

Copyright Information

© 2003 Delta Tau Data Systems, Inc. All rights reserved.

This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

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 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 ACTIVE X 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.....	3
HOW DO I GET SUPPORT?	3
INSTALLING/UNINSTALLING PTALKDT	4
BEFORE YOU RUN SETUP.....	4
RUNNING SETUP	4
WHAT WAS INSTALLED?	4
<i>Uninstalling PTalkDT OCX</i>	6
HOW TO DESIGN WITH PTALKDT	7
IN DESIGN MODE	7
RUN TIME MODE	8
<i>Distributing Your Final Application</i>	9
ALTERING, SAVING AND RETRIEVING PTALKDT SETTINGS AT RUN TIME	10
<i>Communication Settings</i>	10
<i>General Settings</i>	10
YOUR FIRST VISUAL BASIC MMI WITH PTALKDT	12
OVERVIEW	12
<i>Instructions</i>	12
YOUR FIRST MICROSOFT VISUAL C++ MMI WITH PTALKDT	16
OVERVIEW	16
<i>Instructions</i>	16
PTALKDT REFERENCE	29
DOCUMENTATION CONVENTIONS	29
OVERVIEW	29
PTALKDT PROPERTIES	30
<i>Enabled</i>	30
<i>LastError</i>	30
<i>LastErrorString</i>	30
<i>DeviceNumber</i>	31
<i>DownloadDeleteTemp</i>	32
<i>DownloadDo</i>	32
<i>DownloadHide</i>	33
<i>DownloadLog</i>	33
<i>DownloadMap</i>	34
<i>DownloadMaxErrors</i>	34
<i>DownloadParse</i>	35
<i>DownloadShowErrors</i>	35
<i>UploadAppend</i>	36
<i>UploadHide</i>	36
<i>UploadNoComments</i>	37
<i>UploadShowProgress</i>	37
PTALKDT METHODS	38
<i>DPRAvailable()</i>	38

<i>DownloadFile (file name)</i>	38
<i>DPRDouble (LSB_word, MSB_word)</i>	40
<i>DPRFixed (LSB_word, MSB_word)</i>	41
<i>DPRDWordBit Set/Reset and BitSet Methods</i>	41
<i>DPRGetDWord and DPRSetDWord Methods</i>	42
<i>DPRGetFloat and DPRSetFloat Methods</i>	42
<i>DPRGetWord and DPRSetWord Methods</i>	43
<i>Flush ()</i>	45
<i>GetControlResponse (response, control char)</i>	45
<i>GetLineAck (response)</i>	45
<i>GetLineCR (response)</i>	46
<i>GetResponse (response, command)</i>	46
<i>IsLineWaiting ()</i>	46
<i>LoadSettings ()</i>	47
<i>LockPMAC ()</i>	47
<i>ReleasePMAC()</i>	48
<i>SaveSettings ()</i>	49
<i>SendChar (character)</i>	49
<i>SendLine(Command)</i>	49
<i>ShowPropertyPage ()</i>	51
<i>UploadData (file name, command, options, expected number of lines)</i>	51
PTALKDT EVENTS	52
<i>OnError</i>	52
<i>Trouble Shooting</i>	52
DUAL PORTED RAM AUTOMATIC FEATURE EXAMPLE.....	53
GLOSSARY OF TERMS	57
INDEX	59

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 wont be effected. 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 <i>download</i> a text file or multiple text files to PMAC. A powerful string substitution preprocessor is included.
---------------------	---

Flush	A useful method to <i>clear out</i> 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.
ShowProperty Page	Shows PTalkDT's <i>property page</i> for modifying the communications settings.
SaveSettings	Stores PTalkDT communications settings to disk.
UploadData	This allows you to <i>upload</i> 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 95, Windows NT
- 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 are the quickest, include your **REGISTRATION NUMBER!**).

INSTALLING/UNINSTALLING PTALKDT

Before You Run Setup

Please take a few minutes before you install PTalkDT to do the following:

Read 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.

Visual Basic users should install Visual Basic BEFORE PTalkDT.

Running Setup

When you run the setup program to install PTalkDT on your computer, you will be able to specify where on your hard drive to install PTalkDT. It is preferred to install it in the suggested directory for consistency.

Windows 95

To run the setup program in Windows 95 click the *Start* button from the taskbar and select the Run... menu option. Insert the first distribution diskette labeled "Disk 1" and type

A:setup

and press ENTER. Follow the installation instructions on the screen. If you encounter any problems, please refer to the Troubleshooting Guide in the Appendix of this manual.

Windows NT

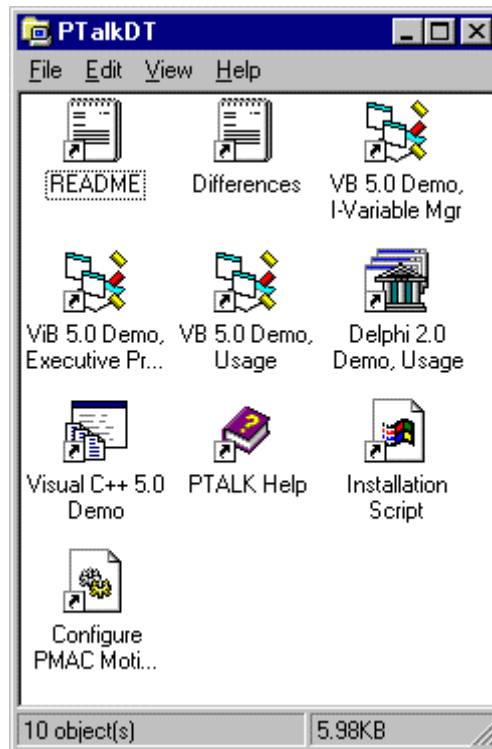
To run the setup program in Windows NT, from the Program Manager select the *File* menu and select the Run... menu option. Insert the first distribution diskette labeled "Disk 1" and type

A:setup

and press ENTER. Follow the installation instructions on the screen. If you encounter any problems, please refer to the Troubleshooting Guide in the Appendix of this manual.

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, one Delphi and one Visual C++ demo project icons. The new program group will look something like this:



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.

The “Installation Script” is a text based script file used to create the installation you just used. This may come in handy when it comes time for your own.

