PcommServer Library of PMAC Functions

License Statement and Limited Warranty

Delta Tau Data Systems Inc.

21314 Lassen St.

Chatsworth, CA 91311

(818) 998 2095

Fax: (818) 998 7807

Table of Contents

USER MANUAL

INTRODUCTION

The PMAC 32-bit communication driver provides all channel of communication between the host computer and PMAC controllers. All methods of communication to PMAC are included. All types of PMACs (Turbo and non-Turbo) use this driver for communication to the host computer.

About PcommServer Pro2 Library

The Delta Tau 32-bit communication driver PcommServer is a set of more than 400 functions written as a development tool for the creation of PMAC applications on Windows 2000 and XP. The routines are designed with robustness, speed and portability in mind. PcommServer may be used by all PMAC types.

Methods of communication include the bus (ISA and PCI), Dual Ported RAM, Serial, USB and Ethernet.

PcommServer is compatible with the 32-bit Borland and Microsoft development products, which include:

- Microsoft Visual Basic for .NET
- Microsoft C# for .NET
- Microsoft Visual C++ for .NET
- Microsoft Visual C++ 6.0
- Microsoft Visual Basic 6.0
- Borland C++ Builder 6.0
- DOS (basic communication) Examples and
- Linux driver files for ISA and PCI PMACs

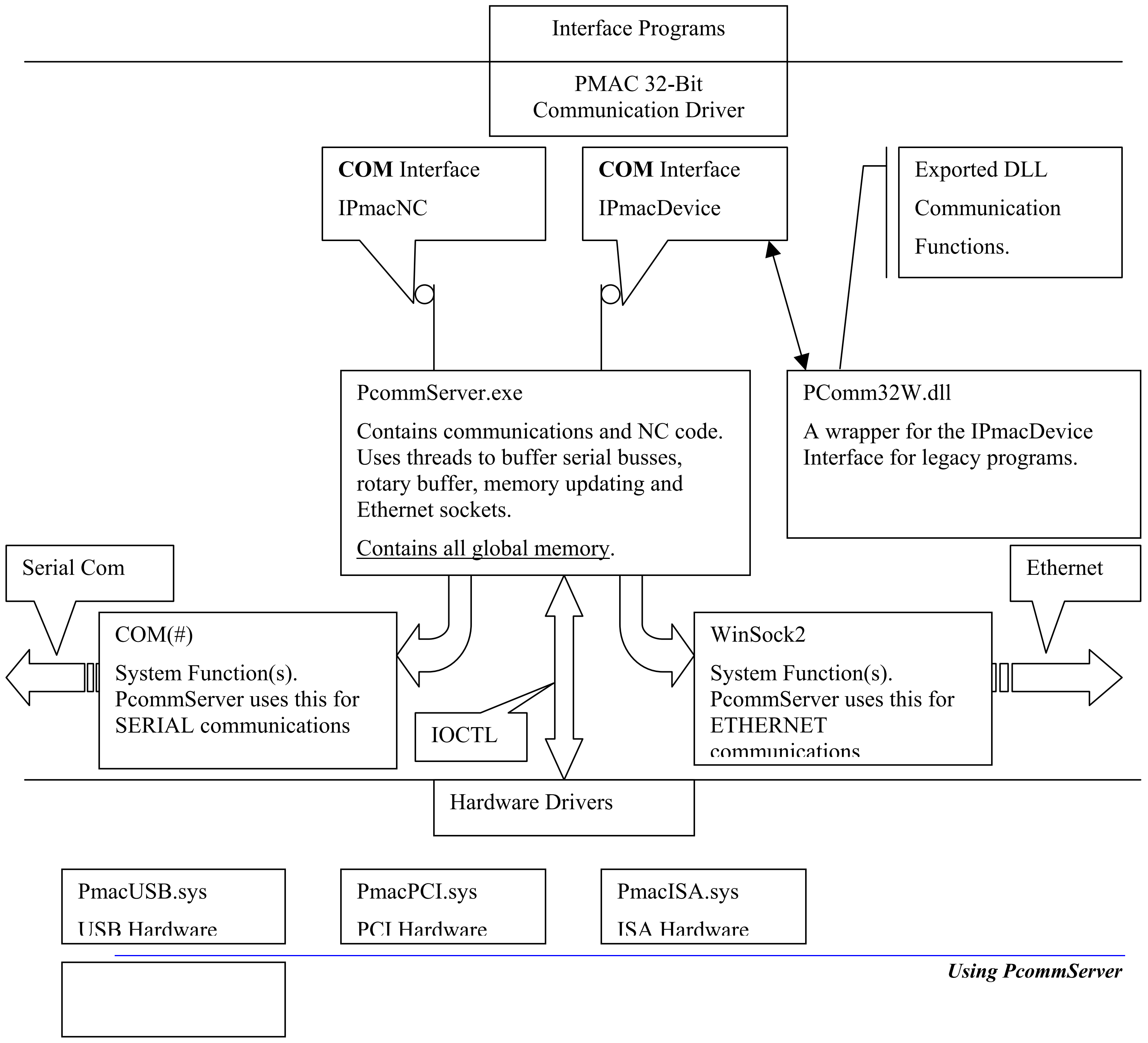
This manual assumes that you know Windows basics and general programming practices.

A Global View of the Driver

The driver can be used for Windows 2000/XP application development. The driver consists of following sets of files:

- PcommServer.exe – A ComInterface server application, responsible for core communication and transferring the data between the host computer and PMAC controllers.
- PmacISA.SYS, PmacPCI.SYS, PmacUSB.SYS – Windows 2000/XP kernel drivers.
- PmacISA.INF, PmacPCI.INF or PmacUSB.INF – Windows Setup Information files.
- ETHConfigure.EXE, USBConfigure.EXE and USBETHConfigure.EXE – Ethernet and USB configuration applications are responsible for boot firmware download and the IP configuration application is responsible for USB and Ethernet modes of communication. Furthermore, PmacETH.SYS loads the Ethernet mode at startup.
- A complete Source Code along with a simple User Interface for Linux operating system is packed in the file pmac.0.0.1.tar.gz.

The illustration below shows how these modules are related.


Interface Programs
PMAC 32-Bit Communication Driver
COM Interface IPmacNC
COM Interface IPmacDevice
Exported DLL Communication Functions.
PcommServer.exe
Contains communications and NC code. Uses threads to buffer serial busses, rotary buffer, memory updating and Ethernet sockets.
Contains all global memory.
PComm32W.dll
A wrapper for the IPmacDevice Interface for legacy programs.
Serial Com
Ethernet
COM(#)
System Function(s). PcommServer uses this for SERIAL communications
WinSock2
System Function(s). PcommServer uses this for ETHERNET communications
IOCTL
Hardware Drivers
PmacUSB.sys
USB Hardware
PmacPCI.sys
PCI Hardware
PmacISA.sys
ISA Hardware

Supported Operating Systems

The following operating systems are supported:

- Windows 2000
- Windows XP

Communication Modes

Plug & play ports

- PCI BUS PMAC
- USB Port PMAC

Non-plug & play ports

- ISA Bus PMAC
- Serial Port PMAC
- Ethernet port PMAC

Hardware Requirements

The PMAC 32-Bit Communication Driver for Windows requires a minimum specification of hardware for reliable operation and acceptable performance. These requirements include:

- 500 MHz Pentium III and above (of course, a faster computer will yield better throughput.)
- At least 35 MB of free disk space and 128 MB of RAM.
- A free serial communications port, USB port, Ethernet port, PCI BUS slot, or ISA BUS slot to talk to PMAC for on-line processing
- Any monitor with SVGA resolution (800x600 with at least 256 colors

GETTING STARTED

Setting up Communications with PMAC

No applications, including all Delta Tau software programs, will be used to add PMACs in your system. Rather, communication settings have been centralized in your operating system, making the set up of each PMAC much like that of other devices in your computer (i.e. printer, video card, sound card, etc.) All setup is done either automatically for Plug and Play device or through the Control Panel's **Add New Hardware** Wizard for non-Plug and Play devices. Detailed procedure on how to install and configure PMAC devices for all applications is explained in DT Driver_Install.PDF

Usage of PcommServer

In this section we discuss the usage of PcommServer in general, as well as specifically through several examples programs. In the following section, we explain the procedure on how to create new programs based on PcommServer in different development environments.

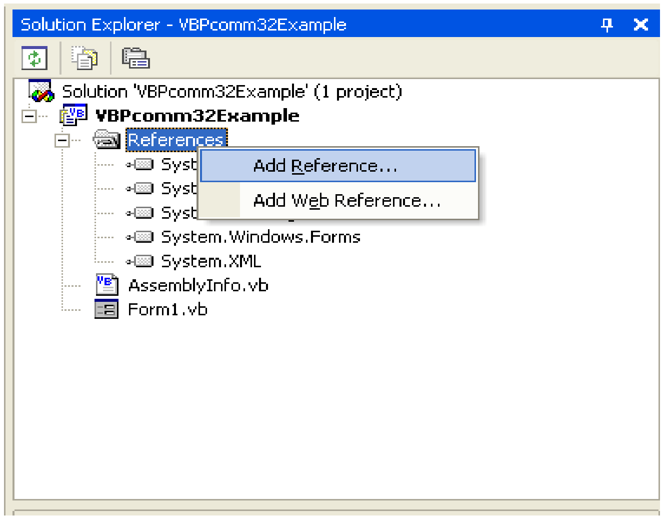
- Microsoft VB.NET sample projects
- Microsoft C#.NET sample projects
- Microsoft C++.NET (Managed C++ code) sample projects
- Microsoft C++ 6.0 sample projects
- Microsoft VB 6.0 sample projects
- Borland C++ Builder 6.0 sample projects

Microsoft VB.NET sample code

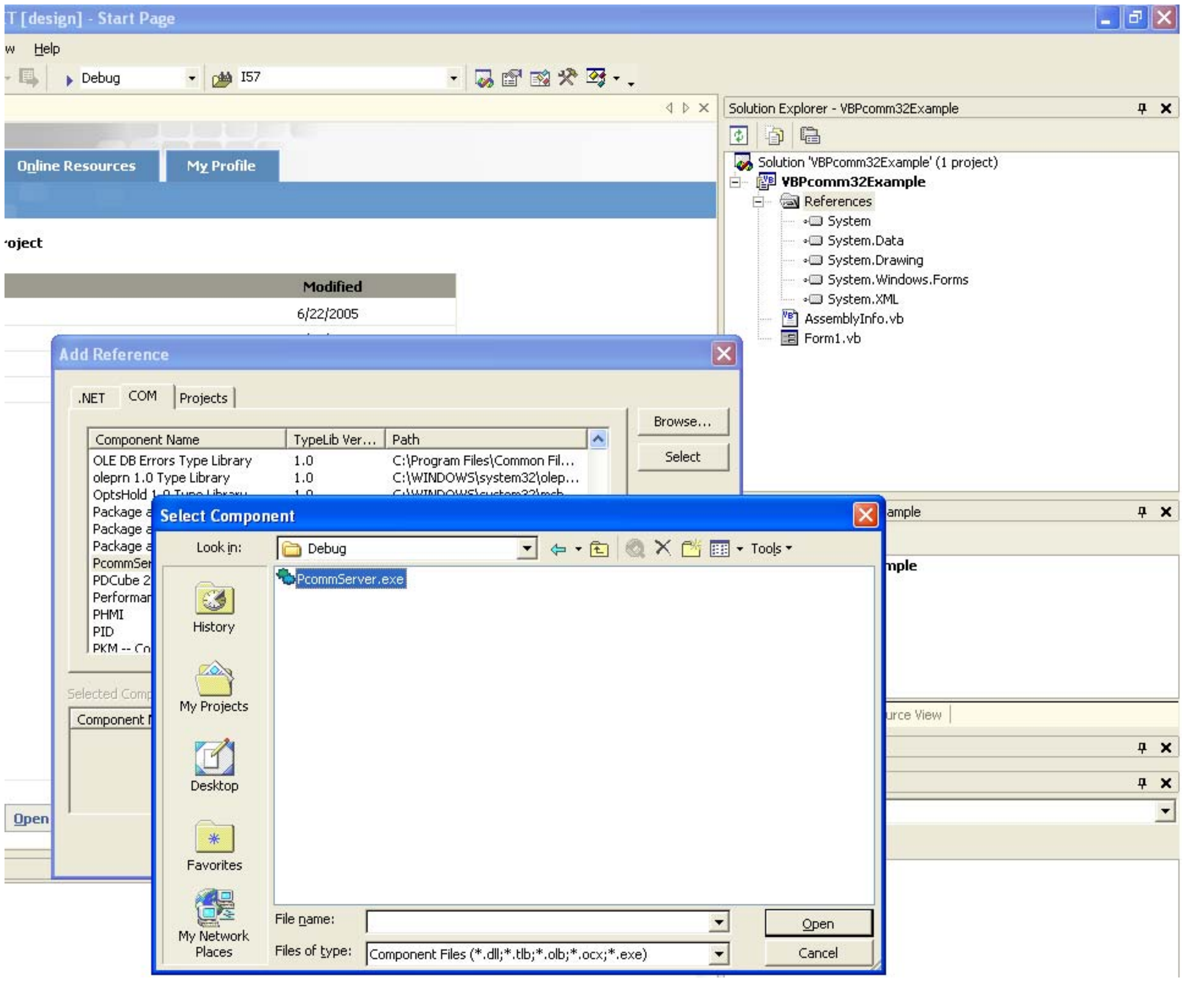
This example contains two steps. First we obtain a reference to PcommServer and then use it our code

<u>Obtaining a reference to PcommServer using VB.NET</u>

1. This document assumes that the user can create a visual basic .NET project.
2. After a project is created from the Solution Explorer of .NET right mouse click on the Reference Folder then Select **Add Reference…**



3. The **Add Reference** tabbed Dialog box will appear. Next select the **COM** tab and select the **Browse …** button and select PcommServer.exe.
4. Now that the PcommServer reference is available it may be used from VB.NET



Using the Reference to PcommServer in VB.NET

Using the reference just created from VB.NET is very easy. First the PmacDevice Object must be declared. This is done in the section **A**. The object pmac is declared as a PmacDevice from PcommServer. Next an instance of the pmac object must be created. This is done using the **New** statement see section **A**. Section **B** represents actual code that communicates to the PcommServer. In the following example, Open, Close, GetResponse, and SelectDevice parameters can be determined from the online help string that occurs when **pmac.** Is typed. In the following example three labels were created on the Visual Basic Form Label1, Label2 and Label3. These three labels display the PMAC version, date and type.

```
Public Class Form1

    Inherits System.Windows.Forms.Form

A   Public pmac As  New PCOMMSERVERLib.PmacDevice

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load


        Dim bSuccess As Boolean
        Dim status As Integer
        Dim Data As String
        Dim dwDevice As Integer


B       pmac.SelectDevice(0, dwDevice, bSuccess)

        pmac.Open(dwDevice, bSuccess)
        pmac.GetResponseEx(dwDevice, "VER", False, Data, status)
        Label1.Text = Data
        pmac.GetResponseEx(dwDevice, "DATE", False, Data, status)
        Label2.Text = Data
        pmac.GetResponseEx(dwDevice, "TYPE", False, Data, status)
        Label3.Text = Data

        pmac.Close(0)

    End Sub

End Class
```

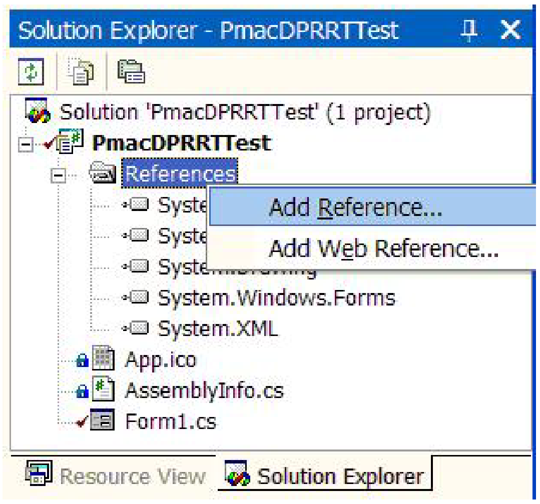
A simple VB.NET example is available for compilation in the folder <PROGRAM FILES>\DTDRIVER\DOTNETVB folder.

Microsoft C# .NET Example code

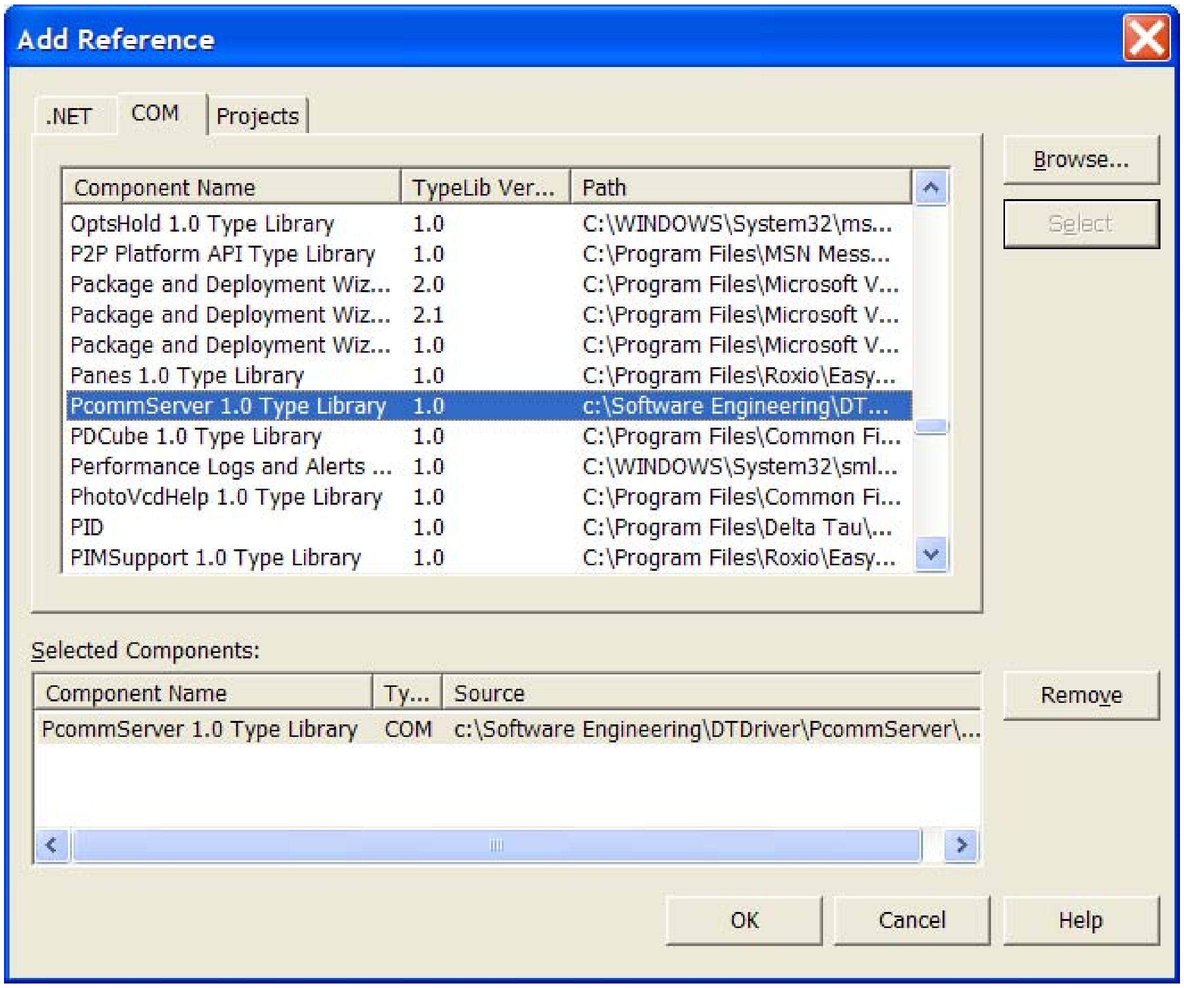
C# has become the environment of choice for many programmers for many applications. In this section we describe the procedure on how to obtain and use the reference to PcommServer for a C#.NET project.

Obtaining a reference to PcommServer using C#.NET

1. This document assumes that the user can create a C#.NET project.
2. After a project is created from the Solution Explorer of .NET right mouse click on the Reference Folder then Select **Add Reference…**



3. The **Add Reference** tabbed Dialog box will appear. Next select the **COM** tab and select the **Browse …** button and select PcommServer 1.0 Type Library.
4. Now that the PcommServer reference is available it may be used from C#.NET


Add Reference

.NET | COM | Projects

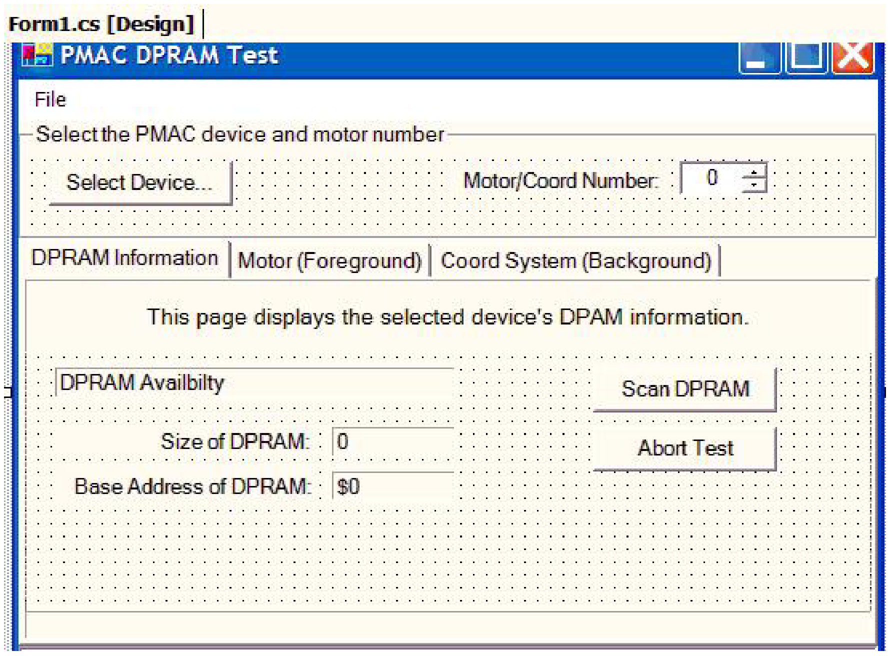
Component Name	TypeLib Ver...	Path
OptsHold 1.0 Type Library	1.0	C:\WINDOWS\System32\ms...
P2P Platform API Type Library	1.0	C:\Program Files\MSN Mess...
Package and Deployment Wiz...	2.0	C:\Program Files\Microsoft V...
Package and Deployment Wiz...	2.1	C:\Program Files\Microsoft V...
Package and Deployment Wiz...	1.0	C:\Program Files\Microsoft V...
Panes 1.0 Type Library	1.0	C:\Program Files\Roxio\Easy...
PcommServer 1.0 Type Library	1.0	c:\Software Engineering\DT...
PDCube 1.0 Type Library	1.0	C:\Program Files\Common Fi...
Performance Logs and Alerts ...	1.0	C:\WINDOWS\System32\sml...
PhotoVcdHelp 1.0 Type Library	1.0	C:\Program Files\Common Fi...
PID	1.0	C:\Program Files\Delta Tau\...
PIMSupport 1.0 Type Library	1.0	C:\Program Files\Roxio\Easy...

Selected Components:

Component Name	Ty...	Source
PcommServer 1.0 Type Library	COM	c:\Software Engineering\DTDriver\PcommServer\...

Using the Reference to PcommServer in C#.NET

Using the reference just created from C#.NET is very easy. First the PmacDevice Object must be declared.



This is done in the section **A**. The object pmac is declared as a PmacDevice from PcommServer. Next an instance of the pmac object must be created. This is done using the **New** statement see section **A**. Section **B** represents actual code that communicates to the PcommServer. In the following example a detailed DRPAM test procedure is provided. SelectDevice button (Button1) procedure is described as follows:

A

```
.
.
using PCOMMSERVERLib;
namespace PmacDPRRTTest
{
  public class Form1 : System.Windows.Forms.Form
  {
        .
        .
    private System.ComponentModel.Container components = null;
    public Form1()
    {
      InitializeComponent();

      Pmac = new PmacDeviceClass();
      m_nDevice = 0;
        .
        .
      SetAllEvents();
      UpdateStatusDisplay();
    }

    public static PCOMMSERVERLib.PmacDeviceClass Pmac;
    public int m_nDevice, m_nMotor, m_nServo, m_nDPRBase, m_nDPRSize;
    public bool m_bDeviceOpen, m_bDPRAvailable, m_bTurbo;
```

B

```
private void button1_Click(object sender, System.EventArgs e)
{
  int dev = m_nDevice;
  bool bSuccess = false;
  string str;
  Pmac.SelectDevice(0,out dev,out bSuccess);
  if(bSuccess)
  {
    OutputListBox.Items.Clear();
    if(m_bDeviceOpen)
      Pmac.Close(m_nDevice);
    m_nDevice = dev;
    m_bDPRAvailable = false;
    m_nDPRBase = 0;
    m_nDPRSize = 0;
    Pmac.Open(m_nDevice,out m_bDeviceOpen);
    if(m_bDeviceOpen)
    {
      m_bDPRAvailable = Pmac.get_DPRAvailable(m_nDevice);
      m_nDPRSize = Pmac.get_DPRSize(m_nDevice);
      Pmac.DPRAddressRange(m_nDevice,false,false,out m_nDPRBase);
      m_bTurbo = Pmac.get_IsTurbo(m_nDevice);
      m_bFGEnabled = Pmac.get_DPRMotorRptEnabled(m_nDevice);
      // Setup for this device
      if(m_bTurbo)
      {
        MotorNumberUD.Maximum = 32;
        MotorMaskLabel.Text = "Motor Mask:";
        FGMoveTimeLBL.Visible = false;
        label22.Visible = false;
        label16.Text = "PSET Bias Position:";
        label19.Text = "Feedrate:";
        label28.Text = "Feed Pot:";
      }
      else
      {
        MotorNumberUD.Maximum = 8;
        MotorMaskLabel.Text = "  Motor(s):";
        FGMoveTimeLBL.Visible = true;
        label22.Visible = true;
        label16.Text = "Compensation Position:";
        label19.Text = "Target Position:";
        label28.Text = "Bias Position:";
      }
      str = string.Format("Device number {0:D} opened
    successfully.",m_nDevice);
    }
    else
    {
      // Disable all required
      str = string.Format("Device number {0:D} failed to open.",m_nDevice);
    }
    OutputListBox.Items.Add(str);
  }
  UpdateStatusDisplay();
}
```

The next step in this example shows how to setup and display events. Following events are generated by PcommServer and must be setup correctly in order to be displayed.

Following events are available in the PmacDevice interface of PcommServer.

