

PcommServer Library of PMAC Functions

License Statement and Limited Warranty

(If you have any questions, contact our Customer Service Department at (818 998-2095)

IMPORTANT: Carefully read all the terms and condition of this agreement before installing this software. Installing this software indicated your acceptance of the terms and conditions contained in this agreement. If you do not agree to the terms and conditions contained in this agreement, promptly return this package, unopened, and all associated documentation to the place of purchase, and your money will be refunded. No refunds will be given for products that have missing or damaged components.

By installing Delta Tau Data Systems Accessory ACC-9PNPRO2, PcommServer (herein referred to as "the SOFTWARE" or "SOFTWARE") the purchasing customer or corporation accepts the following License Agreement.

LICENSE: The purchasing person or corporation has the right to use the SOFTWARE on an unlimited number of computers owned by the person or corporation ("site license"). The purchasing person or corporation has a royalty-free right to distribute only the "run-time modules" with the executable files created in any other vendor product (Language Development Tool) limited as hereinafter set forth in paragraph a through d. Delta Tau Data Systems, Inc. grants you a royalty-free distribution if: (a) you distribute the "run time" modules only in conjunction with the executable files that make use of them as part of your software product; (b) you do not use the Delta Tau Data Systems, Inc. name, logo, or trademark to market your software product; (c) The SOFTWARE end users do not use the "run time" modules or any other SOFTWARE components for development purposes. And, (d) you agree to indemnify, hold harmless, and defend Delta Tau Data Systems, Inc. and its suppliers from and against any and all claims or lawsuits including attorney's fees, that arise or result from the use or distribution of your software product. If any of the conditions set forth in paragraphs a through d are breached, such breach shall constitute an unlawful use of the SOFTWARE, and you shall be prosecuted to the full extent of the law. Furthermore, you shall be liable to Delta Tau Data Systems, Inc. for all damages caused by such a breach and unlawful use of the software, including attorney's fees and costs incurred in any action, lawsuit or claim brought or filed to redress the breach of this agreement. The "run time modules" are those files included in the SOFTWARE package that are required during execution of your software program.

TERM: This license agreement is in effect until terminated. You may at any time terminate this agreement by destroying the software, diskettes, documentation, and all copies thereof. Delta Tau reserves the right to terminate this agreement if you fail to comply with any of the terms and conditions contained herein. Should Delta Tau terminate this agreement because of your failure to comply, you agree to destroy or return to Delta Tau the program and documentation and any copies, in any and all forms, received from Delta Tau or generated in connection with this agreement.

LIMITED WARRANTY: Delta Tau warrants that the diskettes and documentation enclosed within this product will be free from defects in materials and workmanship for a period of ninety days from the date of purchase as evidenced by a copy of your receipt. THE PROGRAM IS PROVIDED "AS-IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF THE MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. This limited warranty gives you specific legal rights; you may have others that vary from state to state. Some states do not allow the exclusion of incidental or consequential damages so some of the above may not apply to you.

The entire and exclusive liability and remedy for breach of the Limited Warranty shall be limited to replacement of defective diskette(s) or documentation and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims, even if Delta Tau has been specifically advised of the possibility of such damages. In no event will Delta Tau's liability for damages to you or any other person ever exceed the actual original price paid, as evidenced by the receipt, for the license to use the software, regardless of any form of the claim. In the event that the original receipt is lost, the suggested list price at the time of purchase will be substituted as the maximum amount for liability for damages.

GOVERNMENT: This license statement shall be construed, interpreted and governed the laws of the State of California. If any provision of this statement is found void or unenforceable, it will not affect the validity of the balance of this statement, which shall remain valid and enforceable according to its terms. If any remedy provided is determined to have failed of its essential purpose, all limitations of liability and exclusions of damages set forth in the Limited Warranty shall remain in full force and effect. This statement may only be modified in writing signed by you and an authorized officer of Delta Tau. Use, duplication, or disclosure by the US Government of the computer software and documentation in this package shall be subject to the restricted rights applicable to commercial computer software. All rights not specifically granted in this statement are reserved by Delta Tau.

Copyright © 2001, 2005, 2007, 2012 Delta Tau Data Systems, Inc. 21314 Lassen St. Chatsworth, CA 91311 818-998-2095. All rights reserved.

Delta Tau, PMAC, and the symbol are registered trademarks of Delta Tau Data Systems, Inc.

Microsoft, Windows 2000 and XP are registered trademarks of the Microsoft Corporation. Borland is a trademark of Borland International, Inc. Windows is a trademark of Microsoft Corporation

Delta Tau Data Systems Inc.

