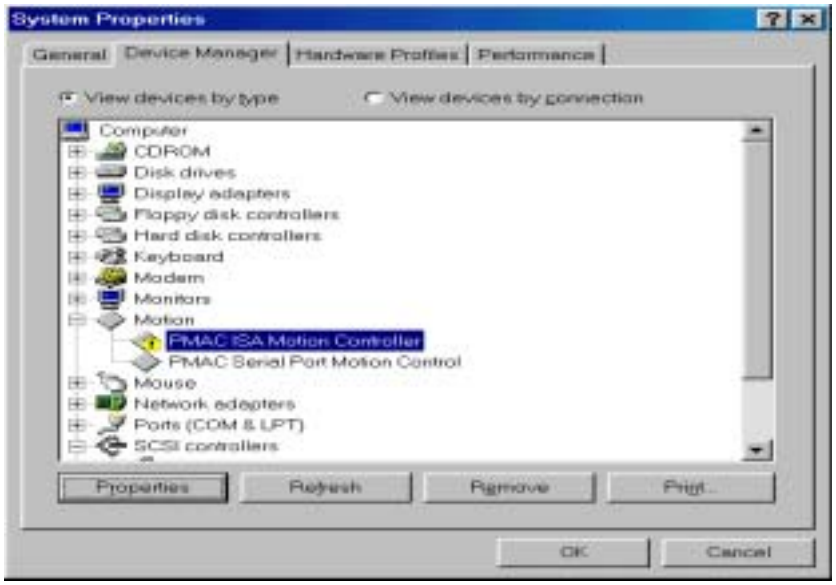
Once hardware is identified from the setup information file(s) select the appropriate model from PMAC ISA or PMAC Serial Port controller.

- Setup information file(s) specify the required **driver file(s) PMACISA.SYS or PMACSER.SYS** for Bus and Serial configuration(s) respectively. In the following wizard specify the path of driver files. Pcomm32 installation copies these files in **Windows\System32\Drivers** folder (**Winnt\system32\Drivers** folder under Windows 2000.)



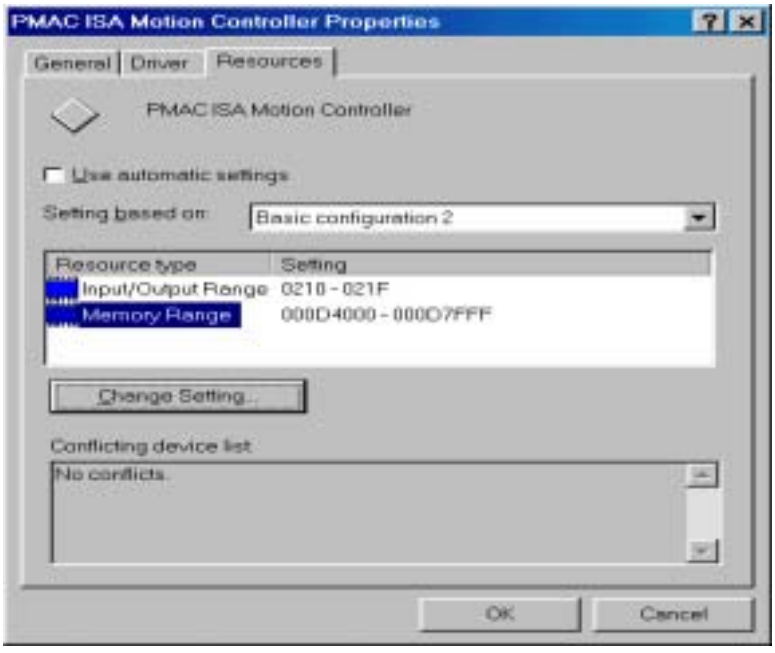
At this stage the driver is installed on your computer. Following step is different on Windows 98/ME than Windows 2000. Please configure the resources accordingly. **A restart of computer is required after the driver installation before use.**

- (**Windows 98/ME only**) There are essentially FOUR configurations available for ISA BUS and only one for the Serial port. For ISA BUS, they are **I/O Port** only, **I/O Port w/DPRAM, I/O Port w/DPRAM & IRQ** and finally **I/O Port w/IRQ** only. PMAC can be configured for only one of these configurations depending on the DPRAM availability, I/O port and/or IRQ jumper settings. Under Windows 98/ME. These resources can only be changed from Control Panel's device manager.