The “Configure PMAC Motion Applet” icon is 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 (i.e. assign the **DeviceNumber** property).

We encourage you to run the Visual Basic and/or Delphi example projects. Please note that these will only work if you have the corresponding development environment. If you encounter problems trying to communicate to your PMAC, run the **MOTION** applet that is located in your PTalk installation directory. After you have tried the example projects, try and 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 programs code](#) that is provided.

Note

In case you are interested, 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.

Uninstalling PTalkDT OCX

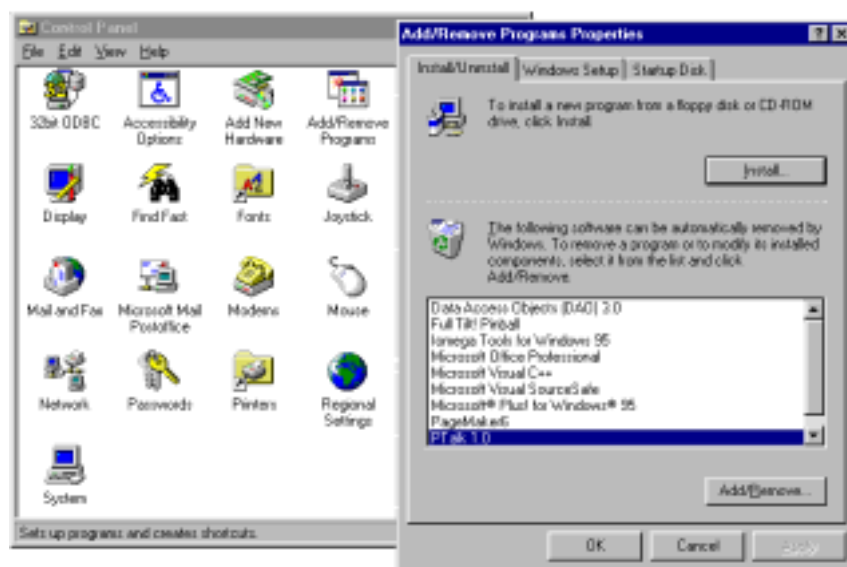
It is highly suggested that you uninstall PTalkDT before upgrading to a newer version of the product.

Windows 95

To uninstall PTalkDT, from Windows 95 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.

Windows NT

In NT 3.51 there will be an "Uninstall" Icon in the program group created for PTalkDT during installation. Double click the "Uninstall" Icon to uninstall PTalkDT. In NT 4.0 use the 95 procedure described above.

HOW TO DESIGN WITH PTALKDT

In Design Mode

First, configure your PMAC(s) in your system by running the “**MOTION**” applet located in your PTalk installation directory.

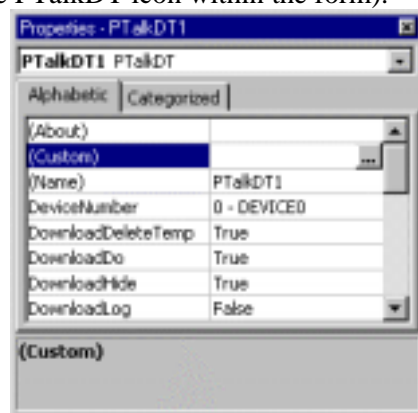
For most of the remainder of this manual, all examples will be described assuming you are using something similar to Visual Basic. If you are using a different development environment, the procedures described here will be analogous.

First add the PTalkDT control to your development environments tool box. 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.

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 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 **SimulateCommunication** 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”. Remember, the **MOTION** applet in the operating systems **CONTROL PANEL** will allow you to add/remove or configure any PMAC(s) in your computer.

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.
SimulateCommunication	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 **SimulateCommunication** 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 **SimulateCommunication** 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.
ShowPropertyPage()	Show PTalkDT’s property page.
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 intended to be 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.

Distributing Your Final Application

When you create and distribute applications that use the PTalkDT control, you need to install the appropriate files on the target computer and register the OCX with the operating system (The Setup Kit included with Visual Basic, or Installshield provide tools to help you write setup programs that install your applications correctly). Also, the “Installshield” text based script file used to create PTalkDT installation is located in the installation directory (look for the *.IWZ file extension) for your viewing pleasure.

For 95 or NT

Intall the **MOTION.EXE** control panel applet as well as the following files to the operating systems “SYSTEM” directory:
MFC40.DLL

MSVCRT40.DLL
MSVCRT.DLL
MSVCIRT.DLL
MFC42.DLL

95 Users

Be sure to add the **PTalkDT.OCX**, **PMAC.DLL**, and **PMAC.VXD** files to the **\WINDOWS\SYSTEM** directory.

NT Users

Be sure to add the **PTalkDT.OCX**, and **PMAC.DLL** to the **\WINDOWS\SYSTEM32** directory. Also, put the **PMAC.SYS** file in the **\WINDOWS\SYSTEM32\DRIVERS** subdirectory.

Notes

1. If you get a message similar to the following "Unable to start supporting device driver." from the operating system you most likely do not have the PTalkDT files correctly located.
2. Registering ActiveX's can also be done with a utility often shipped with the operating system. For example "REGSVR32.EXE" in Windows 95 can do this from the command line (i.e., REGSVR32.EXE PTALKDT.OCX).

Altering, Saving and Retrieving PTalkDT Settings At Run Time

Communication Settings

When you configure the PTalkDT to communicate (i.e. by using the **CONTROL PANEL**'s **MOTION** applet) to PMAC the settings are saved in the operating systems registry under:

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**, **SimulateCommunications** 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