21314 Lassen St.

Chatsworth, CA 91311

(818) 998 2095

Fax: (818) 998 7807

Table of Contents

INTRODUCTION	6
ABOUT PCOMMSERVER PRO2 LIBRARY	6
A GLOBAL VIEW OF THE DRIVER.....	6
SUPPORTED OPERATING SYSTEMS	8
COMMUNICATION MODES.....	8
<i>Plug & play ports.....</i>	8
<i>Non-plug & play ports</i>	8
HARDWARE REQUIREMENTS.....	8
DISTRIBUTION OF PCOMMSERVER PRO2 FILES.....	8
GETTING STARTED.....	10
SETTING UP COMMUNICATIONS WITH PMAC	10
USAGE OF PCOMMSERVER.....	10
<i>Microsoft VB.NET sample code</i>	10
<i>Microsoft C# .NET Example code.....</i>	12
<i>Using PcommServer in MFC using .NET.....</i>	18
<i>Interfacing PCOMMSERVER in Visual Studio 6 (C++) Applications.....</i>	20
<i>Visual Basic 6.0 Example</i>	25
<i>Borland C++</i>	26
<i>Shutting Down Communication</i>	30
PCOMMSERVER FEATURES.....	32
<i>A Guide to Using ASCII Communication Functions.....</i>	32
<i>Common Problems Experienced Using ASCII Communications Functions</i>	32
<i>Thread-Safe ASCII Communications</i>	32
<i>Error Handling - ASCII Communication And Other Functions</i>	32
USING INTERRUPTS	33
DOWNLOADING TO PMAC	34
INTRODUCTION	37
IMPORTANT INFORMATION ABOUT METHOD OF COMMUNICATION BEING USED BY PCOMMSERVER.....	37
NO PARAMETER SPECIFYING MAXCHAR NEEDED IN PCOMMSERVER	37
ROLE CHANGE BETWEEN THE RETURN VALUE AND STATUS WORD.....	37
INITIALIZATION, SHUTDOWN AND CONFIGURATION FUNCTIONS.....	38
SELECTDEVICE() METHOD.....	38
OPEN() METHOD	39
CLOSE() METHOD	39
ASCII COMMUNICATION FUNCTIONS.....	41
GETRESPONSEEX() METHOD	41
GETCONTROLRESPONSEEX() METHOD.....	41
RAWGETRESPONSEEX METHOD.....	42
RAWGETCONTROLRESPONSEEX() METHOD.....	43
GETRESPONSEPROGRESS() METHOD	44
ABORT() METHOD.....	44
GETPMACTYPE() METHOD	45
DEVPMACTYPE ENUMERATION.....	45
GETPMACLOCATION() METHOD	46
DEVLOCATIONTYPE ENUMERATION.....	47
SETCHECKSUMS() METHOD	47
DPRAVAILABLE(), DPRSIZE() PROPERTIES.....	48
GETASCIICOMM(), SETASCIICOMM() METHODS.....	49

DEVASCIIMODE ENUMERATION	49
PUT_USMONITORINGENABLED() AND PUT_USMONITORINGPERIOD() METHODS.....	49
PHASEMOTOR() METHOD	50
PMACRESET() METHOD	51
PMACSAVE() METHOD.....	51
DPRTTEST() METHOD	52
ABORTTEST() METHOD	53
GET_DPRTTESTING() METHOD	53
DOWNLOADING TO PMAC.....	54
DOWNLOAD() METHOD.....	54
DOWNLOADING() METHOD	55
ABORTDOWNLOAD() METHOD	55
UPLOAD PMAC CONFIGURATION.....	57
DEVUPLOAD STRUCTURE	57
UPLOAD() METHOD	57
GETTING PMAC STATUS	59
STATUS QUERY FUNCTIONS	59
GETTURBOMOTORSTATUSEx() AND GETNTMOTORSTATUSEx() METHODS	59
GETTURBOCOORDINATESTATUSEx() AND GETNTCOORDINATESTATUSEx METHODS	60
GETTURBOGLOBALSTATUSEx() AND GETNTGLOBALSTATUSEx() METHODS.....	61
EVENTS SUPPORT IN PCOMMSERVER	63
MESSAGE EVENT	63
PROGRESS EVENT	63
ERROR EVENTS	63
UNSOLICITED REPOSE.....	63
INTERRUPTS	63
DATA GATHERING FUNCTIONS.....	64
GATHER STRUCTURES.....	64
STARTGATHER, STOPGATHER METHODS	64
COLLECTGATHERDATA METHOD	65
GETGATHERSAMPLES METHOD.....	65
GETNUMGATHERSOURCES METHOD.....	66
SETGATHER METHOD	67
GETGATHER METHOD	67
SETQUICKGATHER METHOD	68
SETQUICKGATHERWITHDIRECTCURRENT METHOD	68
CLEARGATHER, CLEARGATHERDATA METHODS	69
INITGATHER METHOD	70
SETCURRENTGATHER.....	70
GETGATHERPOINT METHOD.....	71
PCOMMSERVER DPR FEATURES.....	73
A GLOBAL VIEW OF THE DPR SUPPORT FUNCTIONS	73
<i>Fixed Real Time Data buffer.....</i>	<i>73</i>
<i>Fixed Background Data Buffer.....</i>	<i>73</i>
<i>Binary Rotary Buffer.....</i>	<i>73</i>
<i>Read/Write Functions</i>	<i>74</i>
CONFIGURING DPR REAL TIME FIXED AND BACKGROUND FIXED DATA BUFFERS	75
<i>Startup/ShutDown and Handshaking.....</i>	<i>75</i>