C

```
private void SetAllEvents()
{
  if(Pmac != null)
  {
    _IPmacDeviceEvents_UnsolicitedEventHandler DUnsolicitedEventE
      = new _IPmacDeviceEvents_UnsolicitedEventHandler(OnUnsolicited);
    Pmac.Unsolicited += DUnsolicitedEventE;

    _IPmacDeviceEvents_ProgressEventHandler DProgressEventE
        = new _IPmacDeviceEvents_ProgressEventHandler(OnProgress);
    Pmac.Progress += DProgressEventE;

    _IPmacDeviceEvents_MessageEventHandler DMessageEventE
        = new _IPmacDeviceEvents_MessageEventHandler(OnMessage);
    Pmac.Message += DMessageEventE;

    _IPmacDeviceEvents_ErrorEventHandler DErrorEventE
        = new _IPmacDeviceEvents_ErrorEventHandler(OnError);
    Pmac.Error += DErrorEventE;

    _IPmacDeviceEvents_InterruptEventHandler DInterruptEventE
        = new _IPmacDeviceEvents_InterruptEventHandler(OnInterrupt);
    Pmac.Interrupt += DInterruptEventE;
  }
}
```

Finally, the events are displayed by the following code.

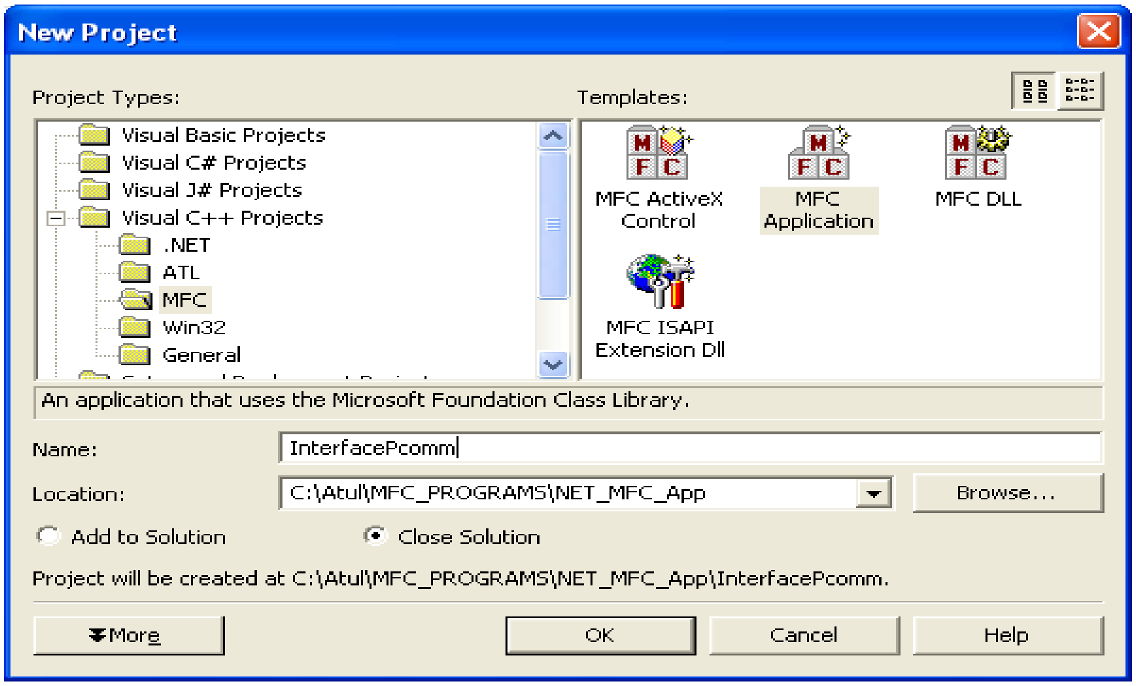
D

```
// An unsolicited event from the server
void OnUnsolicited(int device,String Text)
{
  OutputListBox.Items.Add(Text);
  OutputListBox.SetSelected(OutputListBox.Items.Count-1,true);
  OutputListBox.Refresh();
}
// A progress update from the server
void OnProgress(int device,int percent)
{
  progressBar1.Value = percent;
  if(percent == 100 || percent == 0)
    AbortTestButton.Enabled = false;
  else
    AbortTestButton.Enabled = true;
}
// A message from the server
void OnMessage(int device,String Text,bool linefeed)
{
  OutputListBox.Items.Add(Text);
  OutputListBox.SetSelected(OutputListBox.Items.Count-1,true);
  OutputListBox.Refresh();
}
// A error message from the server
void OnError(int device,String filename,int errorNumber,int
    lineNumber,String msg)
{
  string str;
  str = string.Format("{0} {1:D},
    Line:{2:D},{4}{3}{4}",filename,errorNumber,lineNumber,msg,'"');
  OutputListBox.Items.Add(str);
  OutputListBox.SetSelected(OutputListBox.Items.Count-1,true);
  OutputListBox.Refresh();
}
// A interrupt message from the server
void OnInterrupt(int device,int interruptLevel)
{
  String msg = "";
  switch(interruptLevel)
  {
    case 1:
      msg = "** Interrupt-> In Position **"; break;
    case 2:
      msg = "** Interrupt-> Buffer Request **"; break;
    case 4:
      msg = "** Interrupt-> Warning Following Error **"; break;
    case 8:
      msg = "** Interrupt-> Fatal Following Error **"; break;
    case 16:
      msg = "** Interrupt-> Host Request **"; break;
    case 32:
      msg = "** Interrupt-> IR5 **"; break;
    case 64:
      msg = "** Interrupt-> IR6 **"; break;
    case 128:
      msg = "** Interrupt-> IR7 **"; break;
    default: msg = "** Interrupt-> Spurious **"; break;
  }
  OutputListBox.Items.Add(msg);
  OutputListBox.SetSelected(OutputListBox.Items.Count-1,true);
  OutputListBox.Refresh();
}
```

A detailed C#.NET example is available for compilation in the folder <PROGRAM FILES>\DTDRIVER\DOTNETC# folder.

Using PcommServer in MFC using .NET

1. Create New MFC project in Microsoft Visual Studio .NET 2003. Name the project as InterfacePcomm.



An application that uses the Microsoft Foundation Class Library.

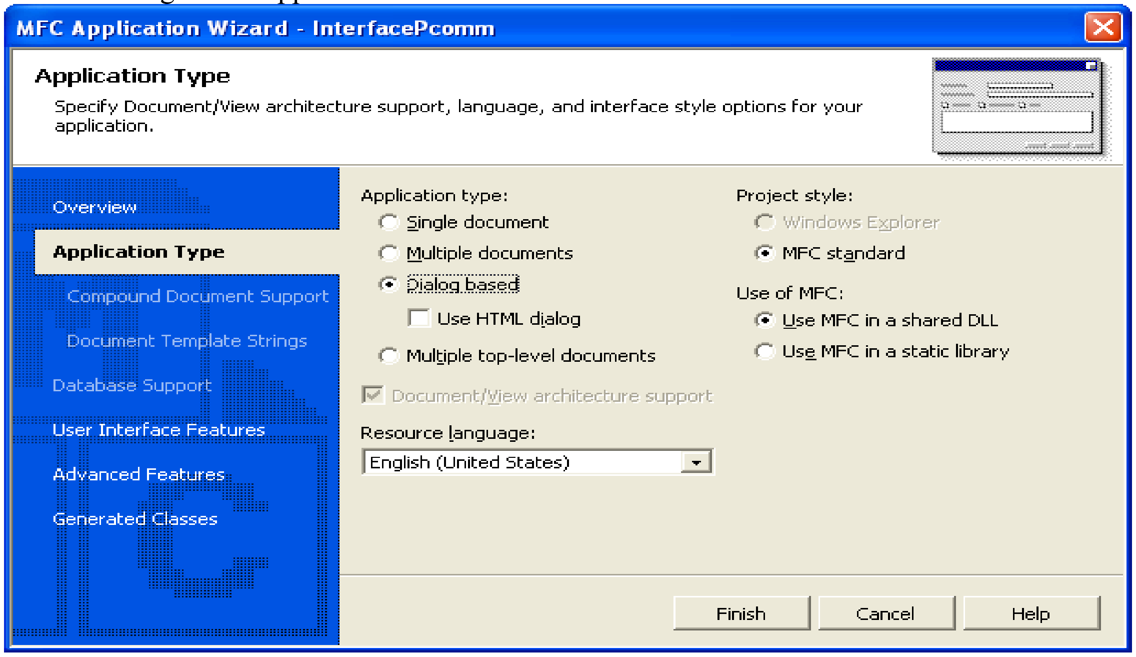
Name: InterfacePcomm

Location: C:\Atul\MFC_PROGRAMS\NET_MFC_App

[Add to Solution] [✓ Close Solution]

Project will be created at C:\Atul\MFC_PROGRAMS\NET_MFC_App\InterfacePcomm.

2. Select Dialog based Application and click Finish.



Application Type

Specify Document/View architecture support, language, and interface style options for your application.

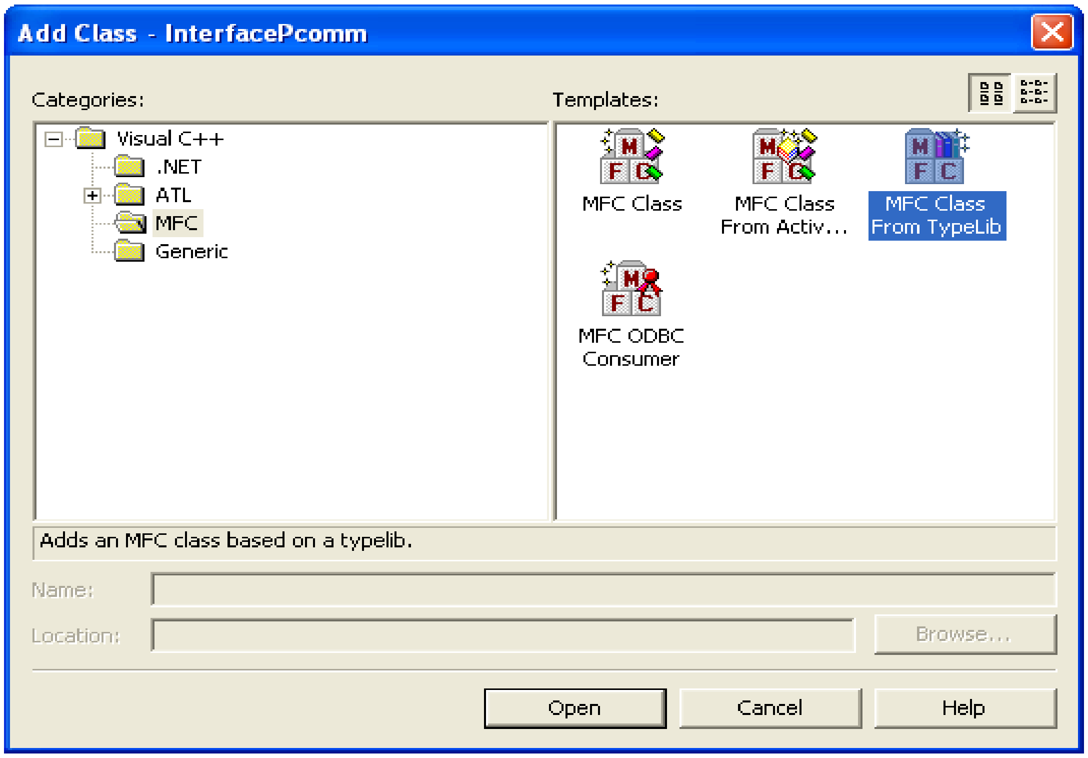
Application type:
Single document
Multiple documents
[✓ Dialog based]
Use HTML dialog
Multiple top-level documents
[✓ Document/View architecture support]

Resource language: English (United States)

Project style:
Windows Explorer
[✓ MFC standard]

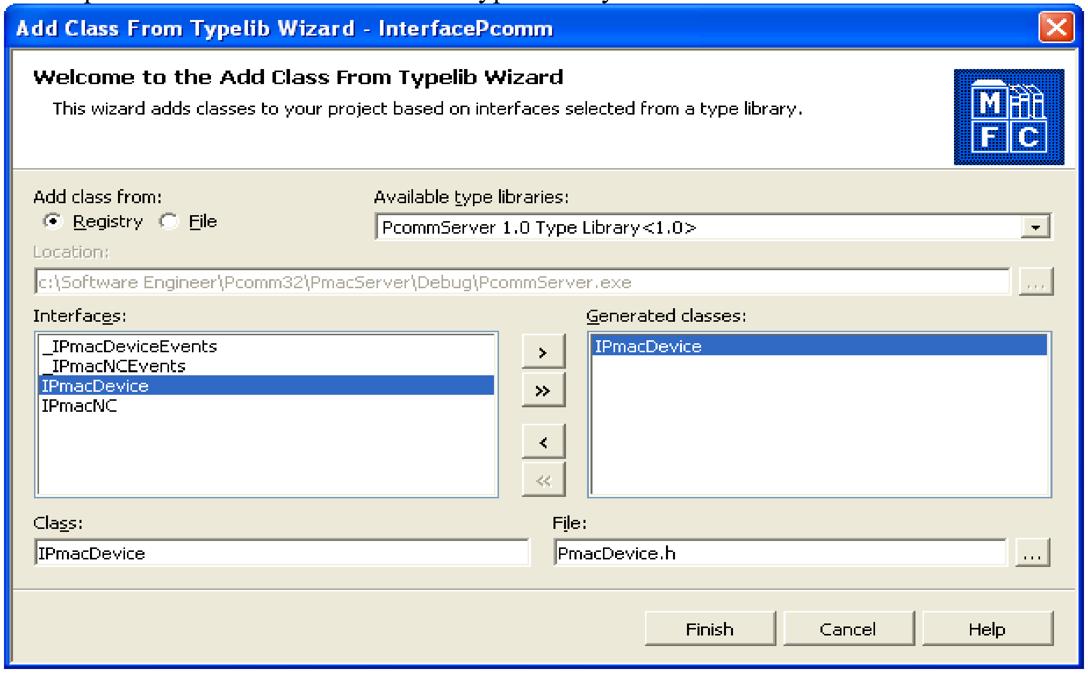
Use of MFC:
[✓ Use MFC in a shared DLL]
Use MFC in a static library

3. Add New Class from type Library.



Adds an MFC class based on a typelib.

4. Click Open and select Pcommserver 1.0 type Library<1.0>



Welcome to the Add Class From Typelib Wizard

This wizard adds classes to your project based on interfaces selected from a type library.

Add class from: [✓ Registry] File

Available type libraries: PcommServer 1.0 Type Library<1.0>

Location: c:\Software Engineer\Pcomm32\PmacServer\Debug\PcommServer.exe

Interfaces:
_IPmacDeviceEvents
_IPmacNCEvents
IPmacDevice
IPmacNC

Generated classes:
IPmacDevice

Class: IPmacDevice

File: PmacDevice.h

Select IPmacDevice under Interface and rename Class as IPmacDevice and file as PmacDevice.h. Click finish.

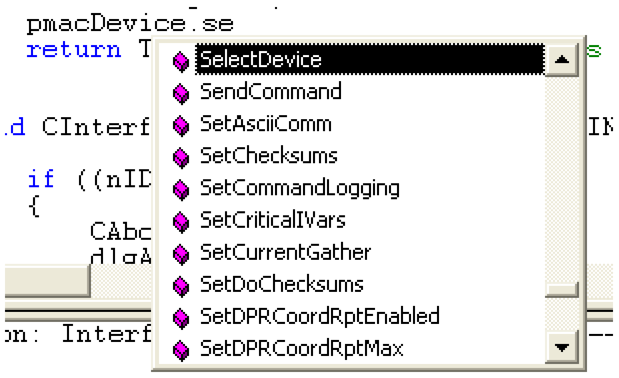
5. Add this line in InterfacePcommDlg.Cpp file

```
    #include "PmacDevice.h"
```

6. Search for function BOOL CInterfacePcommDlg::OnInitDialog() and add following code.

```
// TODO: Add extra initialization here
            long testInterface;
            IPmacDevice pmacDevice;
           long dwDevice;
           BOOL pbSuccess;
   CoInitialize(NULL);
            testInterface = pmacDevice.CreateDispatch(_T("PcommServer.PmacDevice.1"));
   if (!testInterface)
     AfxMessageBox("Can Not Connect PcommServer Interface ");
```

That completes the interface! Compile the code. Now use pmacDevice to access all the PmacDevice function. Type pmacDevice. Will give you list of function available.



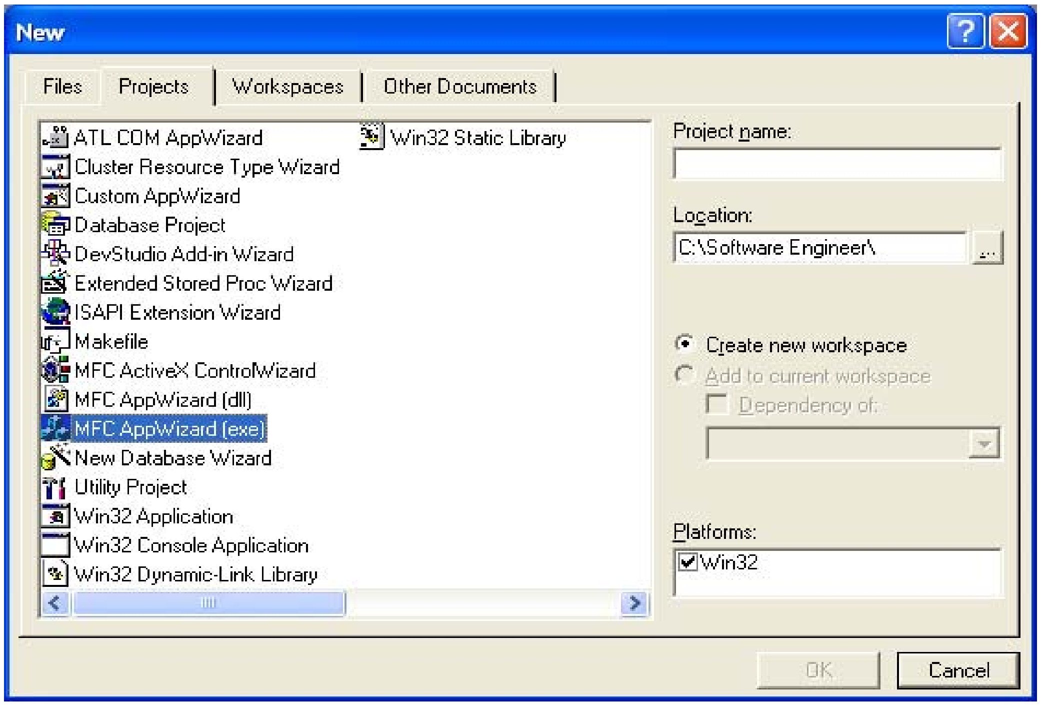
For Example : Add this code to open SelectDevice Dialog box.

```
pmacDevice.SelectDevice(NULL,&dwDevice,&pbSuccess);
```

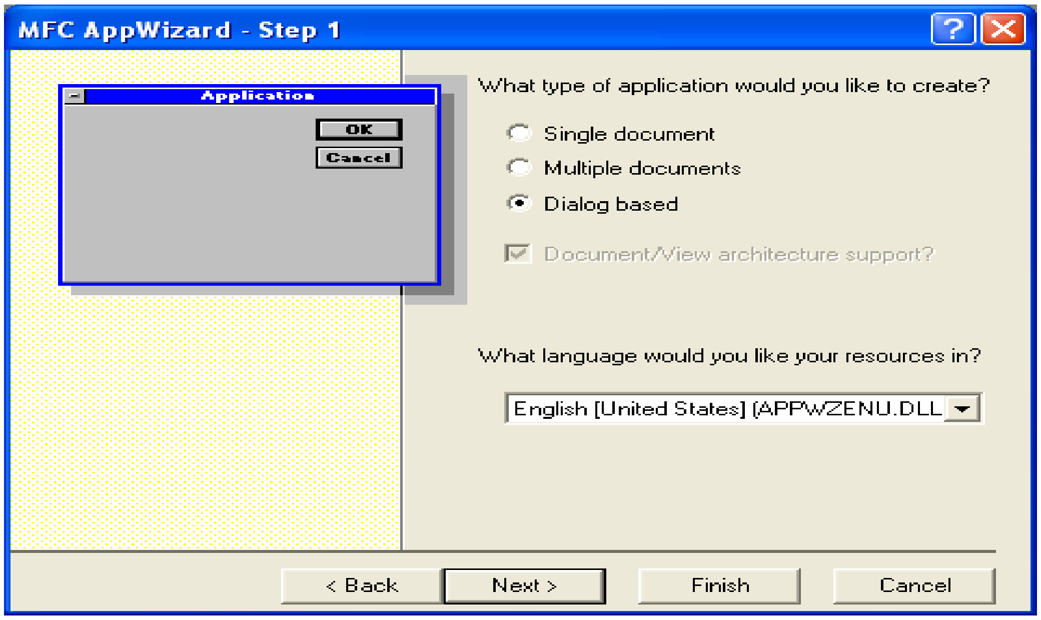
Compile the code and Run! The PmacSelect() dialogbox will appear. Test the PMAC and this completes the PcommServer Interface.

Interfacing PCOMMSERVER in Visual Studio 6 (C++) Applications

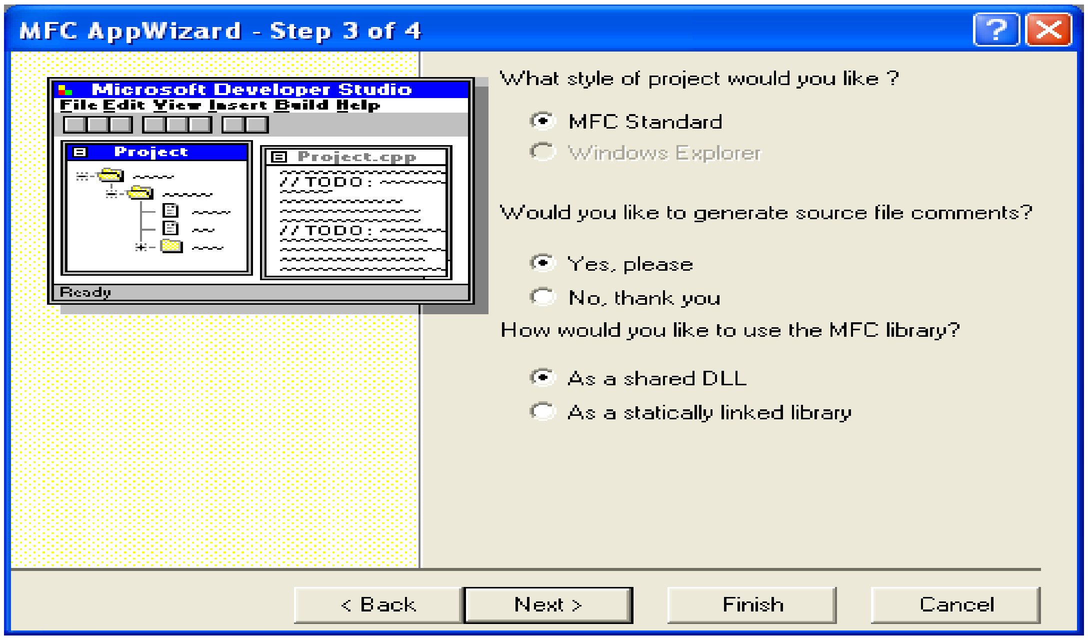
1. Create MFC project using AppWizard. Enter project name and click OK. For our example type InterfacePcomm as project Name.



2. Select Dialog based Application and click Next.


MFC AppWizard - Step 1
Application
OK
Cancel
What type of application would you like to create?
Single document
Multiple documents
Dialog based
Document/View architecture support?
What language would you like your resources in?
English [United States] (APPWZENU.DLL
< Back
Next >
Finish
Cancel

3. Create the project and make sure for MFC support.


MFC AppWizard - Step 3 of 4

What style of project would you like ?

[✓ MFC Standard]
Windows Explorer

Would you like to generate source file comments?

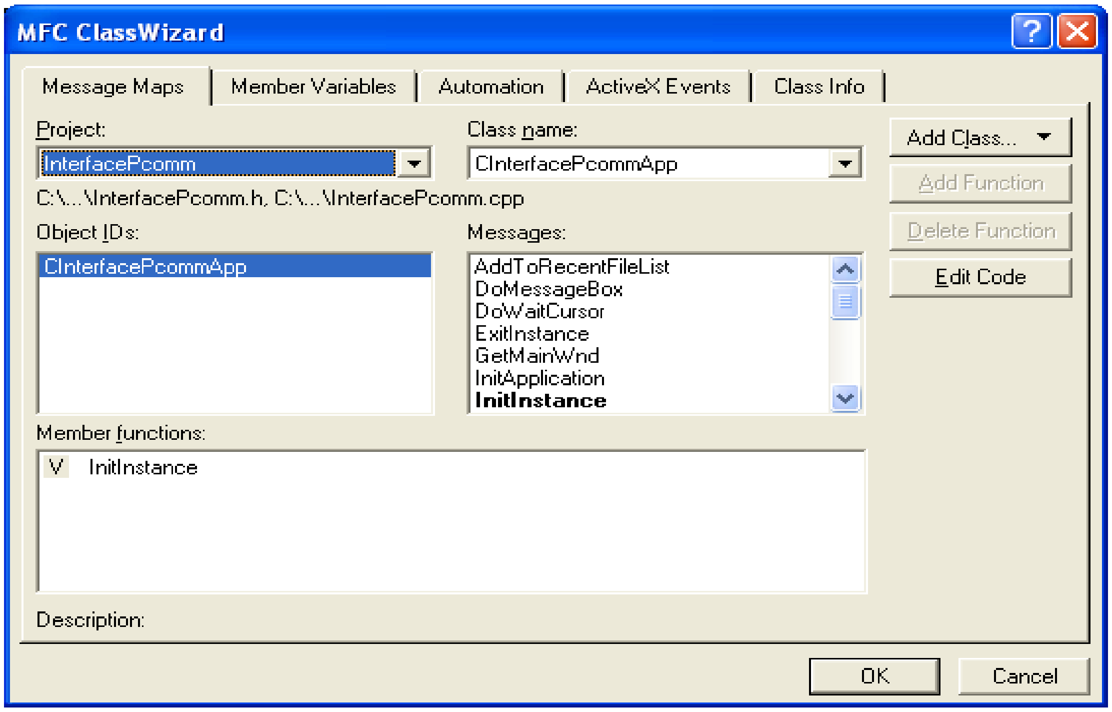
[✓ Yes, please]
No, thank you

How would you like to use the MFC library?