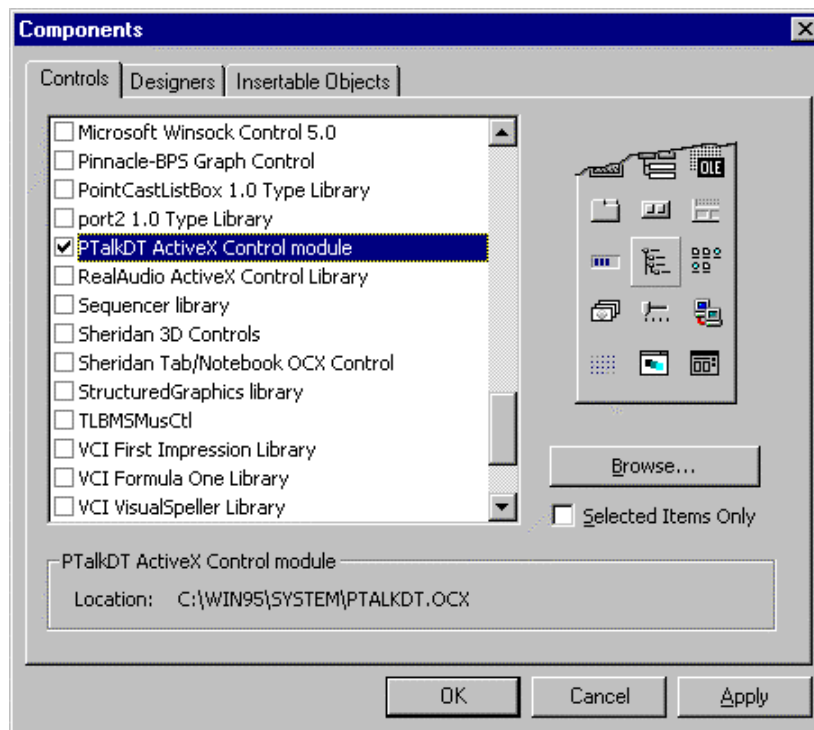
YOUR FIRST VISUAL BASIC MMI WITH PTALKDT

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

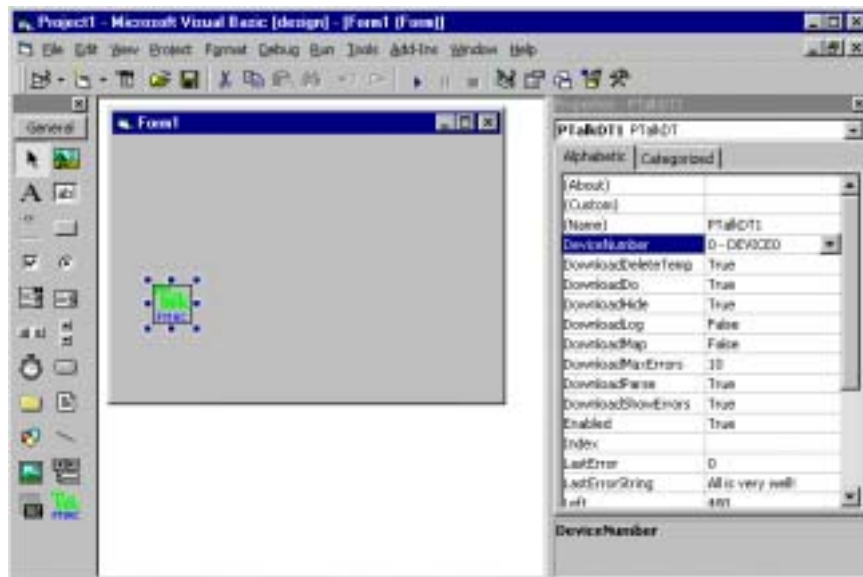
1. Start Visual Basic 5.0 and choose "Standard EXE" for project type.
2. Choose *Project* from the top menu bar and select Components. Select the "PTalkDT Control" module and then select the *OK* button.



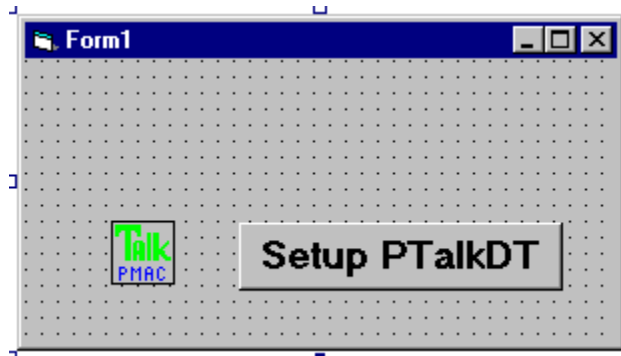
The PTalkDT icon  should appear at the bottom of your tool palette:



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.



- 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".



- 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.ShowPropertyPage  
    PTalkDT1.SaveSettings  
End Sub
```

This code will call PTalkDT's **ShowPropertyPage()** and **SaveSettings()** 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.

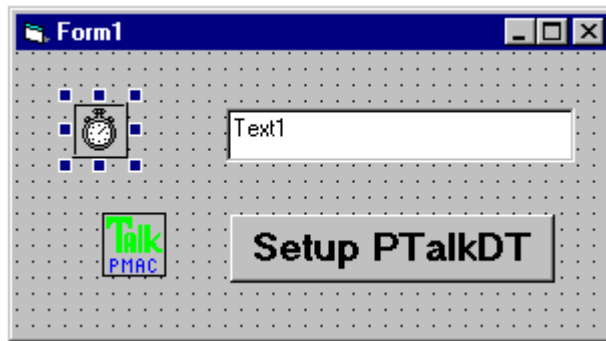
- Now put the **LoadSettings()** method in the Form_Load() method of the form by double clicking on any "free" spot within the form. The routine should look like so when done:

```
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.

- Next lets add real time display of PMAC's servo clock. Add a text control and a timer control to the form.
- Press F4 to view the timer's property window.

10. 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.