USING THE DPR REAL TIME FIXED DATA BUFFERS.....	78
GETCOMMANDEDPOS()	78
GETNETACTUALPOSITION()	78
GETFOLLOWERROR()	79
GETVELOCITY().....	79
GETMASTERPOS()	80
GETCOMPENSATIONPOS().....	80
USING THE DPR BACKGROUND FIXED DATA BUFFER	82
GETTARGETPOS()	82
GETBIASPOS()	82
GETFEEDRATEANDMODE()	83
GETPROGRAMMODE()	83
DEVPROGRAMMODE enum.....	84
GETPROGRAMMOTIONMODE().....	84
DEVMOTIONMODE enum	84
USING DPR BINARY ROTARY BUFFER FUNCTIONS	85
DPRROTBUFCLR() METHOD	85
DPRSETROTBUF() METHOD	85
DPRASCIISTRTORETEX() METHOD.....	86
DPR NUMERIC READ AND WRITE.....	88
GENERAL INFORMATION	88
STANDARD READ/WRITE.....	88
DUAL WORD CONVERSION	88
DPRGETMEM() METHOD	88
DPRSETMEM() METHOD.....	89
DPRGETSHORT() METHOD.....	90
DPRSETSHORT() METHOD	90
DPRGETLONG() METHOD	91
DPRSETLONG() METHOD.....	92
DPRGETFLOAT() METHOD.....	92
DPRSETFLOAT() METHOD	93
DPRGETFIXEDDOUBLE() METHOD	94
PCOMMSERVER STRING MANIPULATION FUNCTIONS	96
DPRFLOAT() METHOD	96
STRTO32F() METHOD	96
STRTO32DP() METHOD	97
STRTO24() METHOD.....	97
STRTO48F() METHOD.....	97
STRTO48L() METHOD.....	98
DATA TYPES, STRUCTURES, CALLBACKS, AND CONSTANTS.....	99
GLOBALSTATUS FOR TURBOAND NON-TURBO	99
Turbo Global Status Structure	99
Turbo Global Status Macros.....	99
Non-Turbo Global Status Structure	100
Non-Turbo Global Status Macros.....	101
COORDINATESYSTEMSTATUS FOR TURBOAND NON-TURBO	102
Turbo CS Status Structure	102
Turbo CS Status Macros	103
Non-Turbo CS Status Structure.....	104
Non-Turbo CS Status Macros	106

SERVOSTATUS TURBO AND NON-TURBO.....	106
<i>Turbo Motor Status Structure</i>	106
<i>Turbo Motor Status Macros</i>	107
<i>Non-Turbo Motor Status Structure</i>	108
<i>Non-Turbo Motor Status Macros</i>	109
COMMUNICATION APPLICATION NOTES	113
INDEX	114

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.

Notes:

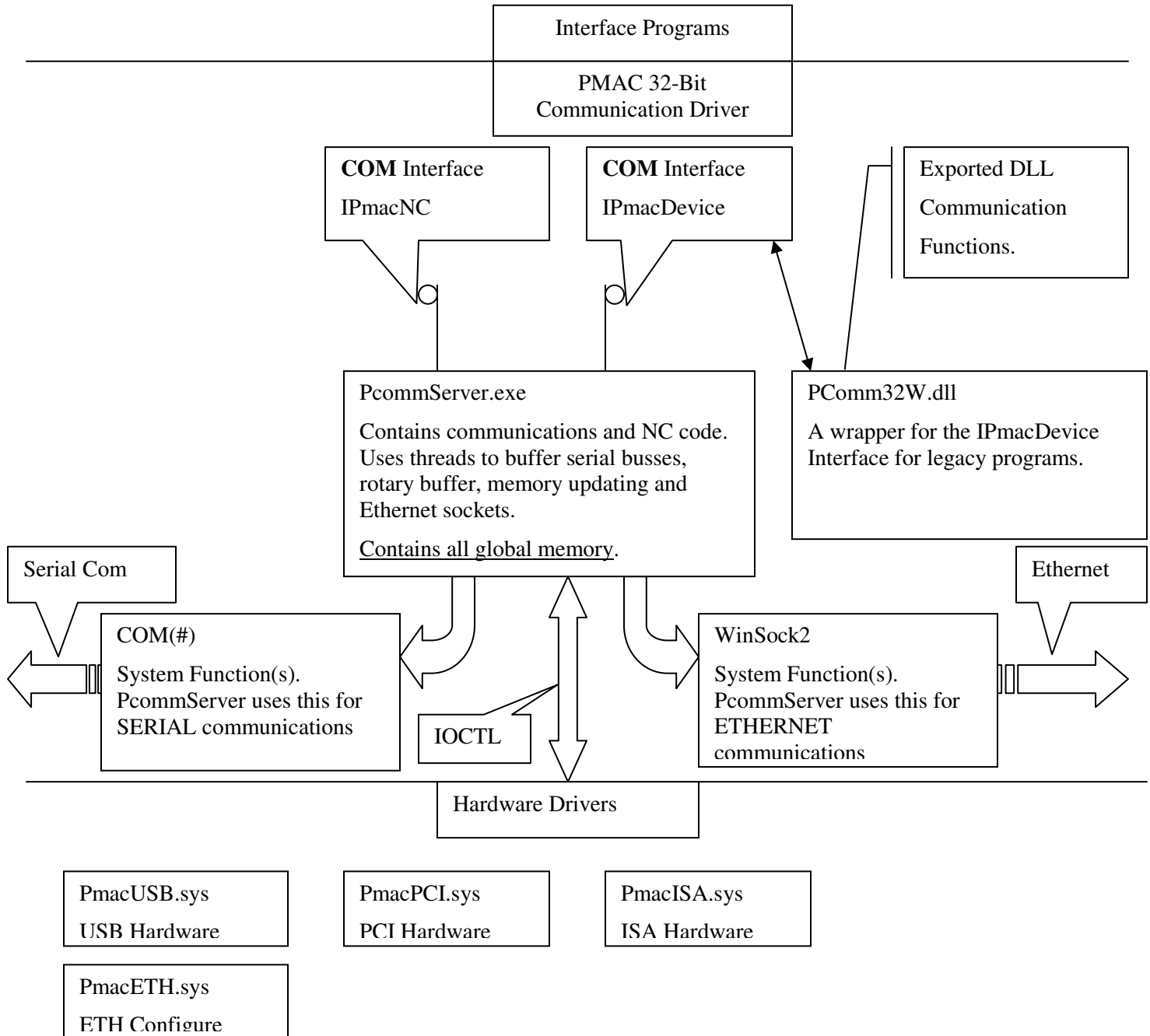
1. PcommServer.exe needs to be registered before use.
2. Developers who want to develop their user interface using PcommServer.exe and subsequently making installations, must register PcommServer.exe before it can be used.
3. PcommServer.exe can be registered using windows' online command or by enabling Inshallshield's self registering property while making installation disk sets.
4. Online command syntax is as follows:

"C:\windows\system32\PcommServer.exe" /regserver

- 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.



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)

Distribution of PcommServer Pro2 files

To distribute an application made using PcommServer Pro2 two steps must be followed.

- A. The first is to copy the needed files to the proper directories. The second is to make sure the key file PcommServer.exe is properly registered.

For both supported operating systems, the key files reside in to following directories:

- PcommServer.exe - \$(WinSysPath)
- PmacISA (**PCI**, **USB**, **USBL**, **USBT**, or **ETH**).sys - \$(WinSysDrvPath)
- PmacISA (**PCI**, or **USB**).inf - \$(WinInfPath)
- ETHUSBConfigure.exe - \$(AppCommonPath)

Where the folders are defined as follows:

Path	Windows 2000	Windows XP
\$(WinPath)	C:\WinNT	C:\Windows
[Windows Root Path]	[default]	[default]
\$(WinInfPath)	\$(WinPath)\Inf	\$(WinPath)\Inf
\$(WinSysPath)	\$(WinPath)\System32	\$(WinPath)\System32

\$(WinSysDrvPath)	\$(WinPath)\System32\Drivers	\$(WinPath)\System32\Drivers
\$(AppCommonPath)	Application Installation folder [C:\Program Files\Delta Tau\<Application Name> by default]	

- B. Second is registering the file PcommServer.exe. How this is done will depend on the installation package used. But the proper command line syntax is:

“\$(WinSysPath)\PcommServer.exe” /Regserver

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.