[✓ As a shared DLL]
As a statically linked library

4. Click Finish to generate project template.

5. Add new class from type library.


MFC ClassWizard

Project: InterfacePcomm

Class name: CInterfacePcommApp

C:\...\InterfacePcomm.h, C:\...\InterfacePcomm.cpp

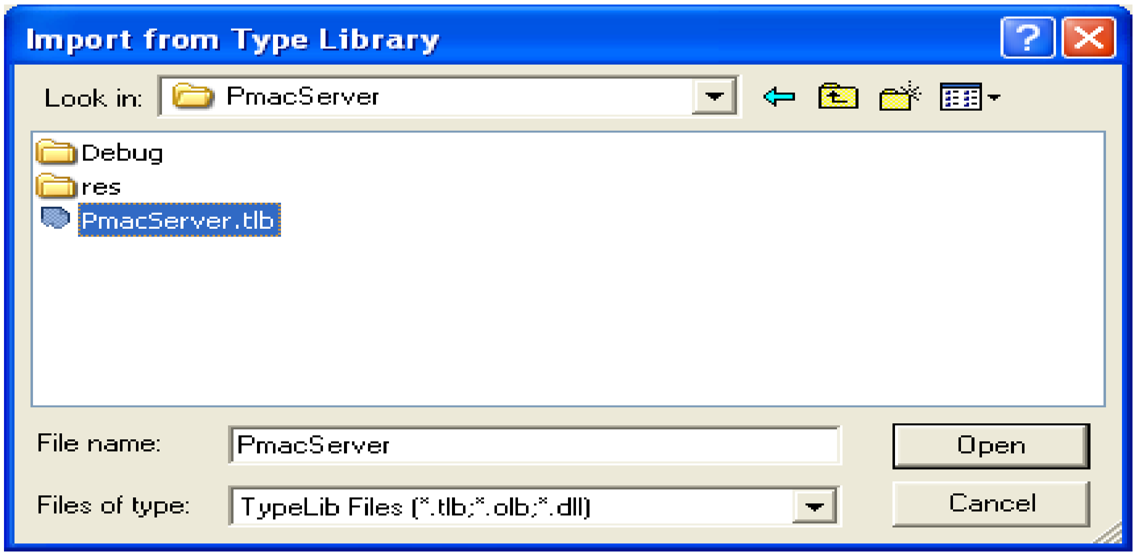
Object IDs: CInterfacePcommApp

Messages:
AddToRecentFileList
DoMessageBox
DoWaitCursor
ExitInstance
GetMainWnd
InitApplication
InitInstance

Member functions:
V InitInstance

Description:

6. Locate, select and open Pmacserver.tlb or Pcommserver.Exe file.



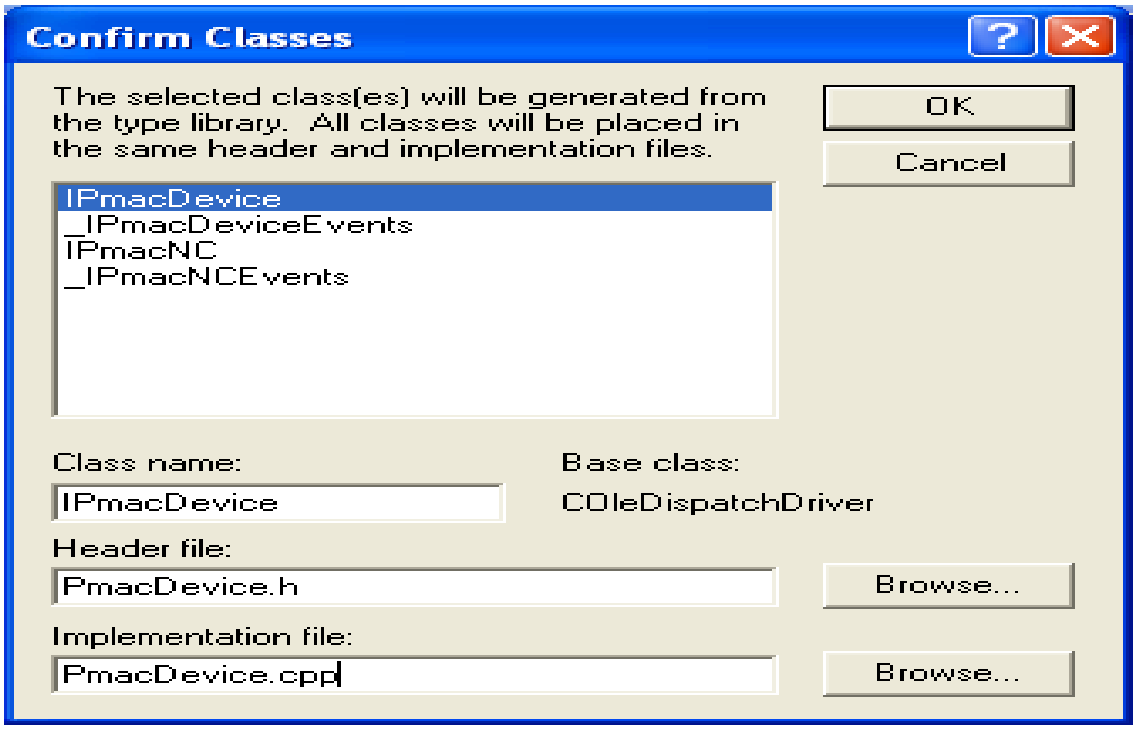
Look in: PmacServer

Debug
res
PmacServer.tlb

File name: PmacServer

Files of type: TypeLib Files (*.tlb;*.olb;*.dll)

7. On opening of the file it will show the available Classes.



The selected class(es) will be generated from the type library. All classes will be placed in the same header and implementation files.

IPmacDevice
_IPmacDeviceEvents
IPmacNC
_IPmacNCEvents

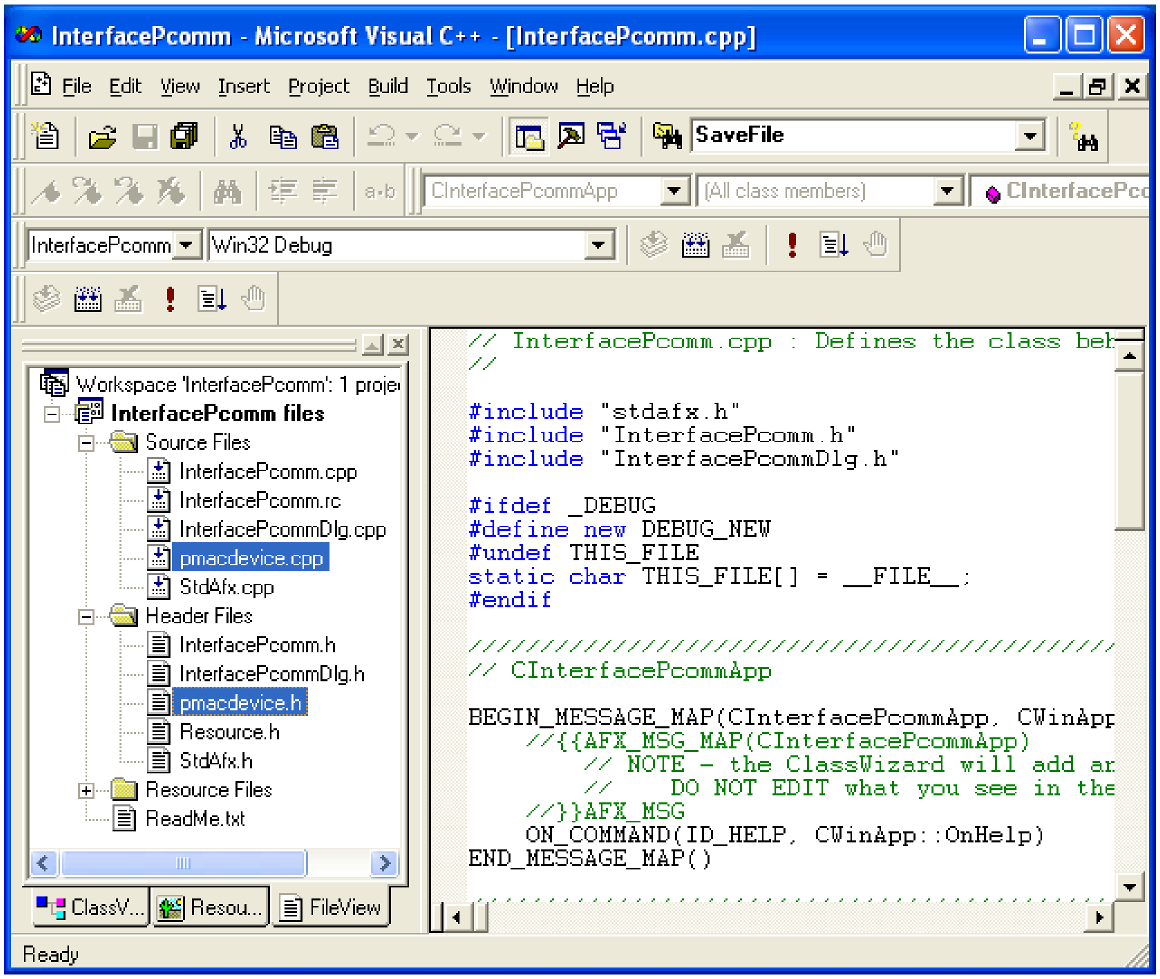
Class name: IPmacDevice

Base class: COleDispatchDriver

Header file: PmacDevice.h

Implementation file: PmacDevice.cpp

8. To interface PMACDEVICE select IpmacDevice and change the Header file name to PmacDevice.h and implementation file name to PmacDevice.cpp. Click Ok. This will add the class in to the created project.



```
// InterfacePcomm.cpp : Defines the class beh
//

#include "stdafx.h"
#include "InterfacePcomm.h"
#include "InterfacePcommDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////
// CInterfacePcommApp

BEGIN_MESSAGE_MAP(CInterfacePcommApp, CWinApp
    //{{AFX_MSG_MAP(CInterfacePcommApp)
        // NOTE - the ClassWizard will add an
        //    DO NOT EDIT what you see in the
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()
```

9. Open PmacDevice.H and add #include "PmacServer_i.c". Make sure to locate this file and set appropriate project directory or copy file in the same folder of the InterfacePcomm project.

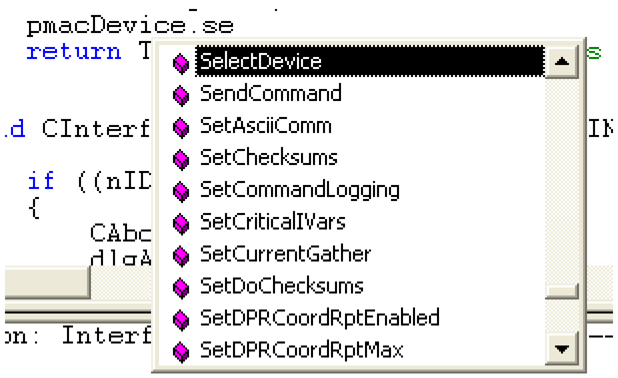
10. Add this line in InterfacePcommDlg.Cpp file

```
#include "PmacDevice.h"
```

11. Search for function BOOL CInterfacePcommDlg::OnInitDialog() and add following code.

```
// TODO: Add extra initialization here
        long testInterface;
        IPmacDevice pmacDevice;
       long dwDevice;
       BOOL pbSuccess;
   CoInitialize(NULL);
        testInterface = pmacDevice.CreateDispatch(_T("PcommServer.PmacDevice.1"));
   if (!testInterface)
     AfxMessageBox("Can Not Connect PcommServer Interface ");
```

That completes the interface! Compile the code. Now use pmacDevice to access all the PmacDevice function. Type pmacDevice. Will give you list of function available.


```
pmacDevice.se
return I
.d CInterf
if ((nID
{
   CAbc
   dlgA
on: Interf
```

For Example : Add this code to open SelectDevice Dialog box.

pmacDevice.SelectDevice(NULL,&dwDevice,&pbSuccess);

Compile the code and Run! The PmacSelect() dialog box will be displayed. Test the PMAC and this completes the PcommServer Interface.

Visual Basic 6.0 Example

Borland C++

C++ is yet another very powerful platform for C++ development and has been used for years by Delta Tau team for developing Pewin32Pro2 Suite components. In this section we describe the procedure on how to obtain and use the reference to PcommServer for a C++ in Builder environment.

<u>Connecting to PcommServer interface using Borland C++</u>

Each ConncetInterface() at the start of application must be is aasociated with a corresponding DisconnectInterface() at the close of application.

```
IPmacDevice *m_iDevice;    // pointer to PMAC interface

bool        ConnectInterface();
void        DisconnectInterface();

//-----------------------------------------------------------------------------
bool TPmac::ConnectInterface()

{
  m_hResult = S_FALSE;
  if(m_iDevice == NULL)
  {
    // Get a handle to the NC service of PcommServer
    m_hResult = CoCreateInstance(CLSID_PmacDevice, NULL, CLSCTX_LOCAL_SERVER,
      IID_IPmacDevice, (void **)&m_iDevice);
    if(SUCCEEDED(m_hResult))
    {
      m_iDevice->AddRef();
      m_bAdviseConnected = (ConnectEvents(m_iDevice) == S_OK);
    }
    else
    {
      m_iDevice = NULL;
      Application->MessageBox ("Unable to launch PcommServer. Check file location and version and then
        restart your application again.","PcommServer Error!!!",MB_ICONSTOP|MB_TOPMOST);
    }
  }
  return SUCCEEDED(m_hResult);
}
//-----------------------------------------------------------------------------
void TPmac::DisconnectInterface()
{
        if(m_iDevice != NULL)
  {
    if  (m_bAdviseConnected)
      DisconnectEvents(m_iDevice);
    m_bAdviseConnected = false;
    m_iDevice->Release();
    m_iDevice = NULL;
  }
}
```

Calling the communication functions is very easy once the Interface to PcommServer is successfully established. Following example describes that procedure on how to select a specific device number and consequently communicate with it under Borland C++ environment.

```
TCHAR       m_szPmacType[MAXDEVICES][300];
DWORD       SelectDevice(HWND hWnd = NULL);
bool       Open(DWORD dwDevice);
bool       Close(DWORD dwDevice);
//---------------------------------------------------------------------------
DWORD TPmac::SelectDevice(HWND hWnd)
{
  DWORD dwDevice = NO_PMAC_DEVICE;
  VARIANT_BOOL bSuccess;
  if(m_iDevice)
    m_iDevice->SelectDevice((long)hWnd,(long *)&dwDevice,&bSuccess);
  return dwDevice;
}
//---------------------------------------------------------------------------
bool TPmac::Open(DWORD dwDevice)
{
  TCHAR str[400],szPmacType[30],szPmacLocation[30];
  WideString wvs, wds;
  AnsiString vs, ds;
  HCURSOR hcurSave;

  if(m_iDevice == NULL)
    return FALSE;
  if(m_bDriverOpen[dwDevice])
    return TRUE; // if already open at document level
  m_bInterrupt[dwDevice] = FALSE;
  hcurSave = ::SetCursor(::LoadCursor(NULL,IDC_WAIT));
  m_iDevice->Open(dwDevice,&m_bDriverOpen[dwDevice]);
  ::SetCursor(hcurSave);
  if(m_bDriverOpen[dwDevice])
  {
      m_iDevice->GetRomVersion(dwDevice,&wvs);
      vs = wvs;
      m_iDevice->GetRomDate(dwDevice,&wds);
      ds = wds;
      m_iDevice->GetFirmwareType(dwDevice,&m_FirmwareType[dwDevice]);
      m_iDevice->GetPmacType(dwDevice,&m_pmactype[dwDevice]);
      m_iDevice->GetPmacLocation(dwDevice,&m_pmacLocation[dwDevice]);
      sprintf(m_szPmacType[dwDevice],"PMAC:%d V%s %s   %s: %s",dwDevice,vs.c_str(),
        ds.c_str(),szPmacType,szPmacLocation);
    }
  }
  else
  {
    sprintf(m_szPmacType[dwDevice],TEXT("Unable To Communicate to PMAC Device %d. Please make sure that
        your PMAC is properly \nconfigured and in case of Serial/USB/Ethernet mode it is powered up and
        cable connected.\n You need to go to general setup & Options menu to select a different PMAC from
        the \nDevice Selection menu!!!"),dwDevice);
    Application->MessageBox(m_szPmacType[dwDevice],"ATTENTION!",MB_OK | MB_TOPMOST);
    return false;
  }
  return (m_bDriverOpen[dwDevice]);
}
//---------------------------------------------------------------------------
```

Finally, the events are captured and displayed in any application using the following procedure.

```
// TEventDispatcher - Protected
HRESULT InvokeEvent(DISPID id, TVariant* params = 0, VARIANT* pVarResult = 0);
typedef void (FAR WINAPI *MESSAGEPROC) ( TForm *sender, AnsiString msg);
typedef void (FAR WINAPI *PROGRESSPROC) ( TForm *sender, ULONG nPercent );
typedef void (FAR WINAPI *INTERRUPTPROC) ( TForm *sender, long lLevel, AnsiString &msg);
typedef void (FAR WINAPI *UNSOLICITEDPROC) ( TForm *sender, AnsiString msg);

HRESULT TPmac::InvokeEvent(DISPID id, TVariant *params, VARIANT *pVarResult)
{
  HRESULT hRet = S_OK;
  long ErrId, ErrLine, IntLevel;
  BOOL msgNewLine;
  DWORD dwDevice;
  AnsiString Gmsg;
  AnsiString msg, ErrMsg, ErrFileName;
  char cTemp[256];
  PROGRESSPROC  m_ProgressProc;
  MESSAGEPROC   m_MessageProc;
  INTERRUPTPROC m_InterruptProc;
  UNSOLICITEDPROC   m_UnSolicitedProc;
  if(params == NULL)    return E_POINTER;
  dwDevice = params[0];
  switch(id)
  {
    case 1:
      if(m_ProgressProc)
      {
        msg = params[1];
        Gmsg.sprintf("Device %d-> %s",dwDevice,msg.Trim().c_str());
      }
      break;
    case 2: // Misc progress function
      if(m_ProgressProc)
        m_ProgressProc(m_parent,params[1]);
      break;
    case 3: // Message without a return required
      ErrFileName = params[1];
      ErrId    = params[2];
      ErrLine  = params[3];
      ErrMsg   = params[4];
      switch(ErrId)
      {
        case MSG_ERR_USB_UNPLUGGED: // Somebody unplugged the USB cable to Device
          Gmsg.sprintf("Device %d-> %s",dwDevice,ErrMsg.c_str());
          if(m_bDriverOpen[dwDevice])
          {
            Close(dwDevice);
            sprintf(m_szPmacType[dwDevice],TEXT("Unable To Communicate to PMAC Device %d"),dwDevice);
          }
          break;
        case MSG_ERR_USB_PLUGGEDIN: // And now they plugged it back in
          Gmsg.sprintf("Device %d-> %s",dwDevice,ErrMsg.c_str());
          break;
        default:
          Gmsg.sprintf("Device %d-> Error 0x%X, %s on line %d",dwDevice,ErrId,ErrMsg.c_str(),ErrLine);
          break;
      }
      break;
    case 4:
      IntLevel = params[1];
      switch(IntLevel)
      {
          case 0: // ISR_IPOS:
            sprintf(cTemp,"** Interrupt-> In Position **");
            break;
          case 1: // ISR_BREQ:
            sprintf(cTemp,"** Interrupt-> Buffer Request **");
            break;
          case 2: // ISR_FFERROR:
            sprintf(cTemp,"** Interrupt-> Fatal Follow Error **");
            break;
          case 3: // ISR_WFERROR:
            sprintf(cTemp,"** Interrupt-> Warning Following Error **");
            break;
          case 4: // ISR_HREQ:
            sprintf(cTemp,"** Interrupt-> Host Request **");
            break;
          case 5: // ISR_IR5:
            sprintf(cTemp,"** Interrupt-> IR5 **");
            break;
          case 6: // ISR_IR6:
            sprintf(cTemp,"** Interrupt-> IR6 **");
            break;
          case 7: // ISR_IR7:
            sprintf(cTemp,"** Interrupt-> IR7 **");
            break;
          default:
            sprintf(cTemp,"** Interrupt-> Spurious **");
          break;
      }
      Gmsg.sprintf("Device %d-> %s",dwDevice,cTemp);
      break;
    case 5:
        // Display Unsolicited Response
      if(m_UnSolicitedProc)
      {
        sprintf(cTemp,AnsiString(params[1]).c_str());
        IdentifyControlChars(cTemp,msg);
        Gmsg.sprintf("Device %d-> %s",dwDevice,msg.c_str());
        m_UnSolicitedProc(m_parent,Gmsg.c_str());
      }
      break;
    case 6: // Data Ready event for HMI CRAP just ignore it
      Gmsg.sprintf("InvokeEvent(): Data ready event. Just ignore it");
      break;
    default:
      hRet = E_INVALIDARG;
      break;
  }
  return hRet;
}
```

Shutting Down Communication

Before closing any application it is important to close handle to the device. So always issue the CloseDevice to shut down any communication links that have been opened. In all of the example programs, this is done as shown below:

```
void CPmacTestDoc::CloseDocument()
{
  if(m_bDriverOpen) {
     // Call ClosePmacDevice()
      m_bDriverOpen = !DeviceClose(m_dwDevice);
  }
}
```

PCOMMSERVER FEATURES

A Guide to Using ASCII Communication Functions

Most if not all of your communication with the PMAC can be handled with the GetResponseEx() function. This function will send a command string (i.e. "#1j+, "?", "Open Prog1", etc) to the PMAC and retrieve and place any pending responses within a response buffer for your use. This is an efficient function to use. GetResponseEx() always matches the command string with the response string or else it "times out."

For getting responses to a PMAC control-character command it's easiest to use GetControlResponseEx().

Common Problems Experienced Using ASCII Communications Functions

This section outlines some of the more frequently encountered issues with solutions. Also see "Communication Application Notes."

Modifying Critical PMAC I-Variables

There are several I-Variables that PcommServer expects or enforces to certain values. The table below describes each and their purpose. Do not modify these I-Variables.

I Variable	Meaning	Desired Value
I3	Handshaking mode	2
I4	Checksummed Serial Communication Enable/Disable	0 or 1
I6	Error Reporting Mode	1
I63	Control-X echo	1
I64	Unsolicited Response Tagged	1

Thread-Safe ASCII Communications

PcommServer is a thread-safe communication driver. LockPmac() and ReleasePmac() are used internally with functions such as PmacGetResponseEx() and PmacDownload(). This means that two or more threads, even two or more applications, may be communicating to the same PMAC through the same method (bus, USB etc.) and not have a synchronization problem.

Unlike old Pcomm32Pro, in PcommServer the LockPmac() and ReleasePmac() functions are NOT needed and are not exported anymore.

Error Handling - ASCII Communication And Other Functions

Extended error handling is implemented within the ASCII communication routines that have the **Ex** suffix:

```
void GetResponseEx(long dwDevice, BSTR question, VARIANT_BOOL bAddLF, BSTR *pAnswer, long *pStatus);
void GetControlResponseEx(long dwDevice,, short question, BSTR *pAnswer, long *pStatus);
```

These routines now provide error status (in the most significant byte) in addition to the number of characters received (all other bytes), whereas the non-Ex routines simply return the number of characters received from PMAC.

The following error status codes exist for the ASCII communication routines: Below are all negative return codes

Mnemonic	Value	Meaning
COMM_EOT	0x80000000	An acknowledge character (ACK ASCII 9) was received indicating end of transmission from PMAC to Host PC.
COMM_TIMEOUT	0xC0000000	A timeout occurred. The time for the PC to wait for PMAC to respond had been exceeded.
COMM_BADCKSUM	0xD0000000	Used when using Checksum communication. If a bad checksum occurred this error will be returned.
COMM_ERROR	0xE0000000	Unable to communicate.
COMM_FAIL	0xF0000000	Serious failure.
COMM_ANYERR	0x70000000	Some error occurred.
COMM_UNSOLICITED	0x10000000	An unsolicited response has been received from PMAC. Usually caused by PLC's or Motion Programs that have "SEND" or "COMMAND" statements.

The mnemonics above, in addition to MACROs to parse the return value, are defined in the provided mioctl.h header file. To get at the individual portions of the return value the following MACROs are helpful:

```
#define COMM_CHARS(c)    (c & 0x0FFFFFFF) // Returns the number of characters
#define COMM_STATUS(c)   (c & 0xF0000000) // Returns the status byte
```

To check for individual error codes the MACROs below are very useful:

```
#define IS_COMM_MORE(c)     ((c & COMM_FAIL) == 0)
#define IS_COMM_EOT(c)      ((c & COMM_FAIL) == COMM_EOT)
#define IS_COMM_TIMEOUT(c)  ((c & COMM_FAIL) == COMM_TIMEOUT)
#define IS_COMM_BADCKSUM(c) ((c & COMM_FAIL) == COMM_BADCKSUM)
#define IS_COMM_ERROR(c)    ((c & COMM_FAIL) == COMM_ERROR)
#define IS_COMM_FAIL(c)     ((c & COMM_FAIL) == COMM_FAIL)
#define IS_COMM_ANYERROR(c) ((c & COMM_ANYERR) > 0)
#define IS_COMM_UNSOLICITED(c) ((c & 0xF0000000) == COMM_UNSOLICITED)
```

Using Interrupts

Interrupts are provided for both Windows 2000/XP operating systems. There is only method of interrupt notification for your program:

1. Set an event.

An event is generated by PcommServer and available for all applications communicating to the PMAC. In order to initialize an Interrupt a Mask (*ulMask*) is sent to PMAC. This parameter determines the interrupt service vector(s) to be used for the interrupt initiated by the function.

The least significant byte of *ulMask* controls which conditions will generate an interrupt. A bit value of 0 enables, 1 disables.

Bit	PMAC Signal
0	In Position of Coordinate System
1	Buffer Request (PMAC's request for more moves)
2	Error, A motor(s) in the coordinate system has had a fatal following error
3	Warning, A motor(s) in the coordinate system has had a warning following error
4	Host Request, PMAC has an ASCII response for the host
5-7	User programmable, see PMAC User's Guide, Writing a Host Communications Program

_IPmacDeviceEvents_InterruptEventHandler enables the Interrupt event function is provided to shut down the interrupt service.

Downloading To PMAC

Downloading ASCII PMAC Data

Downloading of PMAC motion, plc, configuration files etc. may be done by the using the PmacDownload() function. This function can:

- Parse Macros (i.e. #define, #include etc.)
- Compile PLC's to PLCC's
- Create a map file from macros
- Create a log file of download progress
- Invoke events for for displaying messages and progress (same text as the log file that can be created)
- Download a file or a buffer through a line retrieval call back function
- Update a progress bar through a call back function

PROGRAMMER'S REFERENCE

INTRODUCTION

The Programmers Reference of PcommServer details all of the PMAC library functions in groups of similar functionality. The description of each function includes the syntax, arguments, and status word. The groups are ordered as follows:

1. Configuration, Initialization, and Shutdown Functions
2. ASCII Communication Functions
3. Download Functions
4. DPR Real Time Fixed Data Buffer
5. DPR Variable Background Data Buffer Functions
6. DPR Binary Rotary Buffer Functions
7. DPR Numeric Read / Write Functions
8. Data Types and Structures

Important Information About Method of Communication Being Used by PcommServer

There are three methods by which PcommServer may be used to communicate to PMAC, over the Bus(ISA, PCI, USB and Ethernet only), Dual Ported Ram ASCII (ISA and PCI only), or the Serial Port. Immediately after initialization (after a call to OpenPmacDevice()) the method used depends on what is stored in the system registry (either BUS or SERIAL). To change the method used at startup, call the PmacSelect() function so that the change is saved in the registry. Alternatively, if your operating system has the ability to configure drivers go there and select the PMAC driver for configuration.