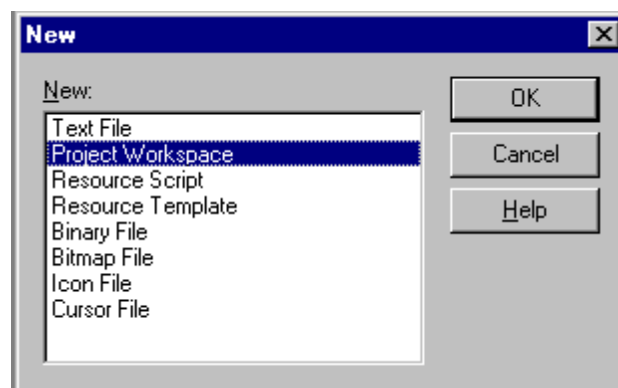
YOUR FIRST MICROSOFT VISUAL C++ MMI WITH PTALKDT

Overview

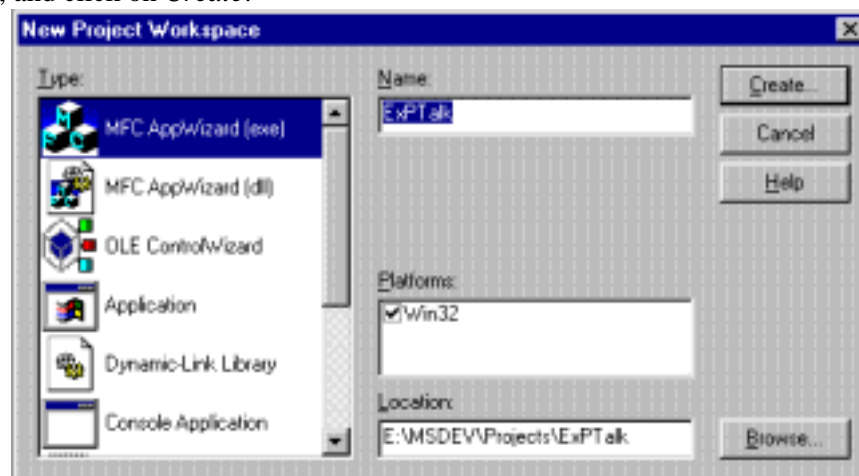
This section will guide you through building a simple Microsoft Visual C++ MMI (man-machine interface) application using PTalkDT. The resulting application displays the value of PMAC's constantly changing servo counter register. The code generated here can be similarly constructed with other development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

Instructions

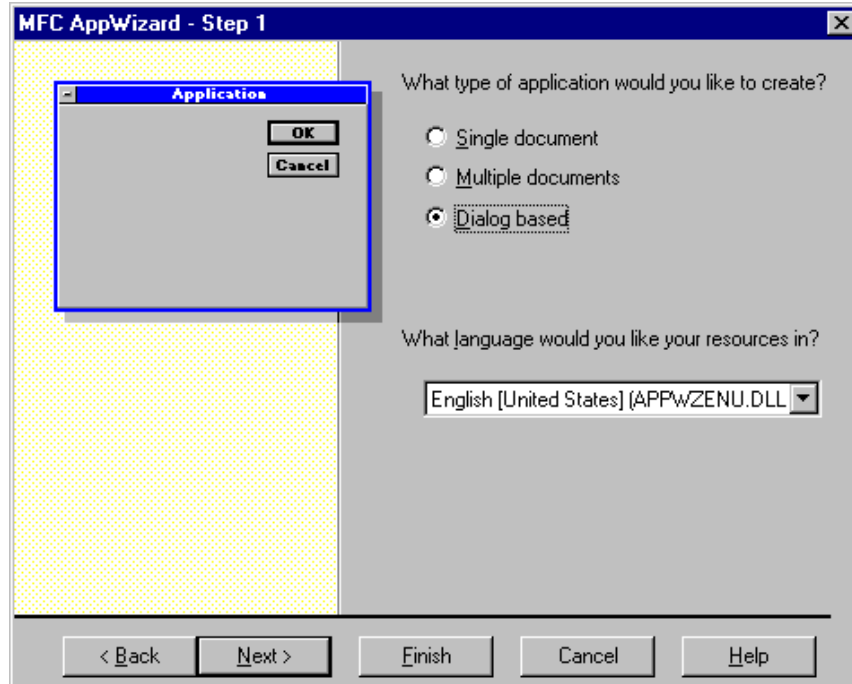
1. Start Visual C++.
2. Choose FILE from the top menu bar and select New. *Highlight Project Workspace* from the list box and then select the *OK* button.



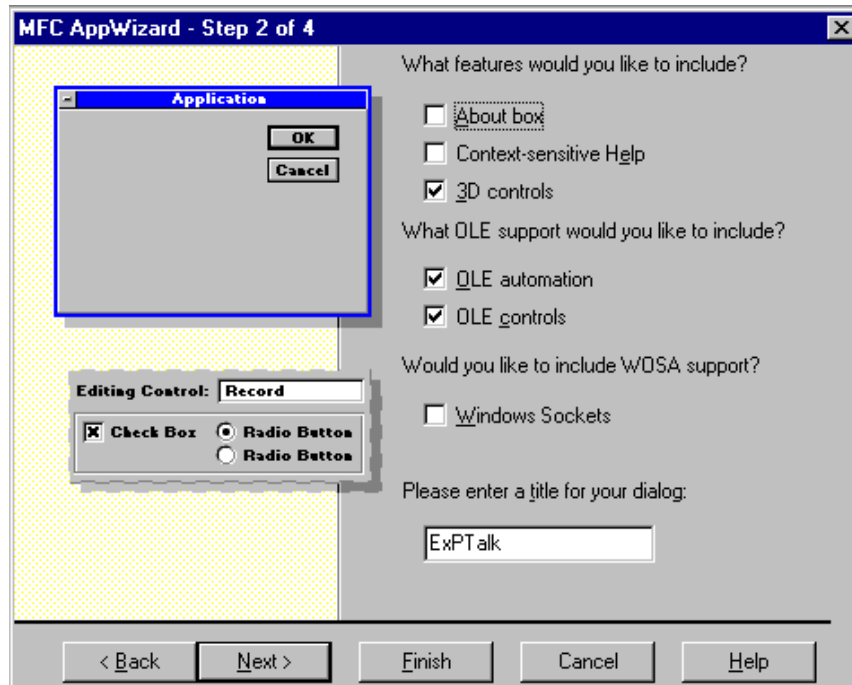
3. In the next dialog box, select *MFC AppWizard (exe)* from the list box, type in a project name (such as **ExPTalk**), and click on *Create*:



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



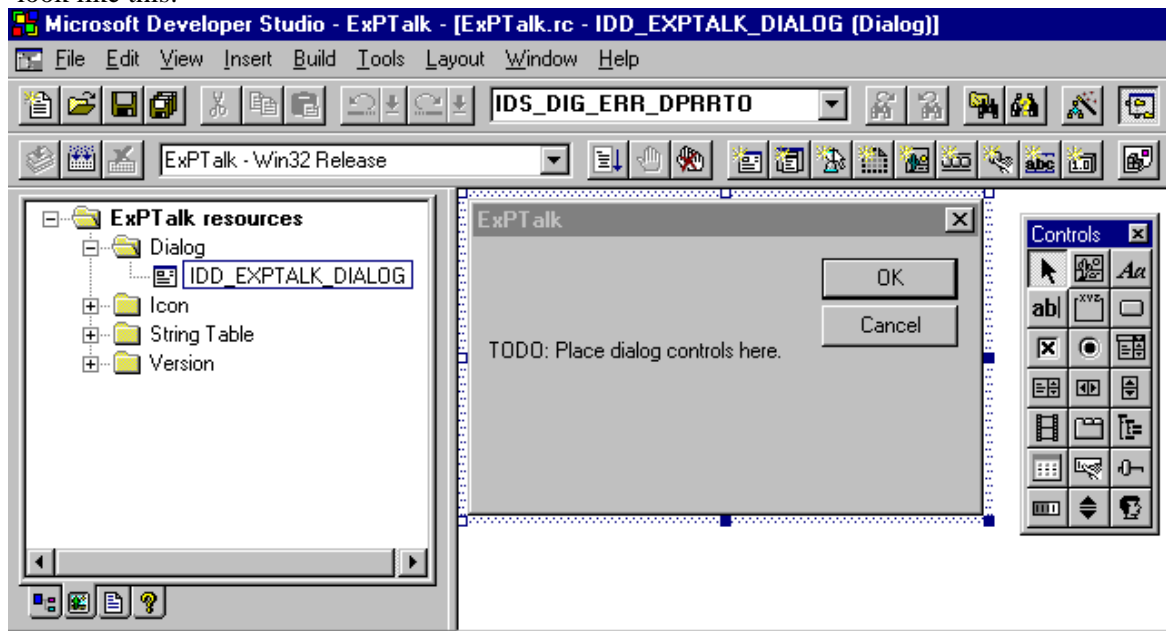
- On the last dialog box, place a check mark for *3D controls*, *OLE automation*, and *OLE controls* and click on *Finish*:



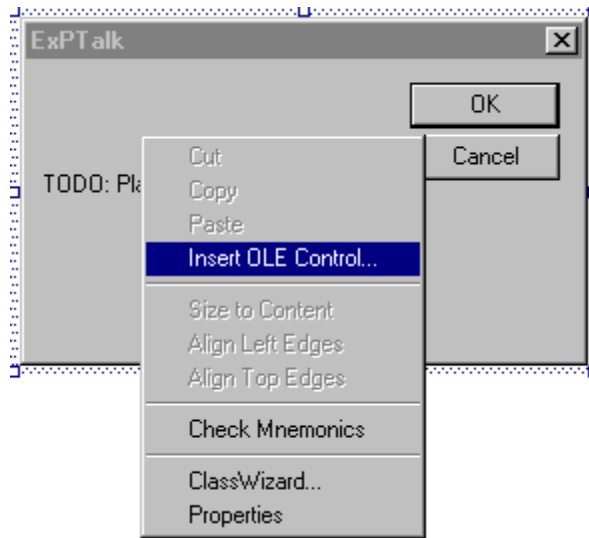
At this point, a set of C++ files have been generated in a directory with the same name as the project name you selected. Go ahead and compile this newly created project and run it to verify it works correctly. When you execute this program, a blank dialog box with an *OK* and *CANCEL* button should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



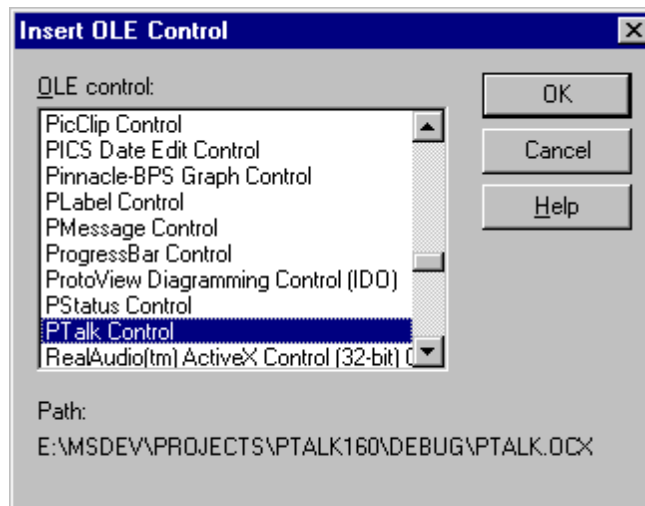
Now, let's go back and add the PTalkDT control to this dialog box. From within the Visual C++ workspace environment, select to view the existing resources (which were created by the AppWizard in the previous steps) and click on the *Dialog* resource. Your screen should look like this:



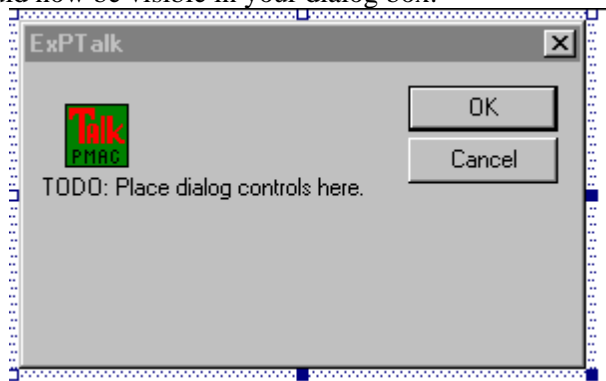
6. With your mouse pointing to the dialog box (on the right, called “ExPTalk”), click the *right* mouse button to expose the following pop-up menu and select *Insert OLE Control...*:



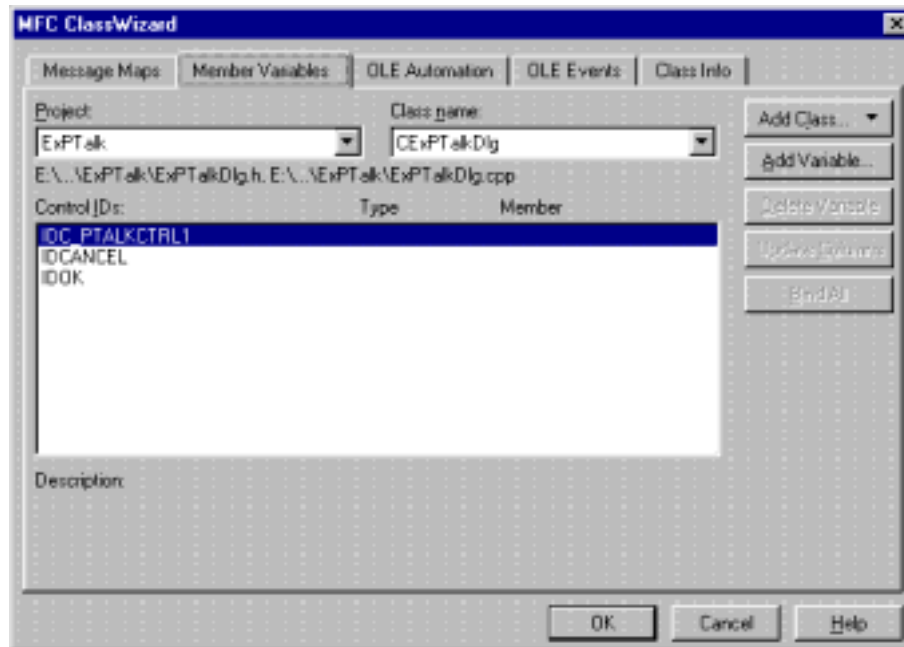
7. A new dialog box will appear containing a list of available controls. Scroll down and choose the control called PTalkDT Control and then click *OK*:



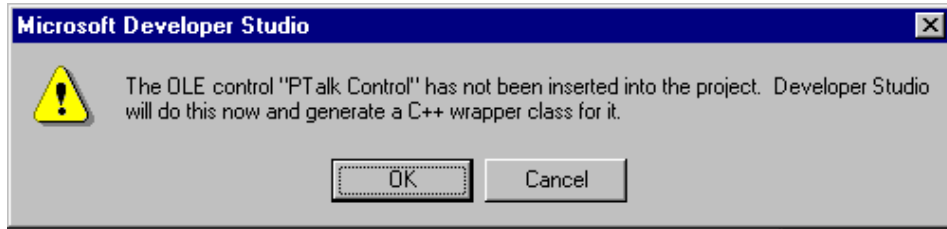
The PTalkDT control should now be visible in your dialog box:



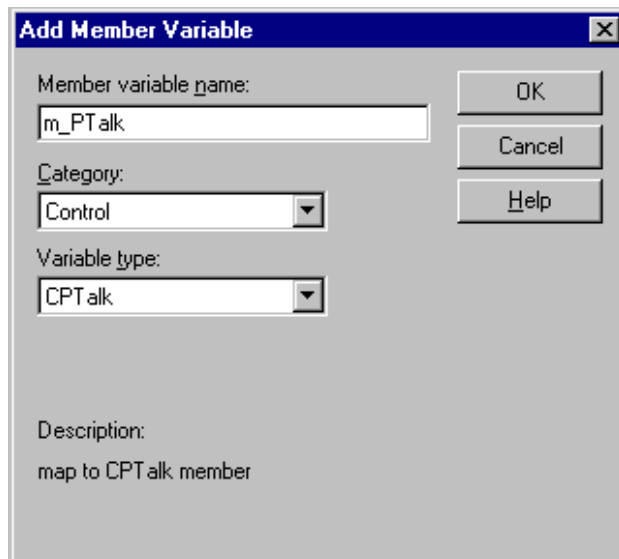
8. Our next step is to use the MFC ClassWizard within Visual C++ to generate code that will create a control class for this newly added PTalkDT control. To do this, select the View menu and then *ClassWizard*. The MFC ClassWizard dialog box will appear. Select the *Member Variables* tab. Your screen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



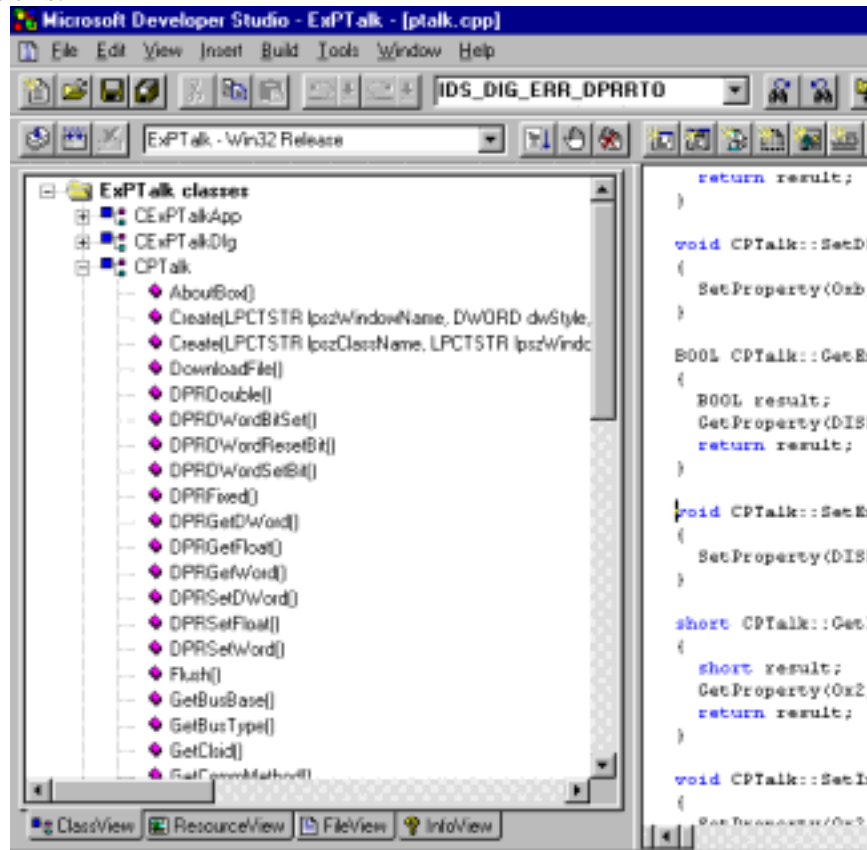
Highlight IDC_PTALKCTRL1 and press *Add Variable*. When you do this, the following dialog box will appear:



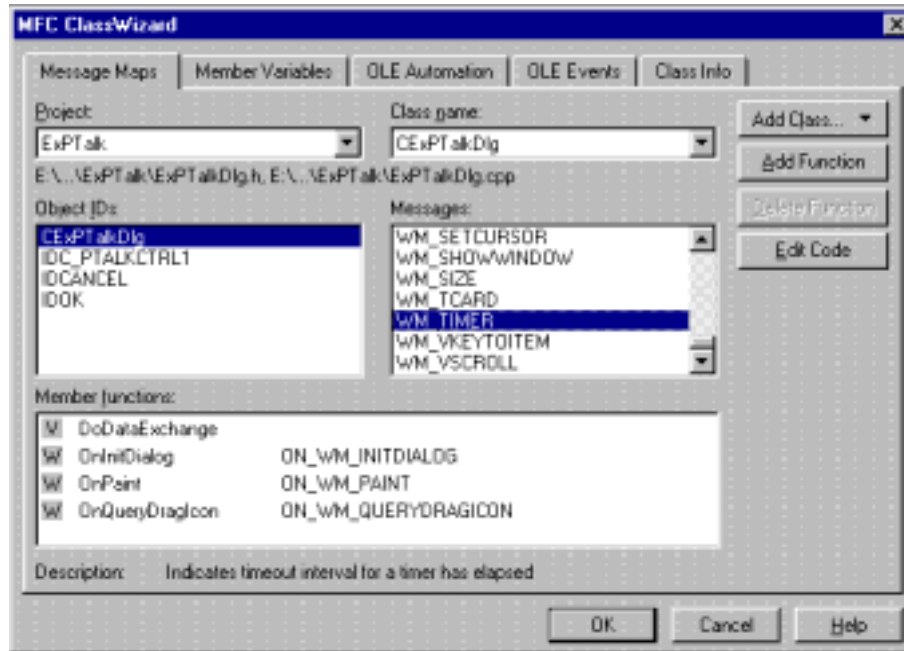
9. Select *OK*. On the next dialog box, select *OK* again.
10. The next dialog box will ask you to type in a name for the variable that will be used to access all of PTalkDT's properties and methods in your C++ code. Use the name shown on the next page and click on *OK*:



Click on *OK* again. At this point, the MFC ClassWizard has generated a new C++ file and header file which contains the code to allow you 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:

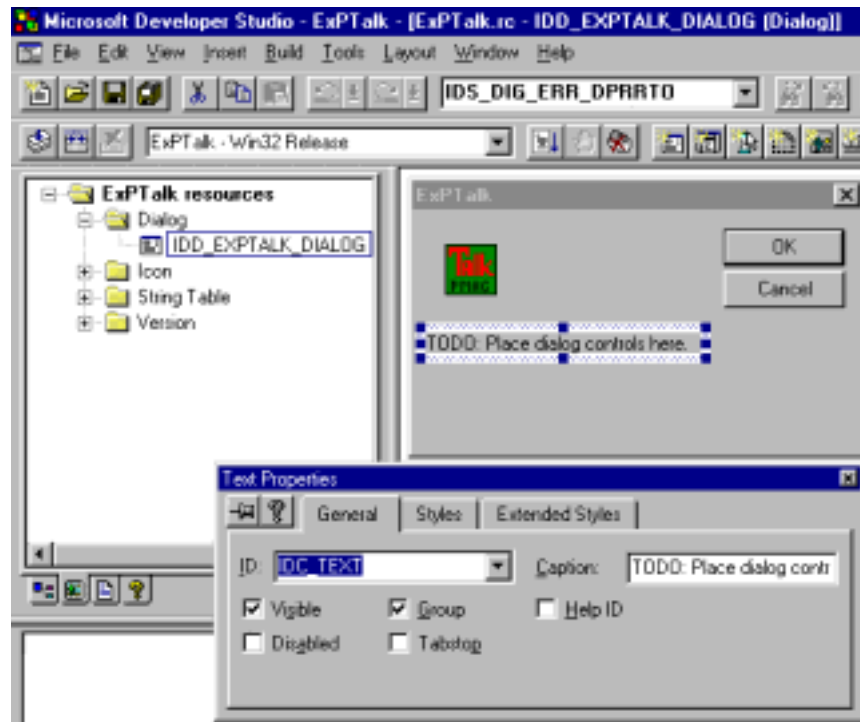


11. We will now add a timer function to our dialog box which will use PTalkDT to continuously query PMAC for information. We will use the MFC ClassWizard again to do this. Select the View menu and then *ClassWizard*. The MFC ClassWizard dialog box will appear. Select the *Message Maps* tab and locate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

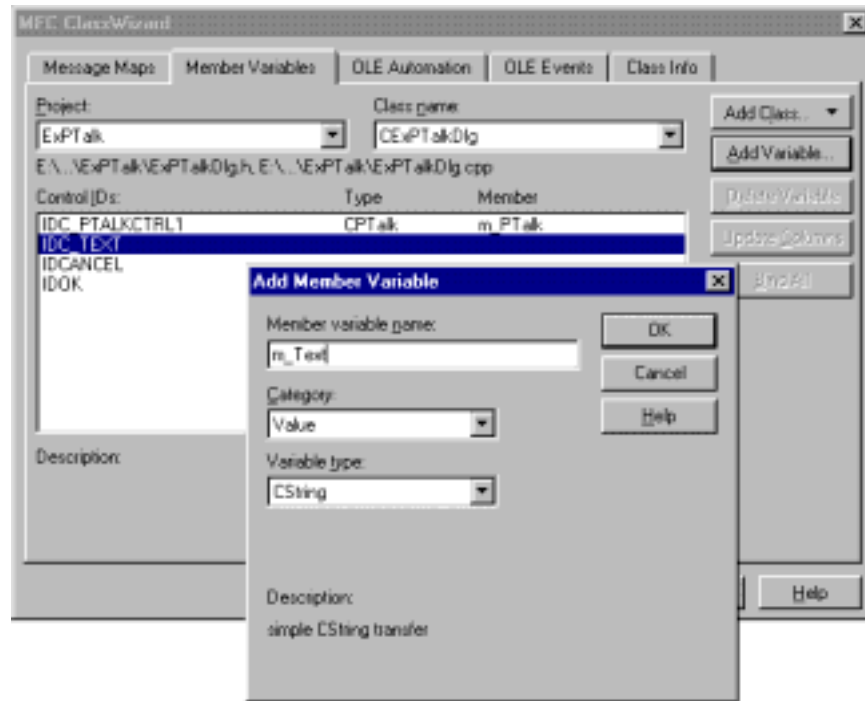


A new function for the timer has now been created. We will add code to this function later on.