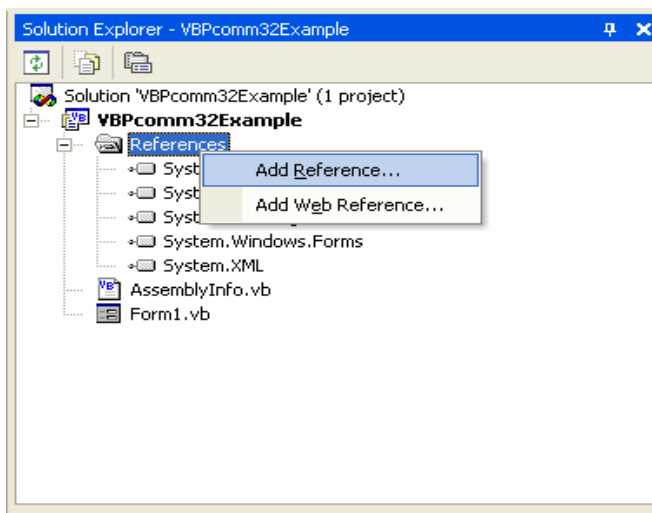
- Microsoft VB.NET sample projects
- Microsoft C#.NET sample projects
- Microsoft C++.NET (Managed C++ code) sample projects
- Microsoft C++ 6.0 sample projects
- Microsoft VB 6.0 sample projects
- Borland C++ Builder 6.0 sample projects

Microsoft VB.NET sample code

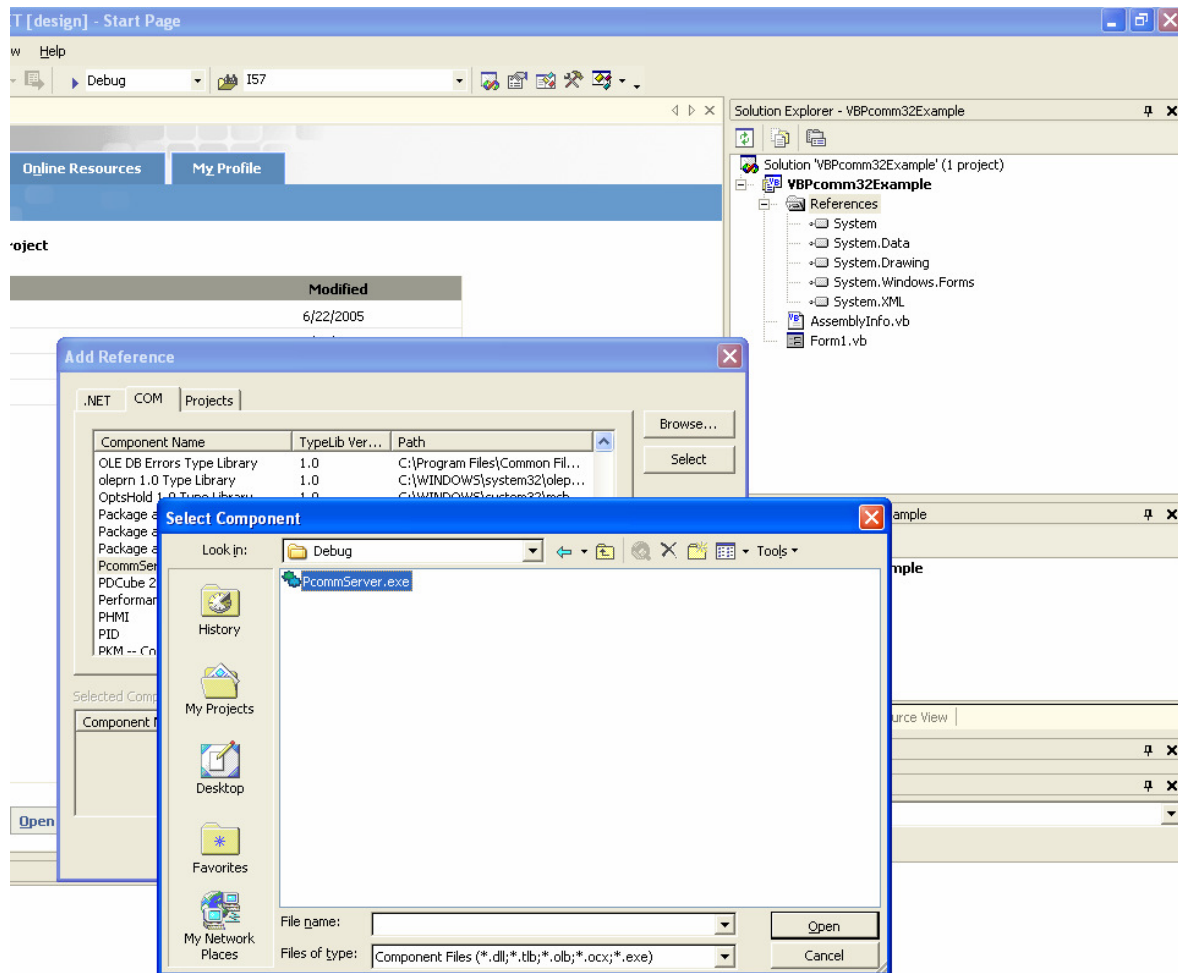
This example contains two steps. First we obtain a reference to PcommServer and then use it our code