NO parameter specifying maxchar needed in PcommServer

All of the ASCII Communication functions fetch complete response until <ACK> is received.

Role change between the return value and status word

The return value in all of PcommServer functions is now S_OK (0) for success or !S_OK in case of failure. The status word is now included wherever needed a parameter of the function.

INITIALIZATION, SHUTDOWN AND CONFIGURATION FUNCTIONS

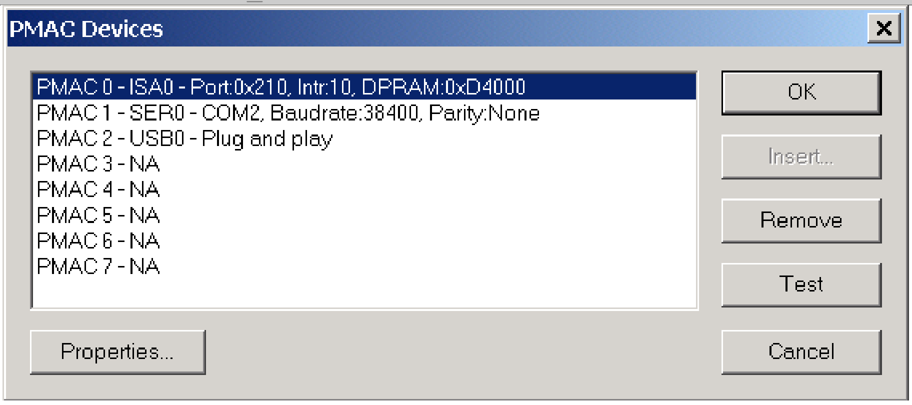
SelectDevice() Method

SelectDevice is used to add, remove and configured previously registered PMAC devices as well as change properties of previously configured device in the PMAC device list. A total of 8 devices (and up to 6 enumeration of any one device) can be configured using SelectDevice() method. Please refer to DT_Driver Installation manual for details. Following are procedures for SelectDevice() method under different development environments.

```
[Visual Basic]
Sub SelectDevice( _
    ByVal hWindow As Integer, _
    ByRef pDeviceNumber As Integer, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void SelectDevice(
    int hWindow,
    out int pDeviceNumber,
    out bool pbSuccess
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE SelectDevice(
        /* [in] */ long hWindow,
        /* [out] */ long *pDeviceNumber,
        /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```



PMAC 0 - ISA0 - Port:0x210, Intr:10, DPRAM:0xD4000
PMAC 1 - SER0 - COM2, Baudrate:38400, Parity:None
PMAC 2 - USB0 - Plug and play
PMAC 3 - NA
PMAC 4 - NA
PMAC 5 - NA
PMAC 6 - NA
PMAC 7 - NA

Provides a way to select and configure currently installed PMAC Devices. A dialog box is displayed, as shown, to allow selection and configuration of all possible PMAC devices. PMAC devices available are those whose driver has been installed. Typically this is used to allow end users of an application to pick and choose from several PMAC devices in a PC.

Arguments

hWindow	Handle to parent window for device configuration dialog.
pDeviceNumber	**Device number** >= 0 and <= 7 : Device selected
pbSuccess	**True if success**

Open() Method

This function opens a channel for your program to use the PMAC driver.

```
[Visual Basic]
Sub Open( _
    ByVal dwDevice As Integer, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void Open(
    int dwDevice,
    out bool pbSuccess
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE Open(
        /* [in] */ long dwDevice,
        /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

In order for this function to succeed, the PMAC Win32 Driver must be previously installed in the operating system. PMAC(*dwDevice*) must be registered in the environment. Then the system registry will contain the location and configuration of the PMAC specified by *dwDevice*. Open looks to the registry for this information. The registry values are located in HKLM/System/CurrentControlSet/Services/Pmac/Device(*dwDevice*).

Every *Open()* should be paired with a call to *Close()* to release the resources used by the driver.

Arguments

dwDevice	Device number.
pbSuccess	True if success.

Close() Method

This function closes the channel from your program to the PMAC driver.

```
[Visual Basic]
Sub Close( _
    ByVal dwDevice As Integer _
)
```

```
[C#]
void Close(
    int dwDevice
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE Close(
        /* [in] */ long dwDevice) = 0;
```

Arguments

dwDevice	Device number.

ASCII COMMUNICATION FUNCTIONS

GetResponseEx() Method

Most if not all of your communication with the PMAC can be handled with the GetResponseEx() function. This function will send a command string (i.e. “#1j+, “?”, “Open Prog1”, etc.) to the PMAC and retrieve and place any pending responses within a response buffer for your use. This is an efficient and *safe* function to use. GetResponseExA () always matches the command string with the response string or else it “times out.”

```
[Visual Basic]
Overridable Public Sub GetResponseEx( _
   ByVal dwDevice As Integer, _
   ByVal question As String, _
   ByVal bAddLF As Boolean, _
   ByRef pAnswer As String, _
   ByRef pstatus As Integer _
)
```

```
[C#]
virtual public void GetResponseEx(
   int dwDevice,
   string question,
   bool bAddLF,
   out string pAnswer,
   out int pstatus
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetResponseEx(
        /* [in] */ long dwDevice,
        /* [in] */ BSTR question,
        /* [in] */ VARIANT_BOOL bAddLF,
        /* [out] */ BSTR *pAnswer,
        /* [out] */ long *pStatus) = 0;
```

Arguments

dwDevice	Device number.
question	command string.
bAddLF	bool add linefeed between multiple line response
pAnswer	String buffer to copy the PMAC’s response into.
pStatus	Status word (The upper byte contains the status of the call, whereas all lower bytes contain the number of characters received from PMAC. If no characters were received from PMAC, check the upper bytes status code for a potential error code. See “Error Handling - ASCII Communication” for a detailed explanation.

GetControlResponseEx() Method

GetControlResponse*Ex*() sends a control character to PMAC and potentially returns the ASCII response from PMAC, similar to GetResponse*Ex*().

```
[Visual Basic]
Overridable Public Sub GetControlResponseEx( _
   ByVal dwDevice As Integer, _
   ByVal question As Short, _
   ByRef pAnswer As String, _
   ByRef pstatus As Integer _
)
```

```
[C#]
virtual public void GetControlResponseEx(
   int dwDevice,
   short question,
   out string pAnswer,
   out int pstatus
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetControlResponseEx(
        /* [in] */ long dwDevice,
        /* [in] */ short question,
        /* [out] */ BSTR *pAnswer,
        /* [out] */ long *pStatus) = 0;
```

Arguments

dwDevice	Device number.
question	ASCII code of control character
pAnswer	String buffer. PMAC's response is placed there by function.
pStatus	Status word (The upper byte contains the status of the call, whereas all lower bytes contain the number of characters received from PMAC. If no characters were received from PMAC, check the upper bytes status code for a potential error code. See "Error Handling - ASCII Communication" for a detailed explanation.

RawGetResponseEx Method

Following two methods perform the exact task of GetResponseEx() and GetControlResponseEx() respectively, except that these return raw data and do not truncate any control character such as <ACK>, <BELL>, <CTRL_X> etc.

```
[Visual Basic]
Overridable Public Sub RawGetResponseEx( _
   ByVal dwDevice As Integer, _
   ByVal question As String, _
   ByVal bAddLF As Boolean, _
   ByRef pAnswer As String, _
   ByRef pstatus As Integer _
)
```

```
[C#]
virtual public void RawGetResponseEx(
   int dwDevice,
   string question,
   bool bAddLF,
   out string pAnswer,
   out int pstatus
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE RawGetResponseEx(
        /* [in] */ long dwDevice,
        /* [in] */ BSTR question,
        /* [in] */ VARIANT_BOOL bAddLF,
        /* [out] */ BSTR *pAnswer,
        /* [out] */ long *pStatus) = 0;
```

Arguments

dwDevice	Device number.
question	command string.
bAddLF	bool add linefeed between multiple line response
pAnswer	String buffer to copy the PMAC's response into.
pStatus	Status word (The upper byte contains the status of the call, whereas all lower bytes contain the number of characters received from PMAC. If no characters were received from PMAC, check the upper bytes status code for a potential error code. See "Error Handling - ASCII Communication" for a detailed explanation.

RawGetControlResponseEx() Method

```
[Visual Basic]
Overridable Public Sub RawGetControlResponseEx( _
    ByVal dwDevice As Integer, _
    ByVal question As Short, _
    ByRef pAnswer As String, _
    ByRef pstatus As Integer _
)
```

```
[C#]
virtual public void RawGetControlResponseEx(
    int dwDevice,
    short question,
    out string pAnswer,
    out int pstatus
);
```

```
[C++]
virtual /*[helpstring][id]*/ HRESULT STDMETHODCALLTYPE RawGetControlResponseEx(
        /* [in] */ long dwDevice,
        /* [in] */ short question,
        /* [out] */ BSTR *pAnswer,
        /* [out] */ long *pStatus) = 0;
```

Arguments

dwDevice	Device number.

question	ASCII code of control character
pAnswer	String buffer. PMAC's response is placed there by function.
pStatus	Status word (The upper byte contains the status of the call, whereas all lower bytes contain the number of characters received from PMAC. If no characters were received from PMAC, check the upper bytes status code for a potential error code. See "Error Handling - ASCII Communication" for a detailed explanation.

GetResponseProgress() Method

For functions returning more than one line response the use has the ability get the progress of how much data has been captured. This function generates the progress event returning the current line number at an interval of 10msec. It is users responsibility to determine the totoal number of lines in the response and distribute the progress bar evenly. This progress is then available and can easily be handled by event handler.

```
[Visual Basic]
Sub GetResponseProgress( _
    ByVal dwDevice As Integer, _
    ByVal question As String, _
    ByVal bAddLF As Boolean, _
    ByRef pAnswer As String, _
    ByRef pstatus As Integer _
)
```

```
[C#]
void GetResponseProgress(
    int dwDevice,
    string question,
    bool bAddLF,
    out string pAnswer,
    out int pstatus
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetResponseProgress(
            /* [in] */ long dwDevice,
            /* [in] */ BSTR question,
            /* [in] */ VARIANT_BOOL bAddLF,
            /* [out] */ BSTR *pAnswer,
            /* [out] */ long *pStatus) = 0;
```

Arguments

dwDevice	Device number.
question	command string.
bAddLF	bool add linefeed between multiple line response
pAnswer	String buffer to copy the PMAC's response into.
pStatus	Status word (The upper byte contains the status of the call, whereas all lower bytes contain the number of characters received from PMAC. If no characters were received from PMAC, check the upper bytes status code for a potential error code. See "Error Handling - ASCII Communication" for a detailed explanation.

Abort() Method

While the GetReponseProgress is running user has the ability to abort it using the Abort() method. Abort() will flush() any remaining data in the port.

```
[Visual Basic]
Sub Abort( _
   ByVal dwDevice As Integer _
)
```

```
[C#]
void Abort(
   int dwDevice
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE Abort(
               /* [in] */ long dwDevice) = 0;
```

Arguments:

dwDevice	Device number.

GetPmacType() Method

Returns a corresponding number of the new PmacType in the DEVPMACTYPE parameter.

```
[Visual Basic]
Overridable Public Sub GetPmacType( _
   ByVal dwDevice As Integer, _
   ByRef pVal As E:DEVPMACTYPE _
)
```

```
[C#]
virtual public void GetPmacType(
   int dwDevice,
   out E:DEVPMACTYPE pVal
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetPmacType(
        /* [in] */ long dwDevice,
        /* [out] */ DEVPMACTYPE *pVal) = 0;
```

Arguments

dwDevice	Device number.
DEVPMACTYPE	Number cossponding to Pmactype. See the PMACType Structure for updated PMAC Types.

DEVPMACTYPE Enumeration

```
[Visual Basic]
Public Enum DEVPMACTYPE
```

```
[C#]
public enum DEVPMACTYPE
```

Namespace: PCOMMSERVERLib

Assembly: Interop.PCOMMSERVERLib (in interop.pcommserverlib.dll)

Members

Member name	Description
DEV_PT_PMAC1	PMAC1 (Non-Turbo)
DEV_PT_PMAC2	PMAC2 (Non-Turbo)
DEV_PT_PMACUL	PMAC Ultralight (Non-Turbo)
DEV_PT_GEOPMAC	Geo PMAC (Non-Turbo)
DEV_PT_PMAC	Barrier between Non-Turbo and Turbo PMACs
DEV_PT_PMAC1T	PMAC1 Turbo
DEV_PT_PMAC2T	PMAC2 Turbo
DEV_PT_PMACUT	PMAC Turbo Ultralite
DEV_PT_UMAC	UMAC Turbo
DEV_PT_QMAC	QMAC Turbo
DEV_PT_PMAC1TSM	PMAC1 Turbo Small Memory
DEV_PT_PMAC2TSM	PMAC2 Turbo Small Memory

```
[C++]
typedef /* [public] */
enum DEVPMACTYPE
    { DEV_PT_PMAC1       = 1,
      DEV_PT_PMAC2       = DEV_PT_PMAC1 + 1,
      DEV_PT_PMACUL      = DEV_PT_PMAC2 + 1,
      DEV_PT_GEOPMAC     = DEV_PT_PMACUL + 1,
      DEV_PT_PMAC        = DEV_PT_GEOPMAC + 1,
      DEV_PT_PMAC1T      = DEV_PT_PMAC + 1,
      DEV_PT_PMAC2T      = DEV_PT_PMAC1T + 1,
      DEV_PT_PMACUT      = DEV_PT_PMAC2T + 1,
      DEV_PT_UMAC        = DEV_PT_PMACUT + 1,
      DEV_PT_QMAC        = DEV_PT_UMAC + 1,
      DEV_PT_PMAC1TSM    = DEV_PT_QMAC + 1,
      DEV_PT_PMAC2TSM    = DEV_PT_PMAC1TSM + 1
    }       DEVPMACTYPE;
```

GetPmacLocation() Method

Returns the corresponding number for a specific location of PMAC, I. e., ISA, PCI, USB, Ethernet or Serial.

```
[Visual Basic]
Overridable Public Sub GetPmacLocation( _
   ByVal dwDevice As Integer, _
   ByRef pVal As E:DEVLOCATIONTYPE _
)
```

```
[C#]
virtual public void GetPmacLocation(
    int dwDevice,
    out E:DEVLOCATIONTYPE pVal
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetPmacLocation(
         /* [in] */ long dwDevice,
         /* [out] */ DEVLOCATIONTYPE *pVal) = 0;
```

DEVLOCATIONTYPE Enumeration

```
[Visual Basic]
Public Enum DEVLOCATIONTYPE
```

```
[C#]
public enum DEVLOCATIONTYPE
```

Namespace: **PCOMMSERVERLib**

Assembly: Interop.PCOMMSERVERLib (in interop.pcommserverlib.dll)

Members

Member name	Description
DEV_LT_UNKNOWN	Default number (initialized value)
DEV_LT_ISA	ISA Bus
DEV_LT_SER	Serial Port
DEV_LT_ETH	Ethernet Port
DEV_LT_PCI	PCI Bus
DEV_LT_USB	USB port
DEV_LT_LAST	Next possible port

```
[C++]
typedef /* [public] */
enum DEVLOCATIONTYPE
     { DEV_LT_UNKNOWN     = 0,
       DEV_LT_ISA   = DEV_LT_UNKNOWN + 1,
       DEV_LT_SER   = DEV_LT_ISA + 1,
       DEV_LT_ETH   = DEV_LT_SER + 1,
       DEV_LT_PCI   = DEV_LT_ETH + 1,
       DEV_LT_USB   = DEV_LT_PCI + 1,
       DEV_LT_LAST = DEV_LT_USB + 1
     }       DEVLOCATIONTYPE;
```

SetChecksums() Method

To enable or disable serial checksummed communications call SetChecksums().

```
[Visual Basic]
Overridable Public Sub SetChecksums( _
   ByVal dwDevice As Integer, _
   ByVal bActive As Boolean _
)
```

```
[C#]
virtual public void SetChecksums(
   int dwDevice,
   bool bActive
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE SetChecksums(
        /* [in] */ long dwDevice,
        /* [in] */ VARIANT_BOOL bActive) = 0;
```

Arguments

dwDevice	Device number.
bActive	bool SetChecksum or clear.

Note: Checksums are only applied to Serial communication and once enabled Checksums are verified for all calls to PMAC.

DPRAvailable(), DPRSize() properties

```
[Visual Basic]
Overridable Public ReadOnly Property DPRAvailable As Boolean
Overridable Public ReadOnly Property DPRSize As Integer
```

```
[C#]
virtual public bool DPRAvailable {get;}
virtual public int DPRSize {get;}
```

```
[C++]
virtual /* [helpstring][id][propget] */ HRESULT STDMETHODCALLTYPE
get_DPRAvailable(
        /* [in] */ long dwDevice,
        /* [retval][out] */ VARIANT_BOOL *pVal) = 0;
virtual /* [helpstring][id][propget] */ HRESULT STDMETHODCALLTYPE get_DPRSize(
        /* [in] */ long dwDevice,
        /* [retval][out] */ long *pVal) = 0;
```

These functions determines if dual ported RAM is available for use by your application and the second function determines the actual size of DPRAM.

Arguments

dwDevice	Device number.
pVal	True if success.
pVal (long)	Size in PMAC words.

GetAsciiComm(), SetAsciiComm() Methods

```
[Visual Basic]
Overridable Public Sub GetAsciiComm( _
    ByVal dwDevice As Integer, _
    ByRef pVal As E:DEVASCIIMODE _
)
Overridable Public Sub SetAsciiComm( _
    ByVal dwDevice As Integer, _
    ByVal newVal As E:DEVASCIIMODE _
)
```

```
[C#]
virtual public void SetAsciiComm(
    int dwDevice,
    E:DEVASCIIMODE newVal
);
virtual public void GetAsciiComm(
    int dwDevice,
    out E:DEVASCIIMODE pVal
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetAsciiComm(
        /* [in] */ long dwDevice,
        /* [out] */ DEVASCIIMODE *pVal) = 0;
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE SetAsciiComm(
        /* [in] */ long dwDevice,
        /* [in] */ DEVASCIIMODE newVal) = 0;
```

Returns / sets the current or desired communications mode. (BUS or DPRAM) as a parameter. Use PmacSetAsciiComm() to set this value and switch between BUS and DPRAM ASCII communication.

Arguments

dwDevice	Device number.
pVal	DEVASCIIMODE.

DEVASCIIMODE Enumeration

```
[C++]
typedef /* [public] */
enum DEVASCIIMODE
    { DEV_BUS      = 0,
      DEV_DPR      = DEV_BUS + 1
    }       DEVASCIIMODE;
```

In addition to above ASCII functions, following special commands have been added for robust communication between the host computer and PMAC devices.

put_USMonitoringEnabled() and put_USMonitoringPeriod() Methods

Use these methos to enable/disable the unsolicited response and set the monitoring period in the PcommServer.

```
[Visual Basic]
Property USMonitoringEnabled As Boolean

Property USMonitoringPeriod As Integer
```

```
[C#]
bool USMonitoringEnabled {get;set;}
int USMonitoringPeriod {get;set;}
```

```
 [C++]
put_USMonitoringEnabled(
            /* [in] */ long dwDevice,
            /* [in] */ VARIANT_BOOL newVal) = 0;
put_USMonitoringPeriod(
            /* [in] */ long dwDevice,
            /* [in] */ long newVal) = 0;
[propput, id(246), helpstring("property USMonitoringEnabled")] HRESULT
USMonitoringEnabled([in]long dwDevice, [in] VARIANT_BOOL newVal);
      [propput, id(247), helpstring("property USMonitoringPeriod")] HRESULT
USMonitoringPeriod([in]long dwDevice, [in] long newVal);
```

Arguments

dwDevice	Device number.
newVal	bool enable or disable
newVal	Period in msec

PhaseMotor() Method

This commands sends the command “$” or “#n$” (where n is the motor number) and waits for response from PMAC. Will return either with and <ACK> meaning that command was successful or an <BELL> followed by the error string.

```
[Visual Basic]
Overridable Public Sub PhaseMotor( _
   ByVal dwDevice As Integer, _
   ByVal question As String, _
   ByRef pStatuss As Integer _
)
```

```
[C#]
virtual public void PhaseMotor(
   int dwDevice,
   string question,
   out int pStatuss
);
```

```
[C++]
```

```
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE PhaseMotor(
        /* [in] */ long dwDevice,
        /* [in] */ BSTR question,
        /* [out] */ long *pStatuss) = 0;
```

Arguments

dwDevice	Device number.
question	command string.
pStatus	Status word (The upper byte contains the status of the call, whereas all lower bytes contain the number of characters received from PMAC. If no characters were received from PMAC, check the upper bytes status code for a potential error code. See "Error Handling - ASCII Communication" for a detailed explanation.

PmacReset() Method

PmacReset handles both "$$$" as well as golobal reset "$$$***" commands. For global reset it sets up critical I-variables to optimal values "I3=2 I6=1 I63=1 I64=1".

```
[Visual Basic]
Overridable Public Sub PmacReset( _
   ByVal dwDevice As Integer, _
   ByVal question As String, _
   ByVal bAsciiRingComm As Boolean, _
   ByRef pstatus As Integer _
)
```

```
[C#]
virtual public void PmacReset(
   int dwDevice,
   string question,
   bool bAsciiRingComm,
   out int pstatus
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE PmacReset(
        /* [in] */ long dwDevice,
        /* [in] */ BSTR question,
        /* [in] */ VARIANT_BOOL bAsciiRingComm,
        /* [out] */ long *pStatus) = 0;
```

Arguments

dwDevice	Device number.
question	command string.
bAsciiRingComm	bool, must specify if the MACRO Ring ASCII is ON or OFF.
pStatus	Status word (The upper byte contains the status of the call, whereas all lower bytes contain the number of characters received from PMAC. If no characters were received from PMAC, check the upper bytes status code for a potential error code. See "Error Handling - ASCII Communication" for a detailed explanation.

PmacSave() Method

PmacSave issues a "save" command and waits upto 30 seconds for and ACK from PMAC. If a PMAC does net respond within 30 seconds then returns a timout other a success message returns.

```
[Visual Basic]
Overridable Public Sub PmacSave( _
    ByVal dwDevice As Integer, _
    ByVal question As String, _
    ByRef pstatus As Integer _
)
```

```
[C#]
virtual public void PmacSave(
    int dwDevice,
    string question,
    out int pstatus
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE PmacSave(
        /* [in] */ long dwDevice,
        /* [in] */ BSTR question,
        /* [out] */ long *pStatus) = 0;
```

Arguments

dwDevice	Device number.
question	command string.
pStatus	Status word (The upper byte contains the status of the call, whereas all lower bytes contain the number of characters received from PMAC. If no characters were received from PMAC, check the upper bytes status code for a potential error code. See "Error Handling - ASCII Communication" for a detailed explanation.

DPRTest() Method

Following three methods provide the DPR test procedure.

```
[Visual Basic]
Sub DPRTest( _
    ByVal dwDevice As Integer, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void DPRTest(
    int dwDevice,
    out bool pbSuccess
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRTest(
            /* [in] */ long dwDevice,
            /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

Arguments

dwDevice	Device number.
pbSuccess	True if success.

AbortTest() Method

```
[Visual Basic]
Sub AbortTest( _
    ByVal dwDevice As Integer _
)
```

```
[C#]
void AbortTest(
    int dwDevice
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE AbortTest(
              /* [in] */ long dwDevice) = 0;
```

Arguments

dwDevice	Device number.

get_DPRTesting() Method

```
[Visual Basic]
Function get_DPRTesting( _
    ByVal dwDevice As Integer _
) As Boolean
```

```
[C#]
bool get_DPRTesting(
    int dwDevice
);
```

Implements get_DPRTesting

```
[C++]
[id(262), helpstring("method get_DPRTesting")] HRESULT get_DPRTesting([in]long
dwDevice, [out, retval] VARIANT_BOOL *pVal);
```

Arguments

dwDevice	Device number.
pVal	Pointer to successful start of DPRTest thread

Note: All messages and progress events are generated from the PcommServer and available for user at the interface.

DOWNLOADING TO PMAC

Download() Method

This function takes an ASCII file, processes it, and downloads it from the PC to the PMAC. Processing includes MACRO parsing and compiling PLCs, for example. This function can generate several residual files, as described in the table below.

File name	Usage
Filename.EXT	Original file with the original EXTension (should not be *.PMA, *.56K, *.LOG, *.MAP).
Filename.PMA	After parsing the file for #define, #includes and other MACRO's this file is generated. It may be downloaded if no compiling is necessary.
Filename.56K	This file will be created if the Filename.PMA was compiled. Compilation occurs when the *macro* parameter is set to TRUE.
Filename.LOG	The status of the download at each stage is recorded when the *log* parameter is set to TRUE.
Filename.MAP	A lookup table is created when MACRO definitions exist. They are recorded and saved to a file when the *map* parameter is set to TRUE.

```
[Visual Basic]
Overridable Public Sub Download( _
   ByVal dwDevice As Integer, _
   ByVal filePath As String, _
   ByVal bMacro As Boolean, _
   ByVal bMap As Boolean, _
   ByVal bLog As Boolean, _
   ByVal bDnld As Boolean, _
   ByRef pbSuccess As Boolean _
)
```

```
[C#]
virtual public void Download(
   int dwDevice,
   string filePath,
   bool bMacro,
   bool bMap,
   bool bLog,
   bool bDnld,
   out bool pbSuccess
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE Download(
        /* [in] */ long dwDevice,
        /* [in] */ BSTR filePath,
        /* [in] */ VARIANT_BOOL bMacro,
        /* [in] */ VARIANT_BOOL bMap,
        /* [in] */ VARIANT_BOOL bLog,
        /* [in] */ VARIANT_BOOL bDnld,
        /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

Arguments

dwDevice	Device number.
filePath	Path of file to download.
bMacro	Flag to parse for macros.
bMap	Flag to create a map file created from macros.
bLog	Flag to create a log file. This is the same messages as sent to the “msgp” procedure.
bDnld	Flag indicating to send final parsed file to the PMAC.
pbSuccess	Pointer to successful start of download thread

Downloading() Method

```
[Visual Basic]
Overridable Public ReadOnly Property Downloading As Boolean
```

```
[C#]
virtual public bool Downloading {get;}
```

Implements get_Downloading

```
[C++]
virtual /* [helpstring][id][propget] */ HRESULT STDMETHODCALLTYPE
get_Downloading(
        /* [in] */ long dwDevice,
        /* [retval][out] */ VARIANT_BOOL *pVal) = 0;
```

Arguments

dwDevice	Device number.
pVal	Pointer to successful start of download thread

AbortDownload() Method

Calling this function will cause a download in progress to be aborted. This applies for driver downloading functions Download().

```
[Visual Basic]
Overridable Public Sub AbortDownload( _
    ByVal dwDevice As Integer _
)
```

```
[C#]
virtual public void AbortDownload(
    int dwDevice
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE AbortDownload(
              /* [in] */ long dwDevice) = 0;
```

Arguments

dwDevice	Device number.