Device manager is invoked by Control Panel's System menu or directly by checking the properties of My Computer from the Desktop. An extra restart may be required if the resources are changed.

- (**Windows 2000 only**) Above step is not needed under Windows 2000 where all resource configurations are available during installation.
- However, resources can be changed on all compatible operating systems, at any time by invoking the device manager.



This driver has eliminated the need for MotionEXE (available in earlier implementations of Delta Tau's 32 bit driver). Once a device is configured successfully, it is registered and available for use.

Parameter configuration of serial device such as, port number, baud rate, timeouts, handshake and parity options are done at the application level. Properties of a serial device are enabled in the SelectDevice() dialog.

After Setting Up The Device Driver

Once configured, PMAC devices are listed under device manager in the computer's system information page. All configured devices (plug & play as well as non-plug & play) are registered and therefore available for use. All available devices are listed upon one simple SelectDevice() method call from PtalkDT Pro.

Enhanced Features

- **Fast serial communication.** The Ring 0 driver has eliminated the need for secondary server (Serserver or Comserver) hence reducing the overhead caused by these applications. This, in turn, reduced the unnecessary overhead and therefore increased the serial port throughput tremendously. At least five times faster serial communication is achieved with this technique. It is expected to improve further once port timeouts are optimized.
- **Rearrangement of devices without restarting computer.** Since the devices are configures through Windows' device manager, and since MotionEXE is no longer needed, they can easily be rearranged by the SelectDevice call to "PComm32" library.
- **Multiple accessibility of any port.** The global data memory register keeps necessary information about the hardware, that is, PmacType, Location, Enumeration etc. All applications then get the information from global data register. Global data further keeps track of user count and therefore reduces overhead for reopening the device. This allows multiple access to any port for multiple applications.

Supported Operating Systems

- **Windows 98**
- **Windows ME**
- **Windows 2000**

GLOSSARY OF TERMS

directive
An instruction that tells the downloader how to process this or the upcoming lines of a file.

preprocess
The act of parsing a file and executing all the downloader directives in preparation for downloading the file to PMAC.

event
A function that is automatically called when a certain condition(s) occur.

property
An attribute (or variable) of an OCX control that configures, enables, or disables a certain feature of the control.

DPRAM
This stands for dual port RAM. This hardware option of PMAC allows you to share memory between PMAC and the host computer. DPRAM is useful for high-speed communications and data exchange between PMAC and the host computer

upload
This is the process of transferring information, usually program files and data, from the PMAC to the host computer.

download
This is the process of sending information, usually program files and data, from the host computer to PMAC.

methods
All featured functions in an OCX are referred to as methods. Methods give the OCX its capabilities.

PMAC
The motion computer from Delta Tau Data Systems. PMAC stands for Programmable Multi-Axis Controller.

MMI
This stands for Man Machine Interface. An MMI is the software that is used by a machine user to operate a machine. It is the software on the host computer that the operator uses to control the machine.

OCX control
This collection of library functions is designed to make difficult programming tasks easy. OCX controls are the latest addition to Microsoft's OLE 2.0. They are sometimes referred to as reusable components. OCX controls are improved and enhanced VBXs.

PTalkDT
PTalkDT is a communications OCX control designed to communicate to Delta Tau's PMAC.

INDEX

Download Directives
#define name *38*
#define name {command or variable} *38*
#else *38*
#endif *38*
#ifdef name *38*
#ifndef name *38*
#include "filename" *38*

Global View of the Library 53

installation 5

Methods
DownloadFile *1, 17*
DPRFixed *39*
DPRWord *43*
Flush *1, 17*
GetControlResponse *1, 17, 44*
GetLineAck *2*
GetLineCR *2*
GetResponse *2, 17, 45*
LoadSettings *2, 17*
SaveSettings *2, 17*
SendChar *2, 17*
ShowPropertyPage *2, 17, 48*
UploadData *2, 17*

Properties
Enabled *16*

Usage of PComm32 57