Obtaining a reference to PcommServer using VB.NET

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 PCOMMSEVERLib.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

```

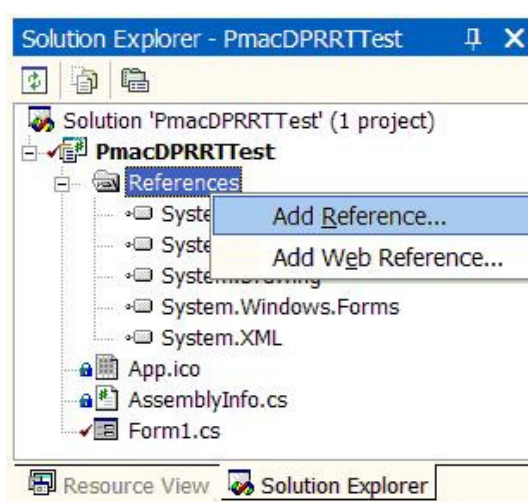
A simple VB.NET example is available for compilation in the folder <PROGRAM FILES>DTDRIVER\DOTNETVB folder.

Microsoft C# .NET Example code

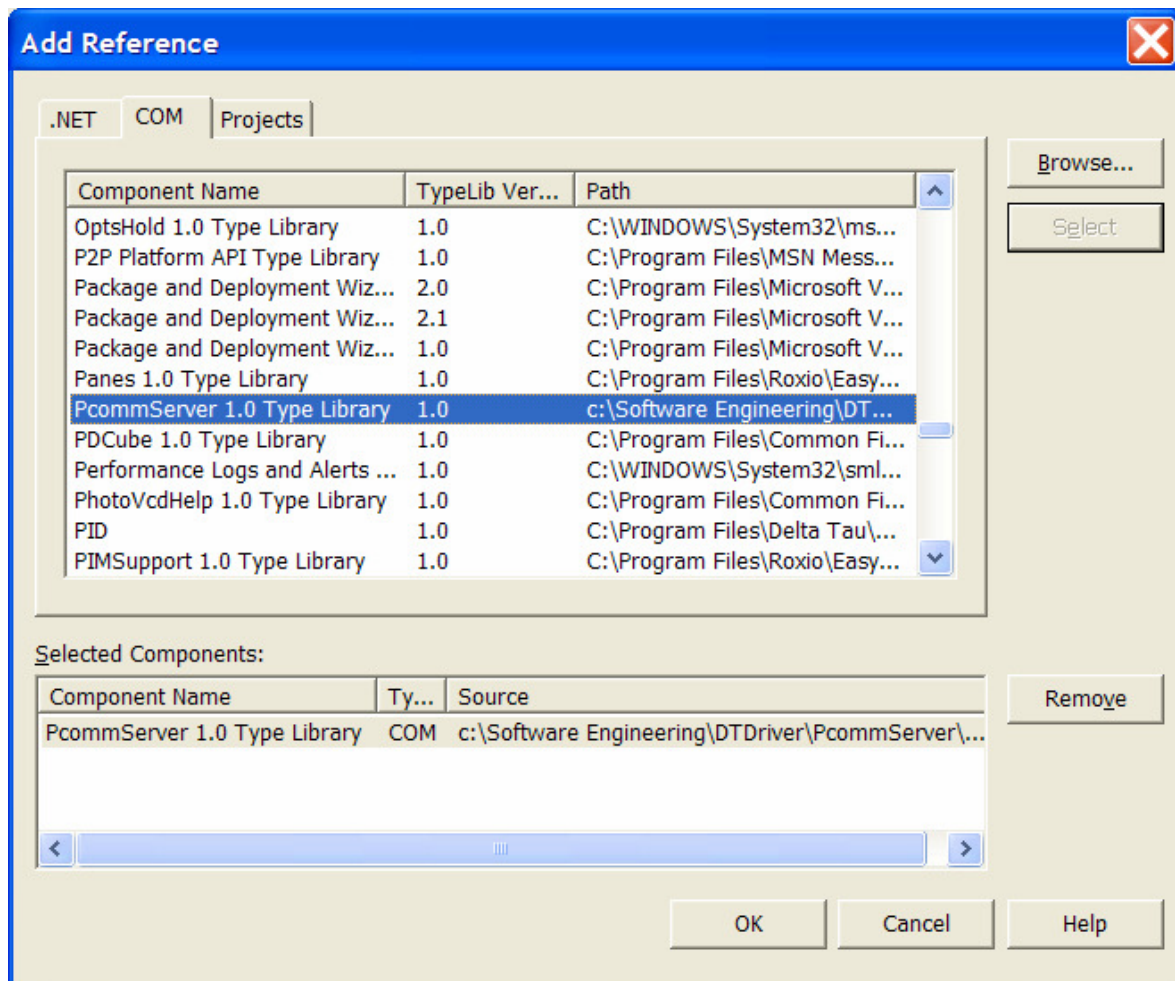
C# has become the environment of choice for many programmers for many applications. In this section we describe the procedure