EVENTS SUPPORT IN PCOMMSERVER

PcommServer's event handler routine handles following events:

Message Event

Message event gives out miscellaneous messages including upload/download messages, log messages which are raised during parsing files etc.

Progress Event

Progress event sends the progress of the function (in percentage) back to the application. These percentage numbers are generated during download, GetProgressResponse upload and TestDPRAM function.

Error Events

Error event handles errors generated during normal communication between the PcommServer library and attached applications. These errors cover all errors including all PMAC command errors (1-20), Watchdog error, USB/Ethernet Unplug error and other communication errors such as timeout, badchecksum, Comm fail etc.

Unsolicited Reponse

Unsolicited response is handled by a "Unsolicited Response" and attached applications have the freedom to capture and display and incoming unsolicited response message.

Interrupts

Finally, interrupts messages are handled by a separate "interrupt event".

Note: Please see examples in VB, C# and C++ on how to capture and display these events.

DATA GATHERING FUNCTIONS

Following set of functions handle data gathering from PMAC including all of the configuration related to number of samples, sample period, start and stop gather.

In order to use following data gather methods refer to the Gather structures and VB 6.0 and VB.NET examples at the start of the manual.

Gather Structures

```
typedef struct DEVWTG_EX
  {
    UINT DEVCOM_TO_G;
    UINT DEVENC_TO_G;
    UINT DEVDAC_TO_G;
    UINT DEVCUR_TO_G;
  } DEVWTG_EX;

  typedef struct DEVGATHER_HEADER
  {
    DWORD  size;                         // Size of this header
    double ulGatherSampleTime;           // Sample gather time in msec
    UINT   uGatherPeriod;                // I19 number servo cycles per sample
    DWORD  dwGatherMask;                 // I20 (determines #sources & types)
    DWORD  dwGatherMask2;                // added for Turbo
    UINT   uGatherSources;               // Number of sources gathered
    UINT   uGatherSamples;               // Number of samples gathered
    UINT   uGatherSampleLen;             // Number 24-bit words per sample
    BOOL   bGatherEnabled[48];           // Sources enabled
    char   szGatherAdr[48 ][15];         // Types and addresses of gathers
    UINT   uGatherSize[48];              // Size of gather type in 24bit words
    double *pGatherData[48];             // Pointers to gathered data
    double dGatherScale[48];             // Scale values for data
  } DEVGATHER_HEADER;
```

StartGather, StopGather Methods

```
[Visual Basic]
Sub StartGather( _
   ByVal dwDevice As Integer, _
   ByRef pbSuccess As Boolean _
)
Sub StopGather( _
   ByVal dwDevice As Integer _
)
```

```
[C#]
void StartGather(
   int dwDevice,
   out bool pbSuccess
);
void StopGather(
   int dwDevice
);
```

```
[C++]
      STDMETHOD(StopGather)(/*[in]*/long dwDevice);
      STDMETHOD(StartGather)(/*[in]*/long dwDevice,/*[out]*/VARIANT_BOOL
*pbSuccess);
```

CollectGatherData Method

```
[Visual Basic]
Overridable Public Sub CollectGatherData( _
    ByVal dwDevice As Integer, _
    ByRef pSources As Integer, _
    ByRef pSamples As Integer, _
    ByRef pbSuccess As Boolean _
) Implements !PmacDevice.CollectGatherData
```

```
[C#]
virtual public void CollectGatherData(
    int dwDevice,
    out int pSources,
    out int pSamples,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(CollectGatherData)(/*[in]*/long dwDevice,/*[out]*/long
*pSources,/*[out]*/long *pSamples,/*[out]*/VARIANT_BOOL *pbSuccess);
```

GetGatherSamples Method

```
[Visual Basic]
Overridable Public Sub GetGatherSamples( _
    ByVal dwDevice As Integer, _
    ByVal sourceNum As Integer, _
    ByRef pVariant As Object, _
    ByRef pbSuccess As Boolean _
) Implements !PmacDevice.GetGatherSamples
```

```
[C#]
virtual public void GetGatherSamples(
    int dwDevice,
    int sourceNum,
    out object pVariant,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(GetGatherSamples)(/*[in]*/long dwDevice,/*[in]*/long
sourceNum,/*[out]*/VARIANT *pVariant,/*[out]*/VARIANT_BOOL *pbSuccess);
```

GetNumGatherSources Method

```
[Visual Basic]
Overridable Public Sub GetNumGatherSources( _
    ByVal dwDevice As Integer, _
    ByRef pVal As Integer _
) Implements !PmacDevice.GetNumGatherSources
```

```
[C#]
virtual public void GetNumGatherSources(
    int dwDevice,
    out int pVal
);
```

```
[C++]
STDMETHOD(GetNumGatherSources)(/*[in]*/long dwDevice,/*[out]*/long *pVal);
```

SetGather Method

```
[Visual Basic]
Overridable Public Sub SetGather( _
    ByVal dwDevice As Integer, _
    ByVal num As Integer, _
    ByVal str As String, _
    ByVal bEnable As Boolean, _
    ByRef pbSuccess As Boolean _
) Implements !PmacDevice.SetGather
```

```
[C#]
virtual public void SetGather(
    int dwDevice,
    int num,
    string str,
    bool bEnable,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(SetGather)(/*[in]*/long dwDevice,/*[in]*/long num,/*[in]*/BSTR
str,/*[in]*/VARIANT_BOOL bEnable,/*[out]*/VARIANT_BOOL *pbSuccess);
```

GetGather Method

```
[Visual Basic]
Sub GetGather( _
    ByVal dwDevice As Integer, _
    ByVal num As Integer, _
    ByRef pStr As String, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void GetGather(
    int dwDevice,
    int num,
    out string pStr,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(GetGather)(/*[in]*/long dwDevice,/*[in]*/long num,/*[in,out]*/BSTR
*pStr,/*[out]*/VARIANT_BOOL *pbSuccess);
```

SetQuickGather Method

```
[Visual Basic]
Sub SetQuickGather( _
    ByVal dwDevice As Integer, _
    ByVal lComMask As Integer, _
    ByVal lEncMask As Integer, _
    ByVal lDacMask As Integer, _
    ByVal lCurMask As Integer, _
    ByVal bEnable As Boolean, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void SetQuickGather(
    int dwDevice,
    int lComMask,
    int lEncMask,
    int lDacMask,
    int lCurMask,
    bool bEnable,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(SetQuickGather)(/*[in]*/long dwDevice,/*[in]*/long
lComMask,/*[in]*/long lEncMask,/*[in]*/long lDacMask,/*[in]*/long
lCurMask,/*[in]*/VARIANT_BOOL bEnable,/*[out]*/VARIANT_BOOL *pbSuccess);
```

Arguments

dwDevice	Device number.
lComMask, lEncMask, lDacMask, lCurMask	Mask s for sources to be collected
bEnable	Whether to collect the data or not?
pbSuccess	Pointer to successful completion of function

SetQuickGatherWithDirectCurrent Method

```
[Visual Basic]
Sub SetQuickGatherWithDirectCurrent( _
    ByVal dwDevice As Integer, _
    ByVal lComMask As Integer, _
    ByVal lEncMask As Integer, _
    ByVal lDacMask As Integer, _
    ByVal lCurMask As Integer, _
    ByVal bEnable As Boolean, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void SetQuickGatherWithDirectCurrent(
    int dwDevice,
    int lComMask,
    int lEncMask,
    int lDacMask,
    int lCurMask,
    bool bEnable,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(SetQuickGatherWithDirectCurrent)(/*[in]*/long dwDevice,/*[in]*/long
lComMask,/*[in]*/long lEncMask,/*[in]*/long lDacMask,/*[in]*/long
lCurMask,/*[in]*/VARIANT_BOOL bEnable,/*[out]*/VARIANT_BOOL *pbSuccess);
```

ClearGather, ClearGatherData Methods

```
[Visual Basic]
Overridable Public Sub ClearGather( _
    ByVal dwDevice As Integer _
) Implements !PmacDevice.ClearGather
Overridable Public Sub ClearGatherData( _
    ByVal dwDevice As Integer _
) Implements !PmacDevice.ClearGatherData
```

```
[C#]
virtual public void ClearGather(
    int dwDevice
);
virtual public void ClearGatherData(
    int dwDevice
);
```

```
[C++]
STDMETHOD(ClearGather)(/*[in]*/long dwDevice);
STDMETHOD(ClearGatherData)(/*[in]*/long dwDevice);
```

InitGather Method

```
[Visual Basic]
Overridable Public Sub InitGather( _
    ByVal dwDevice As Integer, _
    ByVal size As Integer, _
    ByVal msec As Double, _
    ByRef pbSuccess As Boolean _
) Implements !PmacDevice.InitGather
```

```
[C#]
virtual public void InitGather(
    int dwDevice,
    int size,
    Double msec,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(InitGather)(/*[in]*/long dwDevice,/*[in]*/long size,/*[in]*/double
msec,/*[out]*/VARIANT_BOOL *pbSuccess);
```

SetCurrentGather

```
[Visual Basic]
Sub SetCurrentGather( _
    ByVal dwDevice As Integer, _
    ByVal mask As Integer, _
    ByVal bEnable As Boolean, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void SetCurrentGather(
    int dwDevice,
    int mask,
    bool bEnable,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(SetCurrentGather)(/*[in]*/long dwDevice,/*[in]*/long
mask,/*[in]*/VARIANT_BOOL bEnable,/*[out]*/VARIANT_BOOL *pbSuccess);
```

GetGatherPoint Method

```
[Visual Basic]
Sub GetGatherPoint( _
    ByVal dwDevice As Integer, _
    ByVal sourceNum As Integer, _
    ByVal sampleNum As Integer, _
    ByRef pVal As Double, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void GetGatherPoint(
    int dwDevice,
    int sourceNum,
    int sampleNum,
    out Double pVal,
    out bool pbSuccess
);
```

```
[C++]
 STDMETHOD(GetGatherPoint)(/*[in]*/long dwDevice,/*[in]*/long
sourceNum,/*[in]*/long sampleNum,/*[out]*/double *pVal,/*[out]*/VARIANT_BOOL
*pbSuccess);
```

Please see the VB 6.0 and VB.NET Data Gather Example project for details on how to uses the above functions.

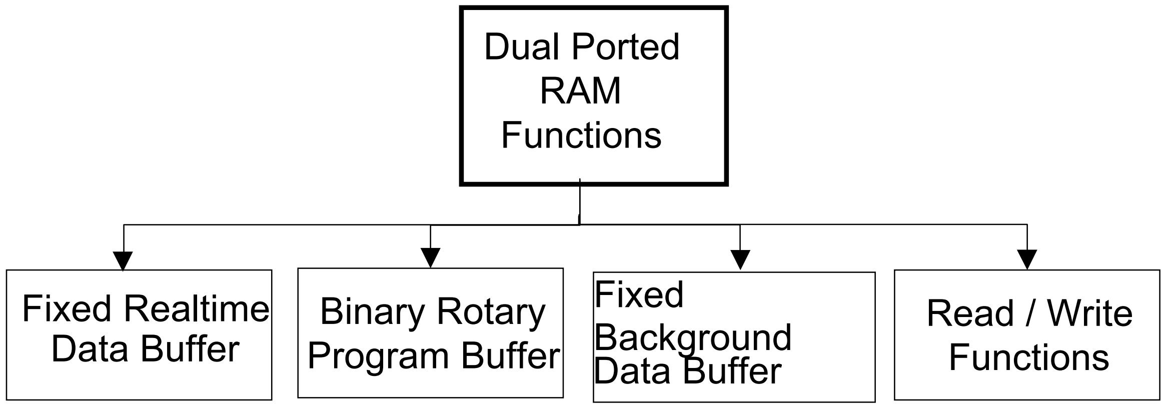
PCOMMSERVER DPR FEATURES

A Global View of the DPR Support Functions

The majority of the functions in this library are Dual Ported RAM (DPR) support functions. Their descriptions in the reference manual have been grouped by functionality as shown below.

Note:

The Automatic DPR Features may be disabled by the PmacSelect() dialog (select device then push the properties button). PcommServer actually performs the update for all applications.


Dual Ported RAM Functions
Fixed Realtime Data Buffer
Binary Rotary Program Buffer
Fixed Background Data Buffer
Read / Write Functions

Dual Ported RAM Support Functionality

Fixed Real Time Data buffer

PMAC has an automatic DPR feature, the Fixed Real Time Data Buffer, in which PMAC continually updates a specific area of DPR with a fixed data structure. This data structure is full of meaningful motor (some non-motor information in Non-Turbo PMAC) information and can be accessed by a host application to show positions, velocities etc. in real time. The data in this feature gets updated in PMAC's Real Time Interrupt period.

Fixed Background Data Buffer

This automatic DPR feature is similar to the Fixed Real Time Data Buffer in that a fixed data structure is copied by PMAC into a specific area of the DPR. The difference is that the information is coordinate-system specific, and the information is updated in PMAC's background cycle.

Binary Rotary Buffer

This PMAC DPR automatic feature can be used to efficiently download large part programs to PMAC's internal rotary buffer. The routines included to support this feature convert the PMAC ASCII to PMAC Binary before placing the code in the DPR for PMAC to retrieve.

Read/Write Functions

Numeric transfers of 16-and 32-bit wide numbers may be read or written to with this set of PcommServer routines. Floating point values are supported for 32-bit transfers. In addition, several helper routines exist for setting individual bits of DPR memory.

There is a distinction between “initializing” and “turning on” a DPR feature. Some DPR features require two actions to be taken before they are “running.” First you initialize the feature (i.e. for the DPR Rotary buffer call the *PmacDPRRotBufChange()*). Once initiated, the feature is turned off or on with a different function call (i.e. for the DPR Rotary buffer call the *PmacDPRRotBuf()*).

In addition to initialization and shutdown order being important, the order in which a program turns on the DPR features is also critical. What you need to know about the order of turning features on is: If you are using more than one DPR automatic feature, always turn on the DPR Binary Rotary Buffer last. The reason for this is that a call to *PmacDPRRotBuf()* that turns the feature on will open a PMAC program buffer (i.e. the PMAC ASCII command “&1 Open rot”). When a PMAC program buffer is open any attempt to initialize or enable other DPR features will fail (since the driver has to set I-variables to enable a feature, and if a buffer is open the I-variable assignments won’t get processed, but rather stored in the buffer).

Configuring DPR Real Time Fixed and Background Fixed Data Buffers

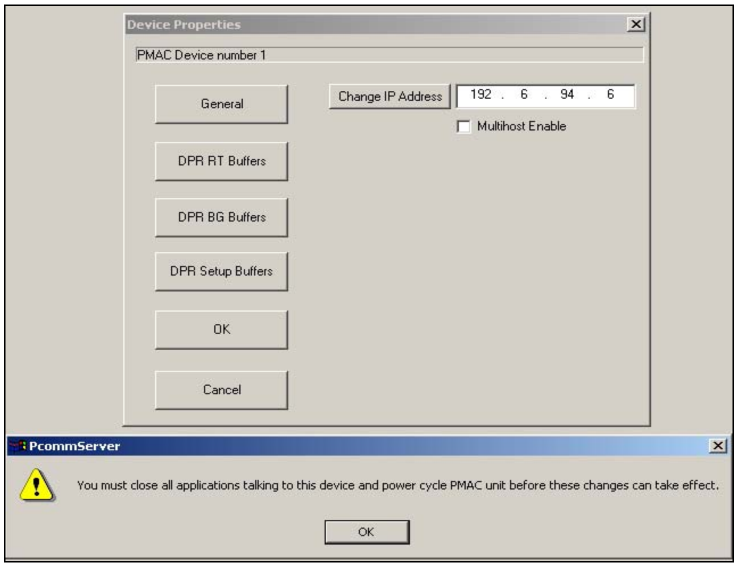
For both RealTime and Background Data buffers the initialization is now done via the properties tab the PmacSelect() dialog. Please see that following properties tab which explains on turning ON/OFF both these buffers, setting up the motor mask and setting the monitor period for these data reporting options.

The same tab is responsible for setting the size of Binary rotary buffers along with their sizes. In the background, following functions are being used to setup these parameters.

Startup/ShutDown and Handshaking

In order to configure DPRAM Realtime and Background Automatic Data reporting functions following properties dialogbox is launched from the properties tab of the selectdevice dialog box. For ISA/PCI/USB and Ethernet modes of communication DPRAM peoperties are categorized in four sections.

Genreral properties include setting Interrupts enable/disable checkbox and IP address change as well as Multihost check box.

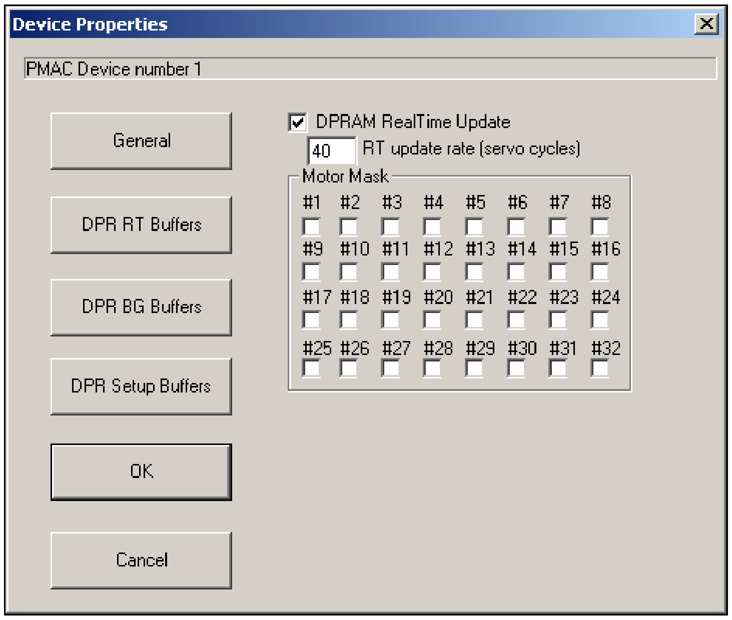


You must close all applications talking to this device and power cycle PMAC unit before these changes can take effect.

Note:

For IP Change to take effect user must close all applications and powercycle PMAC unit before these change can take effect.

DPR RT Buffers DPRAM RealTime Automatic Data Reporting Buffers Setup includes enable/disable checkbox along with monitor perion in servo cycles and the motor mask setup check boxes.



PMAC Device number 1

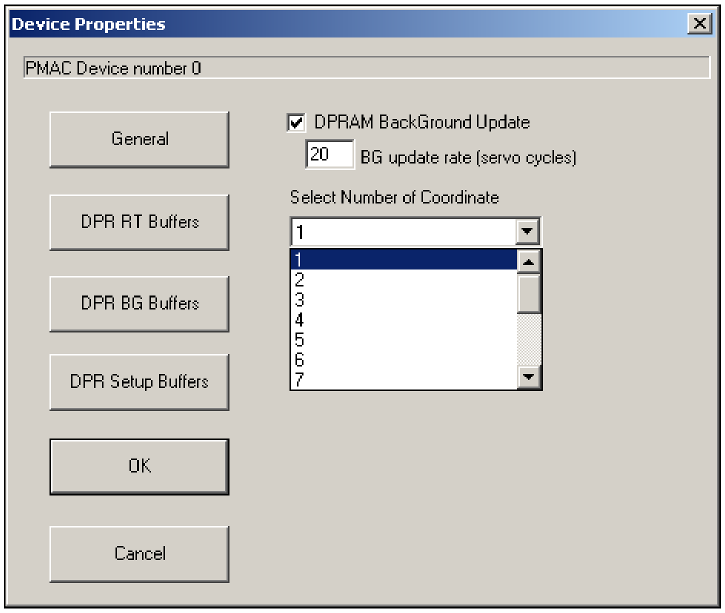
☑ DPRAM RealTime Update

40 RT update rate (servo cycles)

Motor Mask

#1 ☐	#2 ☐	#3 ☐	#4 ☐	#5 ☐	#6 ☐	#7 ☐	#8 ☐
#9 ☐	#10 ☐	#11 ☐	#12 ☐	#13 ☐	#14 ☐	#15 ☐	#16 ☐
#17 ☐	#18 ☐	#19 ☐	#20 ☐	#21 ☐	#22 ☐	#23 ☐	#24 ☐
#25 ☐	#26 ☐	#27 ☐	#28 ☐	#29 ☐	#30 ☐	#31 ☐	#32 ☐

DPR BG Buffers DPRAM Background Automatic Data Reporting Buffers Setup includes enable/disable checkbox along with monitor period in servo cycles and a combo box showing number of coordinate systems to updata data in the DPRAM.



PMAC Device number 0

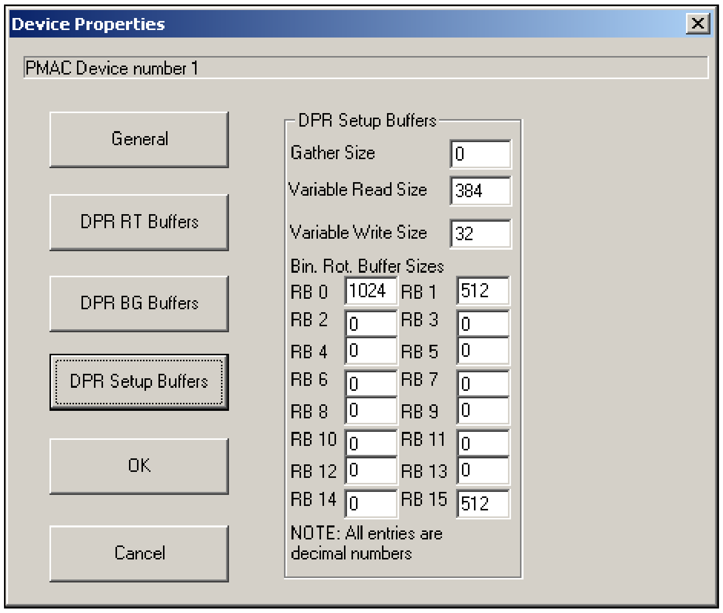
☑ DPRAM BackGround Update

20 BG update rate (servo cycles)

Select Number of Coordinate

1

DPR Setup Buffers provide setup screen for Binary Rotary buffers.



PMAC Device number 1

DPR Setup Buffers

Gather Size: 0
Variable Read Size: 384
Variable Write Size: 32

Bin. Rot. Buffer Sizes

RB 0	1024	RB 1	512
RB 2	0	RB 3	0
RB 4	0	RB 5	0
RB 6	0	RB 7	0
RB 8	0	RB 9	0
RB 10	0	RB 11	0
RB 12	0	RB 13	0
RB 14	0	RB 15	512

NOTE: All entries are decimal numbers

Notes

1. Once these parameters are setup. It is highly recommended to close all applications and restart to have these changes initialize the PcommServer on startup.
2. Both RealTime and Background updates run on separate threads at the start of application.
3. The above functions are therefore not exported through the interface. Rather the status structures are available at the interface and a VB.NET example project is provided for users' reference.

Using the DPR Real Time Fixed Data Buffers

Data Query functions

Following functions provide details on how to obtain Global, Servo and status through DPRAM.

Global Status

```
 [C++]
 STDMETHOD(GetTurboGlobalStatusEx)(/*[in]*/long dwDevice,DEVGSTURBO
*pStatus,/*[out]*/VARIANT_BOOL *pbSuccess);
 STDMETHOD(GetNTGlobalStatusEx)(/*[in]*/long dwDevice,DEVNTGSTURBO
*pStatus,/*[out]*/VARIANT_BOOL *pbSuccess);
```

Motor Status

```
 [C++]
 STDMETHOD(GetTurboMotorStatusEx)(/*[in]*/long dwDevice,/*[in]*/long
lMotor,DEVMSTURBO *pStatus,/*[out]*/VARIANT_BOOL *pbSuccess);
 STDMETHOD(GetNTMotorStatusEx)(/*[in]*/long dwDevice,/*[in]*/long
lMotor,DEVNTMSTURBO *pStatus,/*[out]*/VARIANT_BOOL *pbSuccess);
```

NOTE : We are still working to correct the prototypes of the DPRAM Realtime Data Reporting functions.

```
???????????????????????????????????????????????
PmacDPRGetCommandedPos()
PmacDPRPosition()
PmacDPRFollowError()
PmacDPRGetVel()
PmacDPRGetMasterPos()
PmacDPRGetCompensationPos()

PmacDPRGetPrevDAC()
PmacDPRGetMoveTime()
```

The code segment below shows the sequence of calls (An excerpt from the provided PMACTEST application) that should be used to assure that reading of the DPR would not occur while PMAC is writing:

Notice the sequence for handshaking:

1. Refresh the data within the DPR via the PmacDPRUpdateRealtime() routine
2. Call the Data Query functions to read the DPR Real Time Data buffer.

```
void PmacDPRRealTime::OnTimer(UINT nIDEvent)
{
    char buf[256];

// DO HANDSHAKING
    m_pDoc->DPRUpdateRealtime();

    int servotimer = m_pDoc->DPRGetServoTimer();
    sprintf(buf,"%d",servotimer);
    m_ServoCounter.SetWindowText(buf);

    double aDouble = m_pDoc->DPRPosition(iCurrentMotor-1,1);
    sprintf(buf,"%11.1lf",aDouble);
    m_ActualPosition.SetWindowText(buf);

    aDouble = m_pDoc->DPRGetCommandedPos(iCurrentMotor-1,1);
    sprintf(buf,"%11.1lf",aDouble);
    m_CommandedPosition.SetWindowText(buf);

    aDouble = m_pDoc->DPRFollowError(iCurrentMotor-1,1);
    sprintf(buf,"%11.1lf",aDouble);
    m_FollowingError.SetWindowText(buf);

    m_pDoc->DPRGetCompensationPos(iCurrentMotor-1,1,&aDouble);
    sprintf(buf,"%11.1lf",aDouble);
    m_CompensationPosition.SetWindowText(buf);

    aDouble = m_pDoc->DPRGetVel(iCurrentMotor-1,1);
    sprintf(buf,"%11.1lf",aDouble);
    m_Velocity.SetWindowText(buf);

    long aLong = m_pDoc->DPRGetPrevDAC(iCurrentMotor-1);
    sprintf(buf,"%ld",aLong);
    m_PrevDAC.SetWindowText(buf);

    aLong = m_pDoc->DPRGetMoveTime(iCurrentMotor-1);
    sprintf(buf,"%ld",aLong);
    m_MoveTime.SetWindowText(buf);
}
```

```
void DPRRealTimeTurbo::OnEnablerealtime()
{
    TCHAR buf[255];
  long mask;
  long on = TRUE;
  char *cp;

    m_NumberMotors.GetWindowText(buf,20);
    mask = strtol(buf,&cp,0);
    PmacDPRRealTimeEx (DeviceNum,mask,50,on);

    //  Begin timer
    m_TimerID = SetTimer(
        1,            // timer identifier
        1,                  // 10-second interval
     (TIMERPROC) NULL);     // no timer callback
}

void DPRRealTimeTurbo::OnDisablerealtime()
{
    long on = FALSE;
    PmacDPRRealTimeEx(DeviceNum,0,0,on);
    KillTimer(m_TimerID);
}

void PmacDPRRealTime::OnTimer(UINT nIDEvent)
{
    DWORD dwDevice = 0;
    Double units = 1;
    Long motor = iCurrentMotor-1;

    m_pDoc->DPRDoRealTimeHandshake();

    int servotimer = PmacDPRGetServoTimer(dwDevice);
    sprintf(buf,"%d",servotimer);
    m_ServoCounter.SetWindowText(buf);

    double aDouble = PmacDPRPosition (dwDevice,motor,units);
    sprintf(buf,"%11.1lf",aDouble);
    m_ActualPosition.SetWindowText(buf);

    aDouble = PmacDPRGetCommandedPos (dwDevice,motor,units);
    sprintf(buf,"%11.1lf",aDouble);
    m_CommandedPosition.SetWindowText(buf);

    aDouble = PmacDPRFollowError (dwDevice, motor,units);
    sprintf(buf,"%11.1lf",aDouble);
    m_FollowingError.SetWindowText(buf);

   aDouble = PmacDPRGetBiasPos (dwDevice, motor,units);
    sprintf(buf,"%11.1lf",aDouble);
    m_BiasPos.SetWindowText(buf);

    aDouble = PmacDPRGetVel (dwDevice,motor,3072.0);
    sprintf(buf,"%11.1lf",aDouble);
    m_Velocity.SetWindowText(buf);

    long aLong = PmacDPRGetPrevDAC (dwDevice,motor);
    sprintf(buf,"%ld",aLong);
    m_PrevDAC.SetWindowText(buf);

    CDialog::OnTimer(nIDEvent);
}
```

Using the DPR Background Fixed Data Buffer

Data Query Functions

Coordinate System Status

```
 [C++]
 STDMETHOD(GetTurboCoordinateStatusEx)(/*[in]*/long dwDevice,/*[in]*/long
dwCoord,DEVCSTURBO *pStatus,/*[out]*/VARIANT_BOOL *pbSuccess);
 STDMETHOD(GetNTCoordinateStatusEx)(/*[in]*/long dwDevice,/*[in]*/long
dwCoord,DEVNTCSTURBO *pStatus,/*[out]*/VARIANT_BOOL *pbSuccess);
```

Notes:

1. All of the above structures are listed in detail at the end of this manual.
2. A detailed VB.NET example project "PmacDeviceStatau" is provided for users' reference.
3. These status functions provide complete status structures whether or not you have DPRAM. If DPRAM is not installed then the ASCII port is used to fetch the correct status.