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 ClassWizard again to create a usable variable so that we may access this static text in our code. Select the View menu and then *ClassWizard*. The MFC ClassWizard dialog box will appear. Select the *Member Variables* tab and 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>

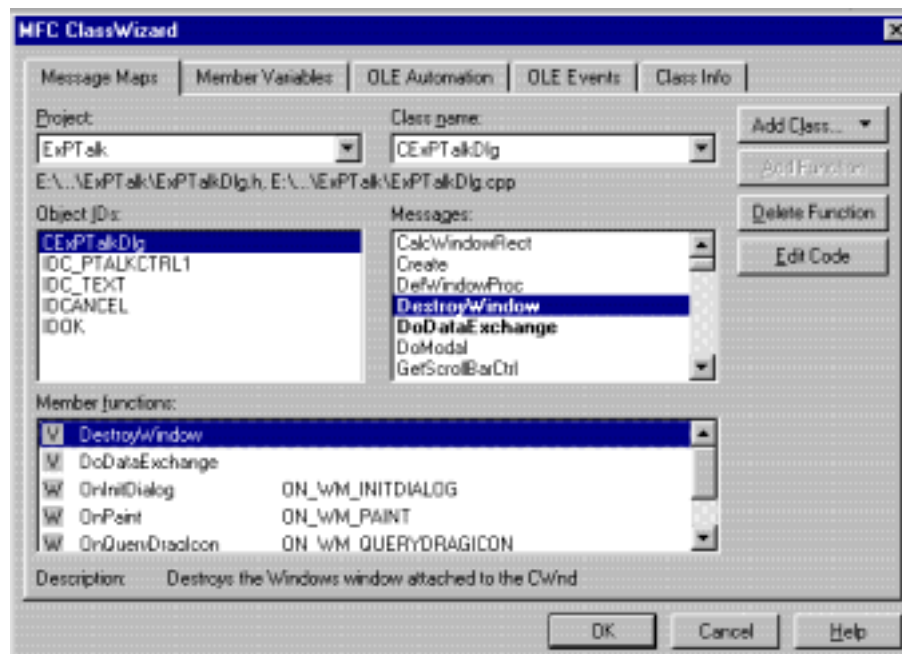
It should look like this after:

```

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

```

- 15) We must use the MFC ClassWizard one last time to create one last function. Select the View menu and then ClassWizard. The MFC ClassWizard 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();
}

```

}

12. 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 determine if 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 **ShowPropertyPage()** 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 error 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

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”, “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 **DownloadFile()** 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

Description

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 will be 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. Of course 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", "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 DownloadFile() 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.

Extensive use of macro string substitution can provide very readable code with little need for comments. The following table describes all the preprocessor directives.

	Example	Description
--	---------	-------------

#define <i>name</i> { <i>command or variable</i> }	#define COUNTER P1	Declares the name of a macro string substitution. For every occurrence of <i>name</i> , the preprocessor will substitute in { <i>command or variable</i> }.
	#define DEBUG_MODE	Declares a variable name that can be used for compiler directives.
#include " <i>filename</i> "	#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 DEBUG_MODE ... #else ...; (<i>this code ignored</i>) #endif	Tests to see if <i>name</i> has been previously declared. If so, the subsequent lines of code are included in the download.
	#ifndef DEBUG_MODE ... #else ...; (<i>this code ignored</i>) #endif	Tests to see if <i>name</i> has NOT been previously declared. If <i>name</i> has NOT been declared, the subsequent lines of code (until the next #else or #endif) are included in the download.
	#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.
	#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.

```
[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 Port 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;
```

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 Ram. **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" can not 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

Control-T will not get sent by this function. 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 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 successfull, 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

UploadAppend

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 ()

Description

Displays a dialog box for modifying PTalkDT's properties.

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

Displays a user-friendly dialog for modifying PTalkDT's at run time. This is the same dialog shown when the *custom* property is double clicked from within Visual Basic's property window.

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	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 <i>number_of_lines</i> . Also, 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 trouble shooting. 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	Cause
PTalkDT can't seem to load or fails unpredictably.	Visual Basic users should be sure to install Visual Basic first then PTalkDT second.
You can't establish serial communications but everything works O.K. once you run the PMAC	Some PMAC firmware versions (before 1.16A) set the hardware handshaking lines incorrectly on

Executive Program.	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.
You can't establish serial communications period.	Are you using a known working serial cable? You may just want to see exactly what your PMAC's baudrate 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. Check the port setup from the operating systems control panel. Also, try the supplied "HyperTerminal" application.
Serial communications is losing characters.	Set up 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.
In Microsoft Visual C++ after inserting a PTalkDT control, you can't see any of the member variables displayed in the class wizard.	The operating system's language may not be set to English (US).
Communications routines return "True", but don't really work.	SimulateCommunications may be set to "True"
Unable to register PTALKDT.OCX .	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 we 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 and is included in the distribution.

```
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;
```


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 a collection of library functions 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 {command or variable}, 40
#include "filename", 40

Events

OnError, 54

Methods

DownloadFile, 2, 10, 38
DPRDouble, 40
DPRDWord, 41
DPRFixed, 41
DPRFloat, 43
DPRGetDWord, 42
DPRWord, 44
Flush, 2, 46
GetControlResponse, 2, 10, 46
GetLineAck, 2, 47
GetLineCR, 2, 47
GetResponse, 2, 48
IsLineWaiting, 48
LoadSettings, 2, 49
SaveSettings, 2, 51
SendChar, 2, 51
ShowPropertyPage, 2, 10, 53
UploadData, 2, 53

Properties

Enabled, 30
LastError, 30
LastErrorString, 31