on how to obtain and use the reference to PcommServer for a C#.NET project.

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...**

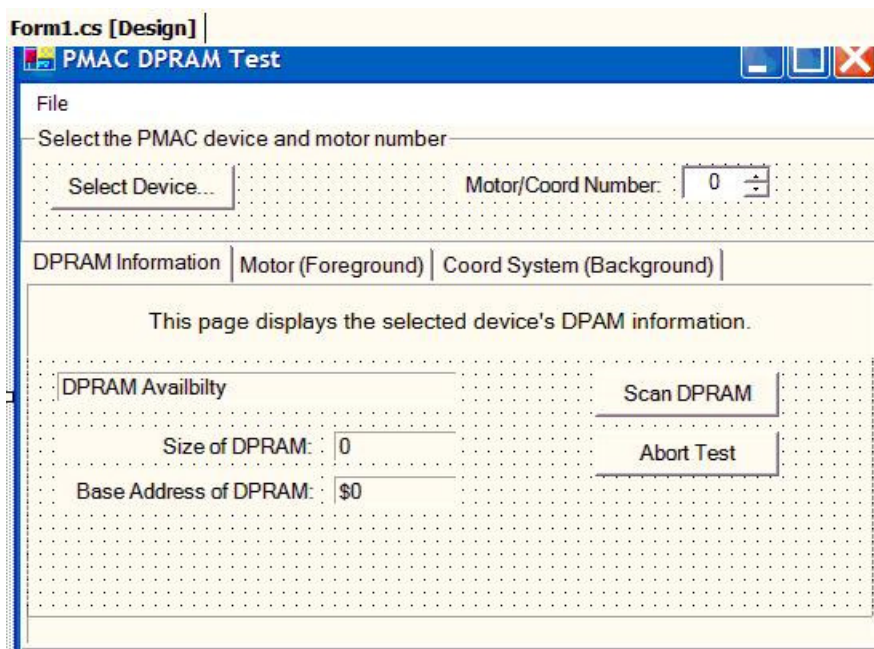


3. The **Add Reference** tabbed Dialog box will appear. Next select the **COM** tab and select the **Browse ...** button and select PcommServer 1.0 Type Library.
4. Now that the PcommServer reference is available it may be used from C#.NET



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.

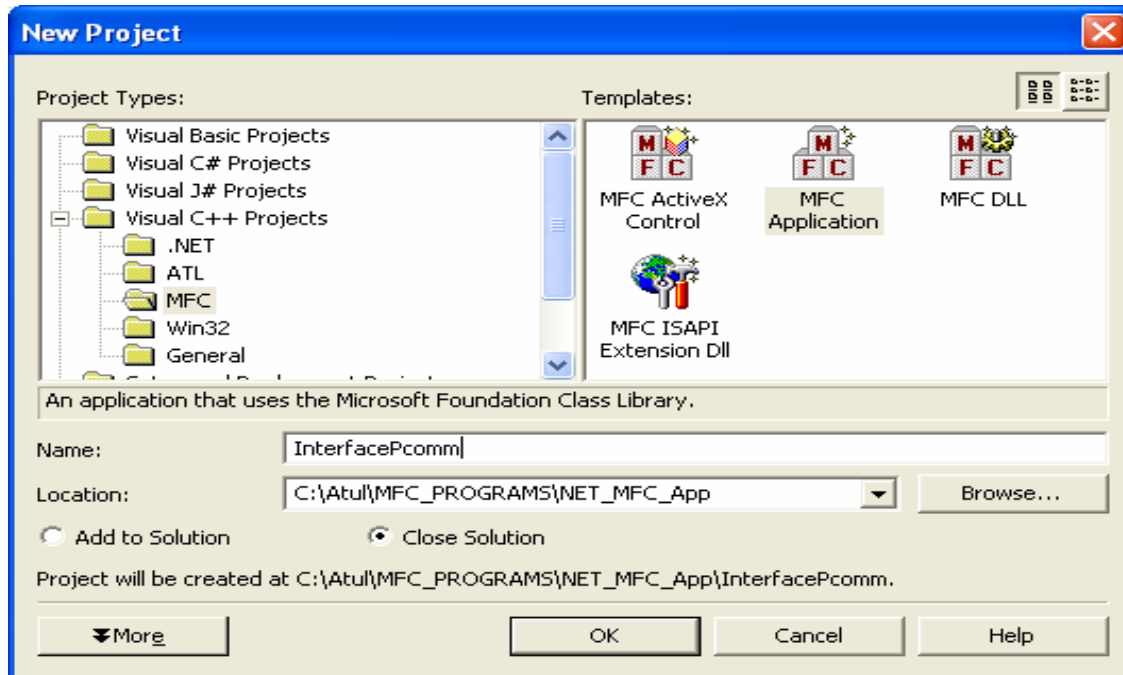
D