NOTE : We are still working to correct the prototypes of the DPRAM Background Data Reporting functions.

Motor Specific

`PmacDPRCommanded()`

`PmacDPRGetTargetPos()`

Coordinate System Specific

`PmacDPRGetFeedRateMode()`

`PmacDPRProgRemaining()`

`PmacDPRTimeRemInMove()`

`PmacDPRTimeRemInTATS()`

Logical Query Functions

`MOTIONMODE PmacDPRGetMotionMode()`

`PROGRAM PmacDPRGetProgramMode()`

To enable this feature, call the PmacDPRBackgroundEx() routine with the **on** parameter set to a non-zero value, the **period** argument set anywhere from 1 to 255 (servo periods) and the **crd** parameter set from 1 – 8. To retrieve the information, use the provided DPR Background Fixed Data buffer query functions. Information for motors/coordinate systems 1- **crd** will be updated.

When enabled, all the data query functions above may be used.

Typical usage of these routines is shown below and also is available in the supplied example source code (PmacTest application).

```
void BackgroundFixedTurbo::OnDisablebgbuf()
{
    PmacDPRBackgroundEx (dwDevice,FALSE,0,0);
    KillTimer(m_TimerID);
    m_bg_buf_enabled = FALSE;
}

void BackgroundFixedTurbo::OnEnablebgbuf()
{
    TCHAR buf[255];

    m_text_I59.GetWindowText(buf,255);
    USHORT temp = atoi(buf);

    if(!PmacDPRBackgroundEx (dwDevice, TRUE,100,temp))
        ::MessageBox(m_hWnd,"Unable to Initialize Background
Buffer","Information",MB_OK|MB_ICONINFORMATION);
    else{
        m_bg_buf_enabled = TRUE;
//  Begin timer
        m_TimerID = SetTimer(
            1,             // timer identifier
            5,                  // 10-second interval
        (TIMERPROC) NULL);     // no timer callback
    }
}

void BackgroundFixedTurbo::OnTimer(UINT nIDEvent)
{
    TCHAR buf[300];
    double my_double;
  BOOL myBool;

    if(m_bg_buf_enabled){

        my_double = PmacDPRCommanded(dwDevice,m_SelectedCS,'A');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedA.SetWindowText(buf);

        my_double= PmacDPRCommanded(dwDevice,m_SelectedCS,'B');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedB.SetWindowText(buf);

        my_double= PmacDPRCommanded(dwDevice,m_SelectedCS,'C');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedC.SetWindowText(buf);

        my_double=PmacDPRCommanded(dwDevice,m_SelectedCS,'U');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedU.SetWindowText(buf);

        my_double=PmacDPRCommanded(dwDevice,m_SelectedCS,'V');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedV.SetWindowText(buf);

        my_double=PmacDPRCommanded(dwDevice,m_SelectedCS,'W');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedW.SetWindowText(buf);

        my_double=PmacDPRCommanded(dwDevice,m_SelectedCS,'X');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedX.SetWindowText(buf);

        my_double=PmacDPRCommanded(dwDevice,m_SelectedCS,'Y');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedY.SetWindowText(buf);

        my_double=PmacDPRCommanded(dwDevice,m_SelectedCS,'Z');
        sprintf(buf,"%11.4lf",my_double);
        m_textCommandedZ.SetWindowText(buf);
        wsprintf(buf,"%d\r\n", PmacDPRProgRemaining (dwDevice,
m_SelectedCS));
        m_textProgRemaining.SetWindowText(buf);

        wsprintf(buf,"0x%04lx\r\n", PmacDPRPe(dwDevice, m_SelectedCS));
        m_textOffsetAddress.SetWindowText(buf);

        wsprintf(buf,"%7ld\r\n", PmacDPRTimeRemInMove(dwDevice,
m_SelectedCS));
        m_textTIM.SetWindowText(buf);

        wsprintf(buf,"%7ld\r\n", PmacDPRTimeRemInTATS(dwDevice,
m_SelectedCS));
        m_textTATS.SetWindowText(buf);

    my_double= PmacDPRGetFeedRateMode (dwDevice,m_SelectedCS,&myBool);
        sprintf(buf,"%11.4lf", my_double);
        m_FeedRate.SetWindowText(buf);

  }
    CDialog::OnTimer(nIDEvent);
}
```

Using the DPR Binary Rotary Motion Program Buffer

Startup/Shutdown functions

```
// Clear the DPRAM rotary buffer
DPRRotBufClr()
// Turn the Rotary buffer transfer on
DPRSetRotBuf()
```

Conversion & Transfer functions

```
DPRAsciiStrToRotEx()
```

Downloading "Stuffing" functions

PMAC ASCII program command strings either a) Created on the fly or b) Retrieved from a file, are converted to binary and sent to the DPR rotary buffer via the DPRAsciiStrToRotEx() function.

DPR REAL TIME FIXED DATA BUFFER

DPR Real Time Fixed Data Buffer Initialization

```
BOOL PmacDPRRealTimeEx(DWORD dwDevice, long mask, UINT period, int on )
void PmacDPRSetRealTimeMotorMask( DWORD dwDevice, long mask )
BOOL PmacDPRUpdateRealTime( DWORD dwDevice )
```

The PmacDPRRealTimeEx() function enables/disables the DPR real time fixed data buffer. The final parameter **on** will enable this feature if set to 1, otherwise it will disable it. When enabled, all the data query functions above may be used. The **period** parameter in PmacDPRRealTimeEx() specifies how often in servo-cycles PMAC will update the data in this buffer. The **mask** parameter is specifies which of the possible 32 motor data sets to update. Bit 0, the least significant bit, enables or disables the first motor by setting it to 1 or 0. Bit 1 enables or disables the second motor etc… The PmacDPRSetRealTimeMotorMask() routine can also be used to modify which motor data sets are being updated after initialization has been done.

Typical usage of these routines is shown below and also is available in the supplied example source code (PmacTest application).

Arguments

dwDevice	Device number.
period	In units of servo periods (See I10 in PMAC Users Manual)
on_off	Turn on (1) or off (0)
mask	Used to specify what motor data to update in the buffer. Bit 0 = motor 1, Bit 1 = motor 2 etc. 1=report, 0=do not report.

DPR Real Time Fixed Data Buffer Query Routines

Global

```
long  PmacDPRGetServoTimer(DWORD dwDevice);
```

The return value of this routine reflects PMAC's servo counter register located at PMAC address 0.

Motor

The "`motor`" parameter in these functions, is a 0 based index. For example, 0 is motor 1, 1 is motor 2 etc.

```
SERVOSTATUS PmacDPRMotorServoStatus(DWORD dwDevice,long motor);
```

Returns a global status structure, SERVOSTATUS, defined in the section "Data Types, Structures, Callbacks, and Constants".

```
BOOL CALLBACK PmacDPRDataBlock(DWORD dwDevice,long motor);
```

This function returns TRUE when move execution has been aborted because the data for the next move section was not ready in time. This is due to insufficient calculation time. It is FALSE otherwise.

```
BOOL PmacDPRPhasedMotor(DWORD dwDevice,long motor);
```

This function returns TRUE when Ix01 is 1 and this motor is being commutated by PMAC, it is FALSE when Ix01 is 0 and this motor is not being commutated by PMAC.

```
BOOL PmacDPRMotorEnabled(DWORD dwDevice,long motor);
```

This function returns TRUE when Ix00 is 1 and the motor calculations are active. It is 0 when Ix00 is 0 and motor calculations are deactivated.

```
BOOL PmacDPRHandwheelEnabled(DWORD dwDevice,long motor);
```

This function returns TRUE when Ix06 is 1 and position following for this axis is enabled. It returns FALSE when IX06 is 0 and position following is disabled.

```
BOOL PmacDPROpenLoop(DWORD dwDevice,long motor);
```

This function returns TRUE when the servo loop for the motor is open, either with outputs enabled or disabled (killed). It returns FALSE when the servo loop is closed (under position control, always with outputs enabled).

```
BOOL PmacDPROnNegativeLimit(DWORD dwDevice,long motor);
```

This function returns TRUE when motor actual position is less than the software negative position limit (Ix14), or when the hardware limit on this end (+LIMn – note!) has tripped; it is FALSE otherwise.

```
BOOL PmacDPROnPositiveLimit(DWORD dwDevice,long motor);
```

This function returns TRUE when motor actual position is greater than the software positive position limit (Ix13), or when the hardware limit on this end (-LIMn – note!) has tripped; it is FALSE otherwise.

```
void  PmacDPRSetJogReturn(DWORD dwDevice,long motor);
```

This can be used to set the Jog Return Position for the motor specified. The current actual position is used to assign the Jog Return Position.

```
MOTION PmacDPRGetMotorMotion(DWORD dwDevice,long motor);
```

Returns an enumeration based on the motion state of the specified motor.

```
typedef enum { inpos,jog,running,homing,handle,openloop,disabled } MOTION;
double PmacDPRGetCommandedPos(DWORD dwDevice,long motor, double units);
```

This function returns the commanded position of the specified motor. Units are in encoder counts unless the parameter *units* is not one.

```
double PmacDPRPosition(DWORD dwDevice,long i,double units);
```

This function returns the actual position of the specified motor in units of encoder counts provided the *units* variable is unity.

```
double PmacDPRFollowError(DWORD dwDevice,long motor,double units);
```

This function returns the following error of the specified motor in units of encoder counts provided the *units* parameter is unity.

```
double PmacDPRGetVel(DWORD dwDevice,long motor,double units);
```

This function returns the velocity of the specified motor in units of (1/(Ix09*32) counts per servo cycle.

```
void PmacDPRGetMasterPos(DWORD dwDevice,long motor,double units,double *the_double);
```

This function returns the master position of the specified motor in units of encoder counts unless the *units* parameter is unity.

```
void PmacDPRGetCompensationPos(DWORD dwDevice,long motor,double units,double *the_double);
```

This function returns the compensation position of the specified motor in units of encoder counts unless the *units* parameter is unity.

```
DWORD PmacDPRGetPrevDAC(DWORD dwDevice,long motor);
```

This function returns the DAC value of the previous servo cycle. It is in units of 1/256 DAC bits.

DPR BACKGROUND FIXED DATA BUFFER

DPR Background Fixed Data Buffer Initialization and Handshaking

Startup/Shutdown and Handshaking Functions

```
BOOL PmacDPRBackgroundEx(DWORD dwDevice,int on,UINT period,UINT crd);
```

To enable this feature, call the PmacDPRBackgroundEx() routine with the **on** parameter set to a non-zero value, the **period** argument set anywhere from 1 to 255 (servo periods), and the **crd** parameter set from 1 – 8. To retrieve the information, use the provided DPR Background Fixed Data buffer query functions. Information for motors/coordinate systems 1- **crd** will be updated.

When enabled all the data query functions above may be used.

Typical usage of these routines is shown below, and also is available in the supplied example source code (PmacTest application).

DPR Background Fixed Data Buffer Query Routines

The “`crd`” / ”motor” parameter in the functions below is a 0 based index. For example, 0 is coordinate system 1, 1 is coordinate system 2 etc.

Data Query Functions

<u>Motor Specific</u>

```
double PmacDPRCommanded(DWORD dwDevice,long crd,char axchar);
```

This routine returns the commanded position for an axis (described by *axchar*) in coordinate system *crd* in the same units as used to define the axis.

```
double PmacDPRGetTargetPos(DWORD dwDevice,long motor,double posscale);
```

The target position of the specified *motor* is returned by this function. The units are in counts unless *posscale* is not unity.

<u>Coordinate System Specific</u>

```
long PmacDPRPe(DWORD dwDevice,long crd);
```

This function returns the program execution status for a given coordinate system, *crd*. The lower 24 bits are used. The low 24 bits are the same as the second word returned on a "??" command from PMAC.

```
long PmacDPRProgRemaining(DWORD dwDevice,long crd);
```

This function returns the number of program lines remaining for a specified coordinate system, *crd*. Same as the "PR" command for PMAC.

```
long PmacDPRTimeRemInMove(DWORD dwDevice,long crd);
```

This function returns the time remaining in the current move for a specified coordinate system, *crd*. The value this function returns is not very intuitive or useful for other than display purposes. The time for a single line in a program may be divided into three parts: acceleration, steady state, and deceleration times.

```
long PmacDPRTimeRemInTATS(DWORD dwDevice,long crd);
```

This function returns the time remaining in accel/decl when I13>0.

```
double PmacDPRGetFeedRateMode(DWORD dwDevice,int csn, BOOL *mode)

BOOL PmacDPRRotBufFull(DWORD dwDevice,long crd);
BOOL PmacDPRSysInposition(DWORD dwDevice,long crd);
BOOL PmacDPRSysWarnFError(DWORD dwDevice,long crd);
BOOL PmacDPRSysFatalFError(DWORD dwDevice,long crd);
BOOL PmacDPRSysRunTimeError(DWORD dwDevice,long crd);
BOOL PmacDPRSysCircleRadError(DWORD dwDevice,long crd);
BOOL PmacDPRSysAmpFaultError(DWORD dwDevice,long crd);
BOOL PmacDPRProgRunning(DWORD dwDevice,long crd);
BOOL PmacDPRProgStepping(DWORD dwDevice,long crd);
BOOL PmacDPRProgContMotion(DWORD dwDevice,long crd);
BOOL PmacDPRProgContRequest(DWORD dwDevice,long crd);

BOOL PmacDPRMotionBufOpen( DWORD dwDevice)
BOOL PmacDPRRotBufOpen( DWORD dwDevice )
```

The functions above return the boolean status of a variety of coordinate system flags.

Global

```
BOOL PmacDPRSysServoError( DWORD dwDevice )
BOOL PmacDPRSysReEntryError( DWORD dwDevice )
BOOL PmacDPRSysMemChecksumError( DWORD dwDevice )
BOOL PmacDPRSysPromChecksumError( DWORD dwDevice )
void PmacDPRGetGlobalStatus(DWORD dwDevice,VOID *gstatus)
```

Logical Query Functions

```
MOTIONMODE PmacDPRGetMotionMode(DWORD dwDevice,long cs);
```

The motion mode of a chosen coordinate system is returned by this function. The MOTIONMODE enumeration is shown below:

```
typedef enum { linear,rapid,circw,circcw,spline,pvt } MOTIONMODE;
```

```
PROGRAM PmacDPRGetProgramMode(DWORD dwDevice,long cs);
```

The program mode of a chosen coordinate system is returned by this function. The PROGRAMMODE enumeration is shown below:

```
typedef enum { stop,run,step,hold,joghold,jogstop } PROGRAM;
```

DPR BINARY ROTARY BUFFER FUNCTIONS

Note:

The *bufnum* parameter in the routines below is a 0 based index. Therefore, 0 would specify buffer/coordinate system 1, 1 would be buffer/coordinate system 2 etc.

DPRRotBufClr() Method

This function will clear Binary Rotary buffers number in DPR (i.e. remove all entries).

```
[Visual Basic]
Overridable Public Sub DPRRotBufClr( _
   ByVal dwDevice As Integer, _
   ByVal bufnum As Integer _
) Implements !PmacDevice.DPRRotBufClr
```

```
[C#]
virtual public void DPRRotBufClr(
   int dwDevice,
   int bufnum
);
```

```
[C++]
STDMETHOD(DPRRotBufClr)(/*[in]*/long dwDevice,/*[in]*/long bufnum );
```

Arguments

dwDevice	Device number.
bufnum	Which of the two rotary buffers to reference.

DPRSetRotBuf() Method

```
[Visual Basic]
Sub DPRSetRotBuf( _
   ByVal dwDevice As Integer, _
   ByVal on As Boolean _
)
```

```
[C#]
void DPRSetRotBuf(
   int dwDevice,
   bool on
);
```

```
[C++]
STDMETHOD(DPRSetRotBuf)(/*[in]*/long dwDevice,/*[in]*/VARIANT_BOOL on);
```

Once initialized the DPSetRotBuf() function can be used to enable or disable the rotary buffer (if **onoff** = 1 then enable if 0 then disable).

Internally, this routine sets I57 to the appropriate value, and also issues an "Open Rot" for non-Turbo PMACs or "Open Bin Rot" for Turbo PMACs.

Arguments

dwDevice	Device number.
on	Boolean value, Use 1 (ON) or 0 (OFF).

DPRAsciiStrToRotEx() Method

```
[Visual Basic]
Sub DPRAsciiStrToRotEx( _
    ByVal dwDevice As Integer, _
    ByVal inpstr As String, _
    ByVal bufnum As Integer, _
    ByVal bSendRemaining As Boolean, _
    ByRef pstatus As Integer _
)
```

```
[C#]
void DPRAsciiStrToRotEx(
    int dwDevice,
    string inpstr,
    int bufnum,
    bool bSendRemaining,
    out int pstatus
);
```

```
 [C++]
 STDMETHOD(DPRAsciiStrToRotEx)(/*[in]*/long dwDevice,/*[in]*/BSTR
inpstr,/*[in]*/long bufnum,/*[in]*/VARIANT_BOOL bSendRemaining,/*[out]*/long
*pStatus);
```

DPRAsciiStrToRotEx() takes an ASCII Native PMAC text string, converts it to Native PMAC Binary, then places it into the DPR Binary Rotary Buffer if it has been set up and there is room.

Arguments

dwDevice	Device number.
inpstr	NULL terminated PMAC command string.
bufnum	Binary rotary buffer number.
bSendImmediately	BOOL flag meant to send the data in one sweep. Use of this flag is only available for USB mode of communication for now and will be implemented in Ethernet mode of communication soon.
pStatus	Pointer to status. See following table for possible status values.

Mnemonic	Returned Value	Explanation
IDS_ERR_059	-59	"RS274 to BIN DPROT Unable to allocate memory"
IDS_ERR_060	-60	"RS274 to BIN DPROT Unable to pack floating point

		number"
IDS_ERR_061	-61	"RS274 to BIN DPROT Unable to convert string to float number"
IDS_ERR_062	-62	"RS274 to BIN DPROT Illegal Command or Format in string"
IDS_ERR_063	-63	"RS274 to BIN DPROT Integer number out of range"
DprOk	0	The code was successfully sent to DPR
DprBufBsy	1	DPR Binary Rotary Buffer is Busy, please try again soon. Also, PMAC may stop running the program for a variety of reasons. When this occurs, the DPR Rotary Buffer will fill up and appear busy to the PC.
DprEOF	2	DPR Binary Rotary Buffer End of File detected

If you get something other than a DprBufBsy, DprOk, or DprEOF, I'd flag the user of the error. In this case the error is a conversion issue (converting to ASCII to BINARY).

Please see the VC++ example project BinRotLoad for detailed instructions and actual use of the above methods.

DPR NUMERIC READ AND WRITE

General Information

The DPRSet{DataType}() functions write numerical data to the specified **offset** while the DPRGet{DataType}() reads at the specified **offset** and returns the data.

From within PMAC, data can be written to the DPR by use of PMAC M-variable assignments. Proper M-variable definitions for the corresponding data type are shown below:

Data Type	M variable Definition
16 bit integer	M{constant}>X/Y:{Address} (i.e. m100->X:$D200,0,16,s)
32 bit integer	M{constant}->DP:{Address} (i.e. m101->DP:$D201)
32 bit floating point	M{constant}->F:{Address} (i.e. m102->DP:$D202)

Standard Read/Write

```
DPRGetMem()
DPRSetMem()
DPRGetShort()
DPRSetShort()
DPRGetLong()
DPRSetLong()
DPRGetFloat()
DPRSetFloat()
```

Dual Word Conversion

```
DPRFloat()
DPRGetFixedDouble()
```

The "Dual Word" Conversion function converts data that is placed in DPR by one of it's automatic features. Whenever a long word in PMAC (48 bit) is placed in DPR (Motor 1 actual position register for example) each 24-bit short word (X and Y) is sign extended and placed in a 32-bit word, making it 64 bits of data that need to be converted.

A typical sequence of function calls for retrieving and converting motor 1 actual position, for example, would look like:

The second parameter passed to DPRGetFixedDouble() is the scale factor. Since this position register in units of (32*I108) encoder counts, the inverse of this is used to scale the return value.

DPRGetMem() Method

Copies a block of dual ported RAM memory.

[Visual Basic]

```
Sub DPRGetMem( _
    ByVal dwDevice As Integer, _
    ByVal offset As Integer, _
    ByVal bytes As Integer, _
    ByRef pVal As Object, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void DPRGetMem(
    int dwDevice,
    int offset,
    int bytes,
    out object pVal,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(DPRGetMem)(LONG dwDevice, LONG offset, LONG bytes, VARIANT
*pVal,VARIANT_BOOL *pbSuccess);
```

Arguments

dwDevice	Device number.
offset	Offset from the start of dual ported RAM.
bytes	Size of memory block to copy.
pVal	Pointer to destination.
pbSuccess	True if success.

DPRSetMem() Method

Copies a block of memory into dual ported RAM.

```
[Visual Basic]
Sub DPRSetMem( _
    ByVal dwDevice As Integer, _
    ByVal offset As Integer, _
    ByVal bytes As Integer, _
    ByVal Val As Object, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void DPRSetMem(
    int dwDevice,
    int offset,
    int bytes,
    object Val,
    out bool pbSuccess
);
```

```
 [C++]
 STDMETHOD(DPRSetMem)(LONG dwDevice, LONG offset, LONG bytes, VARIANT
Val,VARIANT_BOOL *pbSuccess);
```

Arguments

dwDevice	Device number.
offset	Offset from the start of dual ported RAM.
bytes	Size of memory block to copy.
val	Pointer to memory to transfer.
pbSuccess	True if success.

DPRGetShort() Method

This method replaces the old DPRGetWord() function

```
[Visual Basic]
Overridable Public Sub DPRGetShort( _
   ByVal dwDevice As Integer, _
   ByVal address As Integer, _
   ByRef pVal As Short, _
   ByRef pbSuccess As Boolean _
) Implements !PmacDevice.DPRGetShort
```

```
[C#]
virtual public void DPRGetShort(
   int dwDevice,
   int address,
   out short pVal,
   out bool pbSuccess
);
```

```
 [C++]
 STDMETHOD(DPRGetShort)(LONG dwDevice, LONG address, SHORT* pVal,VARIANT_BOOL
*pbSuccess);
```

Arguments

dwDevice	Device number.
offset	Offset from the start of dual ported RAM.
pVal	Pointer to Short Value to copy.
pbSuccess	True if success.

DPRSetShort() Method

This method replaces the old DPRSetWord() function

```
[Visual Basic]
Overridable Public Sub DPRSetShort( _
   ByVal dwDevice As Integer, _
   ByVal address As Integer, _
```

```
    ByVal newVal As Short, _
    ByRef pbSuccess As Boolean _
) Implements !PmacDevice.DPRSetShort
```

```
[C#]
virtual public void DPRSetShort(
    int dwDevice,
    int address,
    short newVal,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(DPRSetShort)(LONG dwDevice, LONG address, SHORT newVal,VARIANT_BOOL
*pbSuccess);
```

Arguments

dwDevice	Device number.
offset	Offset from the start of dual ported RAM.
newVal	Short Value to transfer.
pbSuccess	True if success.

DPRGetLong() Method

This method replaces the old DPRGetDWord() function

```
[Visual Basic]
Overridable Public Sub DPRGetLong( _
    ByVal dwDevice As Integer, _
    ByVal offset As Integer, _
    ByRef pVal As Integer, _
    ByRef pbSuccess As Boolean _
) Implements !PmacDevice.DPRGetLong
```

```
[C#]
virtual public void DPRGetLong(
    int dwDevice,
    int offset,
    out int pVal,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(DPRGetLong)(/*[in]*/long dwDevice,/*[in]*/long offset,/*[out]*/long
*pVal,VARIANT_BOOL *pbSuccess);
```

Arguments

dwDevice	Device number.
offset	Offset from the start of dual ported RAM.
pVal	Pointer to Long Value to copy.

pbSuccess	True if success.

DPRSetLong() Method

This method replaces the old DPRSetDWord() function

```
[Visual Basic]
Sub DPRSetLong( _
    ByVal dwDevice As Integer, _
    ByVal offset As Integer, _
    ByVal newVal As Integer, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void DPRSetLong(
    int dwDevice,
    int offset,
    int newVal,
    out bool pbSuccess
);
```

```
  [C++]
 STDMETHOD(DPRSetLong)(/*[in]*/long dwDevice,/*[in]*/long offset,/*[in]*/long
newVal,VARIANT_BOOL *pbSuccess);
```

Arguments

dwDevice	Device number.
offset	Offset from the start of dual ported RAM.
newVal	Long Value to transfer.
pbSuccess	True if success.

DPRGetFloat() Method

Reads an IEEE 32-bit floating point value from dual ported RAM.

```
[Visual Basic]
Overridable Public Sub DPRGetFloat( _
    ByVal dwDevice As Integer, _
    ByVal offset As Integer, _
    ByRef pVal As Single, _
    ByRef pbSuccess As Boolean _
) Implements !PmacDevice.DPRGetFloat
```

```
[C#]
virtual public void DPRGetFloat(
    int dwDevice,
    int offset,
    out Single pVal,
    out bool pbSuccess
);
```

```
 [C++]
STDMETHOD(DPRGetFloat)(/*[in]*/long dwDevice,/*[in]*/long offset,/*[out]*/float
*pVal,VARIANT_BOOL *pbSuccess);
```

Arguments

dwDevice	Device number.
offset	Offset from the start of dual ported RAM.
pVal	Pointer to Float at offset location.
pbSuccess	True if success.

DPRSetFloat() Method

Writes a floating point value into dual ported RAM.

```
[Visual Basic]
Overridable Public Sub DPRSetFloat( _
   ByVal dwDevice As Integer, _
   ByVal offset As Integer, _
   ByVal newVal As Single, _
   ByRef pbSuccess As Boolean _
) Implements !PmacDevice.DPRSetFloat
```

```
[C#]
virtual public void DPRSetFloat(
   int dwDevice,
   int offset,
   Single newVal,
   out bool pbSuccess
);
```

```
 [C++]
STDMETHOD(DPRSetFloat)(/*[in]*/long dwDevice,/*[in]*/long offset,/*[in]*/float
newVal,VARIANT_BOOL *pbSuccess);
```

Arguments

dwDevice	Device number.
offset	Offset from the start of dual ported RAM.
newVal	Value to store.
pbSuccess	True if success.

DPRFloat() Method

Converts a 48-bit floating point word: 36-bit mantissa / 12 exponent packed into two 32-bit words holding 24 bits each of the 48-bit words. Multiplies the result by “*scale*.”

```
[Visual Basic]
```

```
Overridable Public Sub DPRFloat( _
    ByVal d As Long, _
    ByVal scale As Double, _
    ByRef pVal As Double _
) Implements !PmacDevice.DPRFloat
```

```
[C#]
virtual public void DPRFloat(
    long d,
    Double scale,
    out Double pVal
);
```

```
[C++]
STDMETHOD(DPRFloat)(/*[in]*/__int64 d,/*[in]*/double scale,/*[out]*/double
*pVal);
```

Arguments

d[]	Two 32 bit long values to converts.
scale	Scale multiplier.
pVal	Pointer to Double floating point representation of the 48-bt number.

DPRGetFixedDouble() Method

This method replaces the old DPRLFixed function

```
[Visual Basic]
Sub DPRGetFixedDouble( _
    ByVal dwDevice As Integer, _
    ByVal address As Integer, _
    ByRef pVal As Double, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
void DPRGetFixedDouble(
    int dwDevice,
    int address,
    out Double pVal,
    out bool pbSuccess
);
```

```
[C++]
STDMETHOD(DPRGetFixedDouble)(LONG dwDevice, LONG address, DOUBLE*
pVal,VARIANT_BOOL *pbSuccess);
```

Arguments

dwDevice	Device number.
address	Offset from the start of dual ported RAM.

pVal	Pointer to Double value.
pbSuccess	True if success.

PCOMMSERVER STRING MANIPULATION FUNCTIONS

Following functions provides different conversions between. Users have the freedom to use their own conversion routines. These functions are given fro user convenience only.

strto32f Method

```
[Visual Basic]
Sub strto32f( _
    ByVal str As String, _
    ByRef pVal As Double _
)
```

```
[C#]
void strto32f(
    string str,
    out Double pVal
);
```

```
[C++]
STDMETHOD(strto32f)(/*[in]*/BSTR str,/*[out]*/double *pVal);
```

strtod32dp Method

```
[Visual Basic]
Sub strtod32dp( _
    ByVal str As String, _
    ByRef pVal As Double _
)
```

```
[C#]
void strtod32dp(
    string str,
    out Double pVal
);
```

```
[C++]
STDMETHOD(strtod32dp)(/*[in]*/BSTR str,/*[out]*/double *pVal);
```

strtod24 Method

```
[Visual Basic]
Sub strtod24( _
    ByVal str As String, _
    ByRef pVal As Double _
)
```

```
[C#]
```

```
void strtod24(
   string str,
   out Double pVal
);
```

```
[C++]
STDMETHOD(strtod24)(/*[in]*/BSTR str,/*[out]*/double *pVal);
```

strtod48f Method

```
      [Visual Basic]
Sub strtod48f( _
   ByVal str As String, _
   ByRef pVal As Double _
)
```

```
[C#]
void strtod48f(
   string str,
   out Double pVal
);
```

```
[C++]
STDMETHOD(strtod48f)(/*[in]*/BSTR str,/*[out]*/double *pVal);
```

strtod48l Method

```
[Visual Basic]
Sub strtod48l( _
   ByVal str As String, _
   ByRef pVal As Double _
)
```

```
[C#]
void strtod48l(
   string str,
   out Double pVal
);
```

```
[C++]
STDMETHOD(strtod48l)(/*[in]*/BSTR str,/*[out]*/double *pVal);
```

DATA TYPES, STRUCTURES, CALLBACKS, AND CONSTANTS

GLOBALSTATUS for TURBOand Non-TURBO

Used in DPR Real Time Buffer query routines

Turbo Global Status Structure

```
typedef struct _GLOBALSTATUSTURBO
{ // Global Status  ??? Must be on BYTE boundaries
  // DWord 1 ( ??? 1st 24/32 bit word )
  USHORT rffu2            : 8;   // 0-7
  USHORT internal1        : 3;   // 8-10
  USHORT buffer_full      : 1;   // 11
  USHORT internal2        : 4;   // 12-16
  USHORT internal3        : 1;
  USHORT plc_buf_open     : 1;   // 17
  USHORT rot_buf_open     : 1;   // 18
  USHORT prog_buf_open    : 1;   // 19
  USHORT bin_rot_buf_open : 1;   // 20
  USHORT rffu3            : 1;
  USHORT vme              : 1;
  USHORT ultralite        : 1;
  USHORT pad2             : 8;
  // DWord 2 ( ??? 2nd 24/32 bit word )
  USHORT card_adrssed     : 1;   // 0
  USHORT all_adrssed      : 1;   // 1
  USHORT rffu1            : 2;
  USHORT ring_error       : 1;   // 4
  USHORT ring_io_error    : 1;   // 5
  USHORT tws_error        : 1;   // 6
  USHORT end_gather       : 1;   // 7
  USHORT rapid_m_flag     : 1;   // 8
  USHORT rti_warning      : 1;   // 9
  USHORT earom_error      : 1;   // 10
  USHORT dpram_error      : 1;   // 11
  USHORT prom_checksum    : 1;   // 12
  USHORT mem_checksum     : 1;   // 13
  USHORT comp_on          : 1;   // 14
  USHORT wdt1             : 1;   // 15
  USHORT wdt2             : 1;   // 16
  USHORT ext_trig_gat     : 1;   // 17
  USHORT prep_trig_gat    : 1;   // 18
  USHORT data_gat_on      : 1;   // 19
  USHORT servo_err        : 1;   // 20
  USHORT servo_active     : 1;   // 21
  USHORT intr_reentry     : 1;   // 22
  USHORT intr_active      : 1;   // 23
  USHORT pad1             : 8;
} GLOBALSTATUSTURBO;
```

Turbo Global Status Macros

```
///////////////// Gobal Status for Turbo ////////////////////////////////
cpp_quote ("#define GST_MAIN_ERROR                     & 0x800000000000")
cpp_quote ("#define GST_RTI_REENTRY_ERROR              & 0x400000000000")
cpp_quote ("#define GST_CPU_TYPE_1                     & 0x200000000000")
cpp_quote ("#define GST_SERVO_ERORR                    & 0x100000000000")
cpp_quote ("#define GST_DATA_GATH_ENABLED              & 0x080000000000")
cpp_quote ("#define GST_RESERVED_X18                   & 0x040000000000")
cpp_quote ("#define GST_GATHER_EXTERNAL_TRIG             & 0x020000000000")
cpp_quote ("#define GST_SMALL_MEM_TURBO_PMAC             & 0x010000000000")
cpp_quote ("#define GST_INTERNAL_15                      & 0x008000000000")
cpp_quote ("#define GST_COMPENSATE_TABLE_ON              & 0x004000000000")
cpp_quote ("#define GST_GENERAL_CHECKSUM_ERR             & 0x002000000000")
cpp_quote ("#define GST_FIRMWARE_CHECKSUM_ERR            & 0x001000000000")
cpp_quote ("#define GST_DPRAM_ERROR                      & 0x000800000000")
cpp_quote ("#define GST_EAROM_ERROR                      & 0x000400000000")
cpp_quote ("#define GST_REAL_TIME_INTERR_WARN            & 0x000200000000")
cpp_quote ("#define GST_ILLEGAL_L_VAR_DEF                & 0x000100000000")
cpp_quote ("#define GST_SERVO_MACRO_IC_CONFIG_ERR        & 0x000080000000")
cpp_quote ("#define GST_TWS_VAR_PARTITY_ERROR            & 0x000040000000")
cpp_quote ("#define GST_MACRO_COMM_ERROR                 & 0x000020000000")
cpp_quote ("#define GST_MACRO_RING_ERROR                 & 0x000010000000")
cpp_quote ("#define GST_NO_PHASE_CLOCK_ERROR             & 0x000008000000")
cpp_quote ("#define GST_RESERVED_X2                      & 0x000004000000")
cpp_quote ("#define GST_ALL_CARDS_ADD_SERIALLY           & 0x000002000000")
cpp_quote ("#define GST_THIS_CARDS_ADD_SERIALLY          & 0x000001000000")
cpp_quote ("#define GST_TURBO_ULTRALITE                  & 0x000000800000")
cpp_quote ("#define GST_TURBO_VME                        & 0x000000400000")
cpp_quote ("#define GST_CPU_TYPE                         & 0x000000200000")
cpp_quote ("#define GST_BINARY_ROTARY_BUFF_OPEN          & 0x000000100000")
cpp_quote ("#define GST_MOTION_BUFFER_OPEN               & 0x000000080000")
cpp_quote ("#define GST_ASCII_ROTARY_BUFFER_OPEN         & 0x000000040000")
cpp_quote ("#define GST_PLC_BUFFER_OPEN                  & 0x000000020000")
cpp_quote ("#define GST_UMAC_TURBO                       & 0x000000010000")
cpp_quote ("#define GST_INTERNAL_Y15                     & 0x000000008000")
cpp_quote ("#define GST_INTERNAL_Y14                     & 0x000000004000")
cpp_quote ("#define GST_RESERVED_Y13                     & 0x000000002000")
cpp_quote ("#define GST_RESERVED_Y12                     & 0x000000001000")
cpp_quote ("#define GST_FIXED_BUFFER_FULL                & 0x000000000800")
cpp_quote ("#define GST_MACRO_RING_TEST_ENABLE           & 0x000000000400")
cpp_quote ("#define GST_RING_ACTIVE                      & 0x000000000200")
cpp_quote ("#define GST_MODBUS_ACTIVE                    & 0x000000000100")
cpp_quote ("#define GST_RESERVED_Y7                      & 0x000000000080")
cpp_quote ("#define GST_RESERVED_Y6                      & 0x000000000040")
cpp_quote ("#define GST_MACRO_RING_RCVD_BREAK_MSG        & 0x000000000020")
cpp_quote ("#define GST_MACRO_RING_BREAK                 & 0x000000000010")
cpp_quote ("#define GST_MACRO_RING_SYN_PACK_FAULT        & 0x000000000008")
cpp_quote ("#define GST_RESERVED_Y2                      & 0x000000000004")
cpp_quote ("#define     GST_RESERVED_Y1                  & 0x000000000002")
cpp_quote ("#define GST_E_STOP                           & 0x000000000001")
```

Non-Turbo Global Status Structure

```
typedef struct _GLOBALSTATUS // non-Turbo
{ // Global Status
  // DWord 1 ( ??? 1st 24/32 bit word )
  USHORT rffu2 : 8;         // 0-7
  USHORT internal1 : 3;     // 8-10
  USHORT buffer_full : 1;
  USHORT internal2 : 3;     // 12-14
  USHORT dpram_response : 1;
  USHORT plc_command : 1;
  USHORT plc_buf_open : 1;
  USHORT rot_buf_open : 1;  // 18
  USHORT prog_buf_open : 1; // 19
  USHORT internal3 : 2;
  USHORT host_comm_mode : 1;
  USHORT internal4 : 1;
  USHORT pad2 : 8;
  // DWord 2 ( ??? 2nd 24/32 bit word )
  USHORT card_adrssed : 1;      // 0
  USHORT all_adrssed : 1;       // 1
  USHORT rfful : 2;
  USHORT ring_error : 1;        // 4
  USHORT ring_io_error : 1;     // 5
  USHORT tws_error : 1;         // 6
  USHORT end_gather : 1;
  USHORT rapid_m_flag : 1;
  USHORT rti_warning : 1;
  USHORT earom_error : 1;
  USHORT dpram_error : 1;
  USHORT prom_checksum : 1;
  USHORT mem_checksum : 1;
  USHORT comp_on : 1;
  USHORT stimulate_on : 1;
  USHORT stimulus_ent : 1;
  USHORT prep_trig_gat : 1;
  USHORT prep_next_serv : 1;
  USHORT data_gat_on : 1;
  USHORT servo_err : 1;
  USHORT servo_active : 1;
  USHORT intr_reentry : 1;
  USHORT intr_active : 1;
  USHORT pad1 : 8;
} GLOBALSTATUS;
```

Non-Turbo Global Status Macros

```
////////////// Gobal Status for Non-Turbo ///////////////////////////
cpp_quote ("#define GSNT_REAL_TIME_INTERR_ACTIVE         & 0x800000000000")
cpp_quote ("#define GSNT_REAL_TIME_INTERR_REENTRY        & 0x400000000000")
cpp_quote ("#define GSNT_SERVO_ACTIVE                    & 0x200000000000")
cpp_quote ("#define GSNT_SERVO_ERROR                     & 0x100000000000")
cpp_quote ("#define GSNT_DATA_GATHER_ENABLED             & 0x080000000000")
cpp_quote ("#define GSNT_GATHER_ON_NEXT_SERVO            & 0x040000000000")
cpp_quote ("#define GSNT_GATHER_ON_EXTERNAL_TRIG         & 0x020000000000")
cpp_quote ("#define GSNT_RESERVED_X16                    & 0x010000000000")
cpp_quote ("#define GSNT_RESERVED_X15                    & 0x008000000000")
cpp_quote ("#define GSNT_COMPENSATE_TABLE_OPEN           & 0x004000000000")
cpp_quote ("#define GSNT_GENERAL_CHECKSUM_ERR            & 0x002000000000")
cpp_quote ("#define GSNT_FIRMWARE_CHECKSUM_ERR           & 0x001000000000")
cpp_quote ("#define GSNT_DPRAM_ERROR                     & 0x000800000000")
cpp_quote ("#define GSNT_EAROM_ERROR                     & 0x000400000000")
cpp_quote ("#define GSNT_INTERNAL_X9                     & 0x000200000000")
cpp_quote ("#define GSNT_INTERNAL_X8                     & 0x000100000000")
cpp_quote ("#define GSNT_INTERNAL_X7                     & 0x000080000000")
cpp_quote ("#define GSNT_TWS_VAR_PARITY_ERROR            & 0x000040000000")
cpp_quote ("#define GSNT_MARCO_AUX_COMM_ERROR            & 0x000020000000")
cpp_quote ("#define GSNT_RESERVED_X4                     & 0x000010000000")
cpp_quote ("#define GSNT_RESERVED_X3                     & 0x000008000000")
cpp_quote ("#define GSNT_RESERVED_X2                     & 0x000004000000")
cpp_quote ("#define GSNT_ALL_CARDS_ADD_SERIALLY          & 0x000002000000")
cpp_quote ("#define GSNT_THIS_CARDS_ADD_SERIALLY         & 0x000001000000")
cpp_quote ("#define GSNT_EXTENDED_READ                   & 0x000000800000")
cpp_quote ("#define GSNT_HOST_PORT_COMM_MODE             & 0x000000400000")
cpp_quote ("#define GSNT_INTERNAL_Y21                    & 0x000000200000")
cpp_quote ("#define GSNT_INTERNAL_Y20                    & 0x000000100000")
cpp_quote ("#define GSNT_MOTION_BUFFER_OPEN              & 0x000000080000")
cpp_quote ("#define GSNT_ROTARY_BUFFER_OPEN              & 0x000000040000")
cpp_quote ("#define GSNT_PLC_BUFFER_OPEN                 & 0x000000020000")
cpp_quote ("#define GSNT_PLC_COMMAND                     & 0x000000010000")
cpp_quote ("#define GSNT_VME_PORT_COMM_MODE              & 0x000000008000")
cpp_quote ("#define GSNT_INTERNAL_Y14                    & 0x000000004000")
cpp_quote ("#define GSNT_INTERNAL_Y13                    & 0x000000002000")
cpp_quote ("#define GSNT_INTERNAL_Y12                    & 0x000000001000")
cpp_quote ("#define GSNT_FIXED_BUFFER_FULL               & 0x000000000800")
cpp_quote ("#define GSNT_INTERNAL_Y10                    & 0x000000000400")
cpp_quote ("#define GSNT_INTERNAL_Y9                     & 0x000000000200")
cpp_quote ("#define GSNT_INTERNAL_Y8                     & 0x000000000100")
cpp_quote ("#define GSNT_RESERVED_Y7                     & 0x000000000080")
cpp_quote ("#define GSNT_RESERVED_Y6                     & 0x000000000040")
cpp_quote ("#define GSNT_RESERVED_Y5                     & 0x000000000020")
cpp_quote ("#define GSNT_RESERVED_Y4                     & 0x000000000010")
cpp_quote ("#define GSNT_RESERVED_Y3                     & 0x000000000008")
cpp_quote ("#define GSNT_RESERVED_Y2                     & 0x000000000004")
cpp_quote ("#define GSNT_RESERVED_Y1                     & 0x000000000002")
cpp_quote ("#define GSNT_RESERVED_Y0                     & 0x000000000001")
```

COORDINATESYSTEMSTATUS for TURBOand Non-TURBO

Turbo CS Status Structure

<u>Motion</u>

typedef enum { inpos,jog,running,homing,handle,openloop,disabled } MOTION;

<u>MOTIONMODE</u>

typedef enum { linear,rapid,circw,circcw,spline,pvt } MOTIONMODE;

```
typedef struct _COORDSTATUSTURBO
{ // Coord Status Turbo
  // word 3 Coordinate status ( ?? 3rd 24 bit word )
  USHORT in_prog_pmatch         : 1; // Bit #0
  USHORT desired_position_limit : 1; // Bit #1
  USHORT program_resume_error   : 1; // Bit #2
  USHORT radius_error           : 1; // Bit #3
  USHORT reserved               : 4; // Bits #(4-7)
  USHORT lhb_direction_request  : 1; // Bit #8
  USHORT lhb_move_request       : 1; // Bit #9
  USHORT lhb_change_request     : 1; // Bit #10
  USHORT lhb_sing_seg_request   : 1; // Bit #11
  USHORT lhb_lasr_move          : 1; // Bit #12
  USHORT lhb_flush              : 1; // Bit #13
  USHORT lhb_recalculate        : 1; // Bit #14
  USHORT lhb_last_segment       : 1; // Bit #15
  USHORT lhb_change             : 1; // Bit #16
  USHORT lhb_stop               : 1; // Bit #17
  USHORT lhb_direction          : 1; // Bit #18
  USHORT lhb_sync_m_var_ovrflow : 1; // Bit #19
  USHORT internal               : 3; // Bit #(20-22) look_ahead_buf_lbck : 1;     // #22
  USHORT look_ahead_buf_wrap    : 1; // bit #23
  USHORT pading                 : 8;

  // word 1 Coordinate status ( ?? 1st 24 bit word )
  USHORT prog_running      : 1;       // bit 0
  USHORT single_step_mode  : 1;       // bit 1
  USHORT cont_motion_mode  : 1;       // bit 2
  USHORT tm_mode           : 1;       // bit 3
  USHORT cont_motion_req   : 1;       // bit 4
  USHORT rad_vect_inc_mode : 1;       // bit 5
  USHORT a_axis_inc        : 1;       // bit 6
  USHORT a_axis_infeed     : 1;       // bit 7
  USHORT b_axis_inc        : 1;       // bit 8
  USHORT b_axis_infeed     : 1;       // bit 9
  USHORT c_axis_inc        : 1;       // bit 10
  USHORT c_axis_infeed     : 1;       // bit 11
  USHORT u_axis_inc        : 1;       // bit 12
  USHORT u_axis_infeed     : 1;       // bit 13
  USHORT v_axis_inc        : 1;       // bit 14
  USHORT v_axis_infeed     : 1;       // bit 15
  USHORT w_axis_inc        : 1;       // bit 16
  USHORT w_axis_infeed     : 1;       // bit 17
  USHORT x_axis_inc        : 1;       // bit 18
  USHORT x_axis_infeed     : 1;       // bit 19
  USHORT y_axis_inc        : 1;       // bit 20
  USHORT y_axis_infeed     : 1;       // bit 21
  USHORT z_axis_inc        : 1;       // bit 22
  USHORT z_axis_infeed     : 1;       // bit 23
  USHORT pad2              : 8;
} COORDSTATUSTURBO;
```

PROGRAM

typedef enum { stop,run,step,hold,joghold,jogstop } PROGRAM;

```
typedef struct _PROGRAMSTATUS
{ // Program Execution Status ( ?? 2nd 24 bit word )
  USHORT cir_spline_move :        1; // #0
  USHORT ccw_move :               1;
  USHORT cc_on :                  1;
  USHORT cc_left :                1;
  USHORT pvt_spline_move :        1;
  USHORT seg_stop_request :       1;
  USHORT seg_accel :              1;
  USHORT seg_move :               1;
  USHORT rapid_move_mode :        1;
  USHORT cc_buffered :            1;
  USHORT cc_stop_request :        1;
  USHORT cc_outside_corner :      1;
  USHORT dwell_buffered :         1;
  USHORT sync_m_func :            1;
  USHORT eob_stop :               1;
  USHORT delayed_calc :           1;
  USHORT rot_buff_full :          1;
  USHORT in_position :            1;
  USHORT warn_ferr :              1;
  USHORT fatal_ferr :             1;
  USHORT amp_fault :              1;
  USHORT circle_rad_err :         1; // #21  [(Internal) Move in stack in Turbo]
  USHORT run_time_err :           1;
  USHORT prog_hold :              1; // #23 Look ahead in TURBO
  USHORT pad : 8;
} PROGRAMSTATUS;
```

Turbo CS Status Macros

```
////////// Coordinate Systems Status for Turbo ////////////////////////
cpp_quote ("#define CST_Z_AXIS_USED_IN_FEEDRATE          & 0x800000000000000000")
cpp_quote ("#define CST_Z_AXIS_INCREMENT_MODE            & 0x400000000000000000")
cpp_quote ("#define CST_Y_AXIS_USED_IN_FEEDRATE          & 0x200000000000000000")
cpp_quote ("#define CST_Y_AXIS_INCREMENT_MODE            & 0x100000000000000000")
cpp_quote ("#define CST_X_AXIS_USED_IN_FEEDRATE          & 0x080000000000000000")
cpp_quote ("#define CST_X_AXIS_INCREMENT_MODE            & 0x040000000000000000")
cpp_quote ("#define CST_W_AXIS_USED_FEEDRATE             & 0x020000000000000000")
cpp_quote ("#define CST_W_AXIS_INCREMENT_MODE            & 0x010000000000000000")
cpp_quote ("#define CST_V_AXIS_USED_IN_FEEDRATE          & 0x008000000000000000")
cpp_quote ("#define CST_V_AXIS_INCREMENT_MODE            & 0x004000000000000000")
cpp_quote ("#define CST_U_AXIS_USED_FEEDRATE             & 0x002000000000000000")
cpp_quote ("#define CST_U_AXIS_INCREMENT_MODE            & 0x001000000000000000")
cpp_quote ("#define CST_C_AXIS_USED_IN_FEEDRATE          & 0x000800000000000000")
cpp_quote ("#define CST_C_AXIS_INCREMENT_MODE            & 0x000400000000000000")
cpp_quote ("#define CST_B_AXIS_USED_FEEDRATE             & 0x000200000000000000")
cpp_quote ("#define CST_B_AXIS_INCREMENT_MODE            & 0x000100000000000000")
cpp_quote ("#define CST_A_AXIS_USED_IN_FEEDRATE          & 0x000080000000000000")
cpp_quote ("#define CST_A_AXIS_INCREMENT_MODE            & 0x000040000000000000")
cpp_quote ("#define CST_RADIUS_VEC_INCR_MODE             & 0x000020000000000000")
cpp_quote ("#define CST_CONTINUOUS_MOTION_REG            & 0x000010000000000000")
cpp_quote ("#define CST_MOVE_SPEC_BY_TIME                & 0x000008000000000000")
cpp_quote ("#define CST_CONTINUOUS_MOTION_MODE           & 0x000004000000000000")
cpp_quote ("#define CST_SINGLE_STEP_MODE                 & 0x000002000000000000")
cpp_quote ("#define CST_RUNNING_PROGRAM                  & 0x000001000000000000")
cpp_quote ("#define CST_LOOKAHEAD_IN_PROGRES             & 0x000000800000000000")
cpp_quote ("#define CST_RUN_TIME_ERROR                   & 0x000000400000000000")
cpp_quote ("#define CST_INTERNAL_MOVE_IN_STACK           & 0x000000200000000000")
cpp_quote ("#define CST_AMP_FAULT_ERROR                  & 0x000000100000000000")
cpp_quote ("#define CST_FATAL_FOLLOWING_ERROR            & 0x000000080000000000")
cpp_quote ("#define CST_WARNING_FOLLOWING_ERROR          & 0x000000040000000000")
cpp_quote ("#define CST_IN_POSITION                      & 0x000000020000000000")
cpp_quote ("#define CST_ROTARY_BUFFER_FULL               & 0x000000010000000000")
cpp_quote ("#define CST_DELAYED_CALC_FLAG                & 0x000000008000000000")
cpp_quote ("#define CST_END_OF_BLOCK_STOP_PROGS          & 0x000000004000000000")
cpp_quote ("#define CST_INTNAL_SYNC_M_VAR_ONE_SHOT       & 0x000000002000000000")
cpp_quote ("#define CST_INTNAL_DWELL_MOVE_BUFFERED       & 0x000000001000000000")
cpp_quote ("#define CST_CTTR_COMP_OUTSIDE_CORNER         & 0x000000000800000000")
cpp_quote ("#define CST_CTTR_COMP_STOP_REQ               & 0x000000000400000000")
cpp_quote ("#define CST_CTTR_COMP_MOVE_BUFFERED          & 0x000000000200000000")
cpp_quote ("#define CST_PRE_JOG_MOVE_IN_PROGS            & 0x000000000100000000")
cpp_quote ("#define CST_SEGMENTED_MOVE_IN_PROGS          & 0x000000000080000000")
cpp_quote ("#define CST_SEGMENT_ACCEL_FLAG               & 0x000000000040000000")
cpp_quote ("#define CST_SEGMENT_STOP_REQUEST             & 0x000000000020000000")
cpp_quote ("#define CST_PVT_SPLINE_MOVE_MODE             & 0x000000000010000000")
cpp_quote ("#define CST_CUTTER_COMP_LEFT                 & 0x000000000008000000")
cpp_quote ("#define CST_CUTTER_COMP_ON                   & 0x000000000004000000")
cpp_quote ("#define CST_CCW_CIRCLE_RAPID_MOVE_MODE       & 0x000000000002000000")
cpp_quote ("#define CST_CIRCLE_SPLINE_MOVE_MODE          & 0x000000000001000000")
cpp_quote ("#define CST_LHB_WRAP                         & 0x000000000000800000")
cpp_quote ("#define CST_INTERNAL_Y22                     & 0x000000000000400000")
cpp_quote ("#define CST_INTERNAL_Y21                     & 0x000000000000200000")
cpp_quote ("#define CST_INTERNAL_Y20                     & 0x000000000000100000")
cpp_quote ("#define CST_LHB_SYNC_M_VAR_OVERFLOW          & 0x000000000000080000")
cpp_quote ("#define CST_LHB_DIRECTION                    & 0x000000000000040000")
cpp_quote ("#define CST_LHB_STOP                         & 0x000000000000020000")
cpp_quote ("#define CST_LHB_CHANGE                       & 0x000000000000010000")
cpp_quote ("#define CST_LHB_LAST_SEGMENT                 & 0x000000000000008000")
cpp_quote ("#define CST_LHB_RECALCULATE                  & 0x000000000000004000")
cpp_quote ("#define CST_LHB_FLUSH                        & 0x000000000000002000")
cpp_quote ("#define CST_LHB_LAST_MOVE                    & 0x000000000000001000")
cpp_quote ("#define CST_LHB_SINGLE_SEGMENT_REGUEST       & 0x000000000000000800")
cpp_quote ("#define CST_LHB_CHANGE_REQUEST               & 0x000000000000000400")
cpp_quote ("#define CST_LHB_MOVE_REQUEST                 & 0x000000000000000200")
cpp_quote ("#define CST_LHB_DIRECTION_REQUEST            & 0x000000000000000100")
cpp_quote ("#define CST_RESERVED_Y7                      & 0x000000000000000080")
cpp_quote ("#define CST_RESERVED_Y6                      & 0x000000000000000040")
cpp_quote ("#define CST_RESERVED_Y5                      & 0x000000000000000020")
cpp_quote ("#define CST_RESERVED_Y4                      & 0x000000000000000010")
cpp_quote ("#define CST_RADIUS_ERROR                     & 0x000000000000000008")
cpp_quote ("#define CST_PROGRAM_RESUME_ERROR             & 0x000000000000000004")
cpp_quote ("#define CST_DESIRED_POSITION_LIMIT_STOP      & 0x000000000000000002")
cpp_quote ("#define CST_IN_PROGRAM_PMATCH                & 0x000000000000000001")
```

Non-Turbo CS Status Structure

Motion

typedef enum { inpos,jog,running,homing,handle,openloop,disabled } MOTION;

```
typedef struct _COORDSTATUS
{ // Coord Status
  // word 1 Motor definition word
  unsigned long motor_def;

  // word 2  Coordinate status ( ?? 1st 24 bit word )
  USHORT prog_running : 1;
  USHORT single_step_mode : 1;
  USHORT cont_motion_mode : 1;
  USHORT tm_mode : 1;
  USHORT cont_motion_req : 1;
  USHORT rad_vect_inc_mode : 1;
  USHORT a_axis_inc : 1;
  USHORT a_axis_infeed : 1;
  USHORT b_axis_inc : 1;
  USHORT b_axis_infeed : 1;
  USHORT c_axis_inc : 1;
  USHORT c_axis_infeed : 1;
  USHORT u_axis_inc : 1;
  USHORT u_axis_infeed : 1;
  USHORT v_axis_inc : 1;
  USHORT v_axis_infeed : 1;
  USHORT w_axis_inc : 1;
  USHORT w_axis_infeed : 1;
  USHORT x_axis_inc : 1;
  USHORT x_axis_infeed : 1;
  USHORT y_axis_inc : 1;
  USHORT y_axis_infeed : 1;
  USHORT z_axis_inc : 1;
  USHORT z_axis_infeed : 1;
  USHORT pad2 : 8;
} COORDSTATUS;
```

<u>PROGRAM</u>

typedef enum { stop,run,step,hold,joghold,jogstop } PROGRAM;

```
typedef struct _PROGRAMSTATUS
{ // Program Execution Status ( ?? 2nd 24 bit word )
  USHORT cir_spline_move :        1; // #0
  USHORT ccw_move :               1;
  USHORT cc_on :                  1;
  USHORT cc_left :                1;
  USHORT pvt_spline_move :        1;
  USHORT seg_stop_request :       1;
  USHORT seg_accel :              1;
  USHORT seg_move :               1;
  USHORT rapid_move_mode :        1;
  USHORT cc_buffered :            1;
  USHORT cc_stop_request :        1;
  USHORT cc_outside_corner :      1;
  USHORT dwell_buffered :         1;
  USHORT sync_m_func :            1;
  USHORT eob_stop :               1;
  USHORT delayed_calc :           1;
  USHORT rot_buff_full :          1;
  USHORT in_position :            1;
  USHORT warn_ferr :              1;
  USHORT fatal_ferr :             1;
  USHORT amp_fault :              1;
  USHORT circle_rad_err :         1; // #21 [(Internal) Move in stack in Turbo]
  USHORT run_time_err :           1;
  USHORT prog_hold :              1; // #23 Look ahead in TURBO
  USHORT pad : 8;
} PROGRAMSTATUS;
```

Non-Turbo CS Status Macros

```
///////// Coordinate Systems Status for Non-Turbo ////////////////////
cpp_quote ("#define CSNT_AXIS_USED_IN_FEEDRATE           & 0x800000000000")
cpp_quote ("#define CSNT_Z_AXIS_INCREMENT_MODE           & 0x400000000000")
cpp_quote ("#define CSNT_Y_AXIS_USED_IN_FEEDRATE         & 0x200000000000")
cpp_quote ("#define CSNT_Y_AXIS_INCREMENT_MODE           & 0x100000000000")
cpp_quote ("#define CSNT_X_AXIS_USED_IN_FEEDRATE         & 0x080000000000")
cpp_quote ("#define CSNT_X_AXIS_INCREMENT_MODE           & 0x040000000000")
cpp_quote ("#define CSNT_W_AXIS_USED_FEEDRATE            & 0x020000000000")
cpp_quote ("#define CSNT_W_AXIS_INCREMENT_MODE           & 0x010000000000")
cpp_quote ("#define CSNT_V_AXIS_USED_IN_FEEDRATE         & 0x008000000000")
cpp_quote ("#define CSNT_V_AXIS_INCREMENT_MODE           & 0x004000000000")
cpp_quote ("#define CSNT_U_AXIS_USED_FEEDRATE            & 0x002000000000")
cpp_quote ("#define CSNT_U_AXIS_INCREMENT_MODE           & 0x001000000000")
cpp_quote ("#define CSNT_C_AXIS_USED_IN_FEEDRATE         & 0x000800000000")
cpp_quote ("#define CSNT_C_AXIS_INCREMENT_MODE           & 0x000400000000")
cpp_quote ("#define CSNT_B_AXIS_USED_FEEDRATE            & 0x000200000000")
cpp_quote ("#define CSNT_B_AXIS_INCREMENT_MODE           & 0x000100000000")
cpp_quote ("#define CSNT_A_AXIS_USED_IN_FEEDRATE         & 0x000080000000")
cpp_quote ("#define CSNT_A_AXIS_INCREMENT_MODE           & 0x000040000000")
cpp_quote ("#define CSNT_RADIUS_VEC_INCR_MODE            & 0x000020000000")
cpp_quote ("#define CSNT_CONTINUOUS_MOTION_REG           & 0x000010000000")
cpp_quote ("#define CSNT_MOVE_SPEC_BY_TIME               & 0x000008000000")
cpp_quote ("#define CSNT_CONTINUOUS_MOTION               & 0x000004000000")
cpp_quote ("#define CSNT_SINGLE_STEP_MODE                & 0x000002000000")
cpp_quote ("#define CSNT_RUNNING_PROGRAM                 & 0x000001000000")
cpp_quote ("#define CSNT_PROGRAM_HOLD_IN_PROGRESS        & 0x000000800000")
cpp_quote ("#define CSNT_RUN_TIME_ERROR                  & 0x000000400000")
cpp_quote ("#define CSNT_CIRCLE_RADIUS_ERROR             & 0x000000200000")
cpp_quote ("#define CSNT_AMP_FAULT_ERROR                 & 0x000000100000")
cpp_quote ("#define CSNT_FATAL_FOLLOWING_ERROR           & 0x000000080000")
cpp_quote ("#define CSNT_WARNING_FOLLOWING_ERROR         & 0x000000040000")
cpp_quote ("#define CSNT_IN_POSITION                     & 0x000000020000")
cpp_quote ("#define CSNT_ROTARY_BUFFER_FULL              & 0x000000010000")
cpp_quote ("#define CSNT_DELAYED_CALC_FLAG               & 0x000000008000")
cpp_quote ("#define CSNT_END_OF_BLOCK_STOP_PROGS         & 0x000000004000")
cpp_quote ("#define CSNT_SYNC_M_VAR_ONE_SHOT             & 0x000000002000")
cpp_quote ("#define CSNT_DWELL_MOVE_BUFFERED             & 0x000000001000")
cpp_quote ("#define CSNT_CTTR_COMP_OUTSIDE_CORNER        & 0x000000000800")
cpp_quote ("#define CSNT_CTTR_COMP_STOP_REQ              & 0x000000000400")
cpp_quote ("#define CSNT_CTTR_COMP_MOVE_BUFFERED         & 0x000000000200")
cpp_quote ("#define CSNT_PRE_JOG_MOVE_IN_PROGS           & 0x000000000100")
cpp_quote ("#define CSNT_SEGMENTED_MOVE                  & 0x000000000080")
cpp_quote ("#define CSNT_SEGMENT_ACCEL_FLAG              & 0x000000000040")
cpp_quote ("#define CSNT_SEGMENT_STOP_REQUEST            & 0x000000000020")
cpp_quote ("#define CSNT_PVT_SPLINE_MODE                 & 0x000000000010")
cpp_quote ("#define CSNT_CUTTER_COMP_LEFT                & 0x000000000008")
cpp_quote ("#define CSNT_CUTTER_COMP_ON                  & 0x000000000004")
cpp_quote ("#define CSNT_CCW_CIRCLE_RAPID_MOVE_MODE      & 0x000000000002")
cpp_quote ("#define CSNT_CIRCLE_SPLINE_MOVE_MODE         & 0x000000000001")
```

SERVOSTATUS TURBO and Non-Turbo

Used in DPR Real Time Buffer query routines

Turbo Motor Status Structure

```
////////////////////Real Time (foreground) TURBO /////////////////////
```

```
typedef struct _SERVOSTATUSTURBO
{ // Motor Servo Status ( ?  1st 24 bit word  )
  USHORT rapid_spd_sel : 1;  // B00 - RAPID MOVE SPEED SELECT (IXX90)
  USHORT dac_sign_mag : 1;   // B01 - SIGN/MAGNITUDE SERVO (IXX96)
  USHORT sw_capture : 1;     // B02 - SOFTWARE HOME CAPTURE (IXX97.0)
  USHORT fe_capture : 1;     // B03 - CAPTURE ON FOLLOWING ERROR (IXX97.1)
  USHORT handwheel_ena : 1;  // B04 - HANDWHEEL ENABLE FLAG (IXX06.0)
  USHORT hw_mode : 1;        // B05 - HANDWHEEL MODE FLAG (IXX06.1)
  USHORT phased_motor : 1;   // B06 - PHASED MOTOR ENABLE FLAG (IXX01.0)
  USHORT yenc_phase : 1;     // B07 - Y PHASE ENCODER (IXX01.1)
  USHORT user_servo : 1;     // B08 - USER WRITEN SERVO ENABLE (IXX59.0)
  USHORT user_phase : 1;     // B09 - USER WRITEN PHASE ENABLE (IXX59.1)
  USHORT home_search : 1;    // B10 - HOME IN PROGRESS FLAG
  USHORT block_request : 1;  // B11 - BLOCK REQUEST FLAG
  USHORT limit_stop : 1;     // B12 - Limit Stop Flag
  USHORT desired_vel_0 : 1;  // B13 - Desired Velocity = 0
  USHORT data_block_err : 1; // B14 - DATA BLOCK ERROR
  USHORT dwelling : 1;       // B15 - Dwell Mode
  USHORT integrator_ena : 1; // B16 - Ixx34
  USHORT run_program : 1;    // B17 - MOVE TIMER ACTIVE
  USHORT open_loop : 1;      // B18 - OPEN LOOP MODE
  USHORT amp_enabled : 1;    // B19 - AMPLIFIER ENABLED FLAG
  USHORT algo_ena : 1;       // B20 - EXTENDED ALGO ENABLE FLAG (I3300+50*N)
  USHORT pos_limit : 1;      // B21 - POSITIVE POSITION LIMIT
  USHORT neg_limit : 1;      // B22 - NEGATIVE POSITION LIMIT
  USHORT activated : 1;      // B23 - Ixx00
  USHORT pads : 8;           // B24..31 - Not Available
} SERVOSTATUSTURBO;
typedef struct _MOTORSTATUSTURBO
{ // Motor Status ( ?  2nd 24 bit word  )
  USHORT in_position : 1;      // B00 - IN POSITION
  USHORT warn_ferr : 1;        // B01 - SOFT FOLLOWING ERROR
  USHORT fatal_ferr : 1;       // B02 - FATAL FOLLOWING ERROR
  USHORT amp_fault : 1;        // B03 - AMP FAULT ERROR
  USHORT backlash_dir : 1;     // B04 - BACKLASH DIRECTION FLAG
  USHORT amp_i2t_err : 1;      // B05 - I2T AMP FAULT
  USHORT integral_ferr : 1;    // B06 - INTEGRATED FOLLOWING ERROR FAULT
  USHORT triger_home_flg : 1;  // B07 - TRIGGER/HOME MOVE FLAG
  USHORT phase_find_err : 1;   // B08 - PHASE FINDING ERROR FLAG
  USHORT tbd09 : 1;            // B09 - TBD
  USHORT home_complete : 1;    // B10 - HOME COMPLETE FLAG
  USHORT stopped_on_limit : 1; // B11 - POS LIMIT STOP FLAG
  USHORT: 1;                   // B12 - TBD
  USHORT: 1;                   // B13 - TBD
  USHORT: 1;                   // B14 - TBD
  USHORT cs_assigned : 1;      // B15 - TBD
  USHORT cs_def : 4;           // B16..19 - Coord. Sys. Axis Def
  USHORT coord_sys : 4;        // B20..23 - MOTOR COORDINATE SYSTEM NUMBER (-1)
  USHORT padm : 8;             // B24..31 - Not Available
} MOTORSTATUSTURBO;
```

Turbo Motor Status Macros

```
cpp_quote ("#define MST_MOTOR_ACTIVE                  & 0x800000000000")
cpp_quote ("#define MST_NEG_END_LIMIT_SET             & 0x400000000000")
cpp_quote ("#define MST_POS_END_LIMT_SET              & 0x200000000000")
cpp_quote ("#define MST_EXT_SERVO_ALGO_ENA            & 0x100000000000")
cpp_quote ("#define MST_AMPLIFIER_ENABLE              & 0x080000000000")
cpp_quote ("#define MST_OPEN_LOOP_MODE                & 0x040000000000")
cpp_quote ("#define MST_MOVE_TIME_ACTIVE              & 0x020000000000")
cpp_quote ("#define MST_INTEGRATE_MODE                & 0x010000000000")
cpp_quote ("#define MST_DWELL_IN_PROGRESS             & 0x008000000000")
cpp_quote ("#define MST_DATA_BLOCK_ERROR              & 0x004000000000")
cpp_quote ("#define MST_DESIRED_VELOCITY_0                & 0x002000000000")
cpp_quote ("#define MST_ABORT_DECELERATE_PROGS            & 0x001000000000")
cpp_quote ("#define MST_BLOCK_REQUEST                     & 0x000800000000")
cpp_quote ("#define MST_HOME_SEARCH_PROGS                 & 0x000400000000")
cpp_quote ("#define MST_USER_WRITTEN_PHASE                & 0x000200000000")
cpp_quote ("#define MST_USER_WRITTEN_SERVO                & 0x000100000000")
cpp_quote ("#define MST_Y_ADDRS_COMMUTE                   & 0x000080000000")
cpp_quote ("#define MST_COMMUTATION_ENABLE                & 0x000040000000")
cpp_quote ("#define MST_POS_FOLLOW_OFFSET_MODE            & 0x000020000000")
cpp_quote ("#define MST_POS_FOLLOW_ENABLE                 & 0x000010000000")
cpp_quote ("#define MST_CAPTURE_ERROR_ENABLE              & 0x000008000000")
cpp_quote ("#define MST_SOFTWARE_CAPT_ENABLE              & 0x000004000000")
cpp_quote ("#define MST_SIGN_MAGNITUDE_SERVO              & 0x000002000000")
cpp_quote ("#define MST_RAPID_MAX_VELOCITY                & 0x000001000000")
cpp_quote ("#define MST_CS_1_BIT_3                        & 0x000000800000")
cpp_quote ("#define MST_CS_1_BIT_2                        & 0x000000400000")
cpp_quote ("#define MST_CS_1_BIT_1                        & 0x000000200000")
cpp_quote ("#define MST_CS_1_BIT_0                        & 0x000000100000")
cpp_quote ("#define MST_CS_AXIS_DEF_BIT_3                 & 0x000000080000")
cpp_quote ("#define MST_CS_AXIS_DEF_BIT_2                 & 0x000000040000")
cpp_quote ("#define MST_CS_AXIS_DEF_BIT_1                 & 0x000000020000")
cpp_quote ("#define MST_CS_AXIS_DEF_BIT_0                 & 0x000000010000")
cpp_quote ("#define MST_ASSIGNED_TO_CS                    & 0x000000008000")
cpp_quote ("#define MST_RESERVER_FOR_FUTURE               & 0x000000004000")
cpp_quote ("#define MST_FOREGROUND_IN_POSITION            & 0x000000002000")
cpp_quote ("#define MST_DESIRED_POSITION_STOP             & 0x000000001000")
cpp_quote ("#define MST_STOP_ON_POSITION_LIMIT            & 0x000000000800")
cpp_quote ("#define MST_HOME_COMPLETE                     & 0x000000000400")
cpp_quote ("#define MST_MOTOR_PHASE_REQUEST               & 0x000000000200")
cpp_quote ("#define MST_PHASING_SEARCH_ERROR              & 0x000000000100")
cpp_quote ("#define MST_TIGGER_MOVE                       & 0x000000000080")
cpp_quote ("#define MST_INTEG_FATAL_FOLLOW_ERR            & 0x000000000040")
cpp_quote ("#define MST_I2T_AMP_FAULT_ERROR               & 0x000000000020")
cpp_quote ("#define MST_BACKLASH_DIRECTION_FLAG           & 0x000000000010")
cpp_quote ("#define MST_AMP_FAULT_ERROR                   & 0x000000000008")
cpp_quote ("#define MST_FATAL_ERROR_EXCEEDED              & 0x000000000004")
cpp_quote ("#define MST_WARNING_ERROR_EXCEEDED            & 0x000000000002")
cpp_quote ("#define MST_IN_POSITION_TRUE                  & 0x000000000001")
```

Non-Turbo Motor Status Structure

```
///////////////// Real Time (foreground) Non-Turbo PMAC ///////////////
typedef struct _SERVOSTATUS
{ // Motor Servo Status ( ?  1st 24 bit word  )
  USHORT internal1 : 8;
  USHORT internal2 : 2;
  USHORT home_search : 1;
  USHORT block_request : 1;
  USHORT rfful : 1;
  USHORT desired_vel_0 : 1;
  USHORT data_block_err : 1;
  USHORT dwelling : 1;
  USHORT integration : 1;
  USHORT run_program : 1;
  USHORT open_loop : 1;
  USHORT phased_motor : 1;
  USHORT handwheel_ena : 1;
  USHORT pos_limit : 1;
  USHORT neg_limit : 1;
  USHORT activated : 1;
  USHORT pad : 8;
} SERVOSTATUS;
```

```
typedef struct _MOTORSTATUS
{ // Motor definition word ( ? 2nd 24 bit word )
  USHORT in_position : 1;
  USHORT warn_ferr : 1;
  USHORT fatal_ferr : 1;
  USHORT amp_fault : 1;
  USHORT backlash_dir : 1;
  USHORT amp_i2t_err : 1;
  USHORT integral_ferr : 1;
  USHORT triger_home_flg : 1;
  USHORT phase_find_err : 1;
  USHORT rffu2 : 1;
  USHORT home_complete : 1;
  USHORT stopped_on_limit : 1;
  USHORT rffu3 : 2;
  USHORT amp_enabled : 1;
  USHORT rffu4 : 1;
  USHORT rffu5 : 4;
  USHORT coord_sys : 3;
  USHORT cs_assigned : 1;
  USHORT pad : 8;
} MOTORSTATUS;
```

Non-Turbo Motor Status Macros

```
/////////////// Motor Status for Non-Turbo ///////////////////////////
cpp_quote ("#define MSNT_MOTOR_ACTIVE                    & 0x800000000000")
cpp_quote ("#define MSNT_NEG_END_LIMIT_SET               & 0x400000000000")
cpp_quote ("#define MSNT_POS_END_LIMT_SET                & 0x200000000000")
cpp_quote ("#define MSNT_HANDWHEEL_ENABLE                & 0x100000000000")
cpp_quote ("#define MSNT_PHASED_MOTOR                    & 0x080000000000")
cpp_quote ("#define MSNT_OPEN_LOOP_MODE                  & Ox040000000000")
cpp_quote ("#define MSNT_RUNNING_A_PROGRAM               & 0x020000000000")
cpp_quote ("#define MSNT_INTEGRATE_MODE                  & 0x010000000000")
cpp_quote ("#define MSNT_DWELL_IN_PROGRESS               & 0x008000000000")
cpp_quote ("#define MSNT_DATA_BLOCK_ERROR                & 0x004000000000")
cpp_quote ("#define MSNT_DESIRED_VELOCITY_0              & 0x002000000000")
cpp_quote ("#define MSNT_ABORT_DECELERATION              & 0x001000000000")
cpp_quote ("#define MSNT_BLOCK_REQUEST                   & 0x000800000000")
cpp_quote ("#define MSNT_HOME_SEARCH_ACTIVE              & 0x000400000000")
cpp_quote ("#define MSNT_INTERNAL_X9                     & 0x000200000000")
cpp_quote ("#define MSNT_INTERNAL_X8                     & 0x000100000000")
cpp_quote ("#define MSNT_INTERNAL_X7                     & 0x000080000000")
cpp_quote ("#define MSNT_INTERNAL_X6                     & 0x000040000000")
cpp_quote ("#define MSNT_INTERNAL_X5                     & 0x000020000000")
cpp_quote ("#define MSNT_INTERNAL_X4                     & 0x000010000000")
cpp_quote ("#define MSNT_INTERNAL_X3                     & 0x000008000000")
cpp_quote ("#define MSNT_INTERNAL_X2                     & 0x000004000000")
cpp_quote ("#define MSNT_INTERNAL_X1                     & 0x000002000000")
cpp_quote ("#define MSNT_INTERNAL_X0                     & 0x000001000000")
cpp_quote ("#define MSNT_ASSIGNED_TO_CS                  & 0x000000800000")
cpp_quote ("#define MSNT_CS_1_BIT_2                      & 0x000000400000")
cpp_quote ("#define MSNT_CS_1_BIT_1                      & 0x000000200000")
cpp_quote ("#define MSNT_CS_1_BIT_0                      & 0x000000100000")
cpp_quote ("#define MSNT_RESERVED_Y19                    & 0x000000080000")
cpp_quote ("#define MSNT_RESERVED_Y18                    & 0x000000040000")
cpp_quote ("#define MSNT_RESERVED_Y17                    & 0x000000020000")
cpp_quote ("#define MSNT_RESERVED_Y16                    & 0x000000010000")
cpp_quote ("#define MSNT_RESERVED_Y15                    & 0x000000008000")
cpp_quote ("#define MSNT_AMPLIFIER_ENABLED               & 0x000000004000")
cpp_quote ("#define MSNT_RESERVED_Y13                    & 0x000000002000")
cpp_quote ("#define MSNT_RESERVED_Y12                    & 0x000000001000")
cpp_quote ("#define MSNT_STOP_ON_POSITION_LIMIT          & 0x000000000800")
cpp_quote ("#define MSNT_HOME_COMPLETE                    & 0x000000000400")
cpp_quote ("#define MSNT_RESERVED_Y9                      & 0x000000000200")
cpp_quote ("#define MSNT_PHASING_SEARCH_ERROR             & 0x000000000100")
cpp_quote ("#define MSNT_TIGGER_MOVE                      & 0x000000000080")
cpp_quote ("#define MSNT_INTEG_FATAL_FOLLOW_ERR           & 0x000000000040")
cpp_quote ("#define MSNT_I2T_AMP_FAULT_ERROR              & 0x000000000020")
cpp_quote ("#define MSNT_BACKLASH_DIRECTION_FLAG          & 0x000000000010")
cpp_quote ("#define MSNT_AMP_FAULT_ERROR                  & 0x000000000008")
cpp_quote ("#define MSNT_FATAL_ERROR_EXCEEDED             & 0x000000000004")
cpp_quote ("#define MSNT_WARNING_ERROR_EXCEEDED           & 0x000000000002")
cpp_quote ("#define MSNT_IN_POSITION_TRUE                 & 0x000000000001")
```

COMMUNICATION APPLICATION NOTES

For all communication related questions and common problems refer to Delta Tau’s communication driver installation notes “DT Driver_Install.pdf”.

INDEX