```
// 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();
}
```

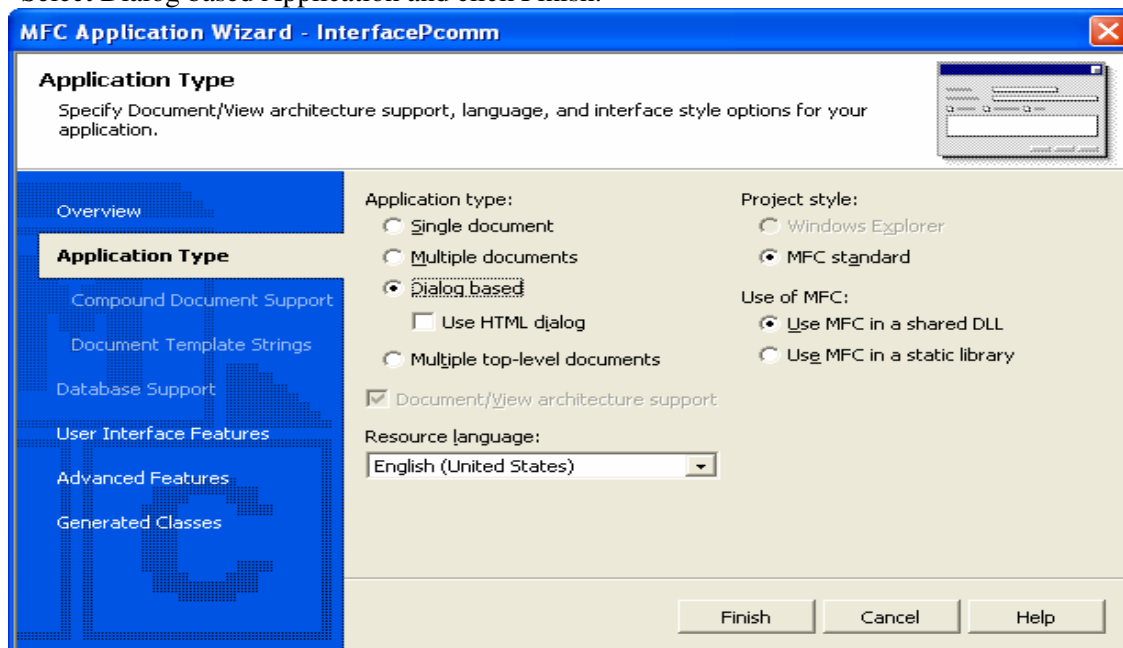
A detailed C#.NET example is available for compilation in the folder <PROGRAM FILES>DTDRIVER\DOTNETC# folder.

Using PcommServer in MFC using .NET

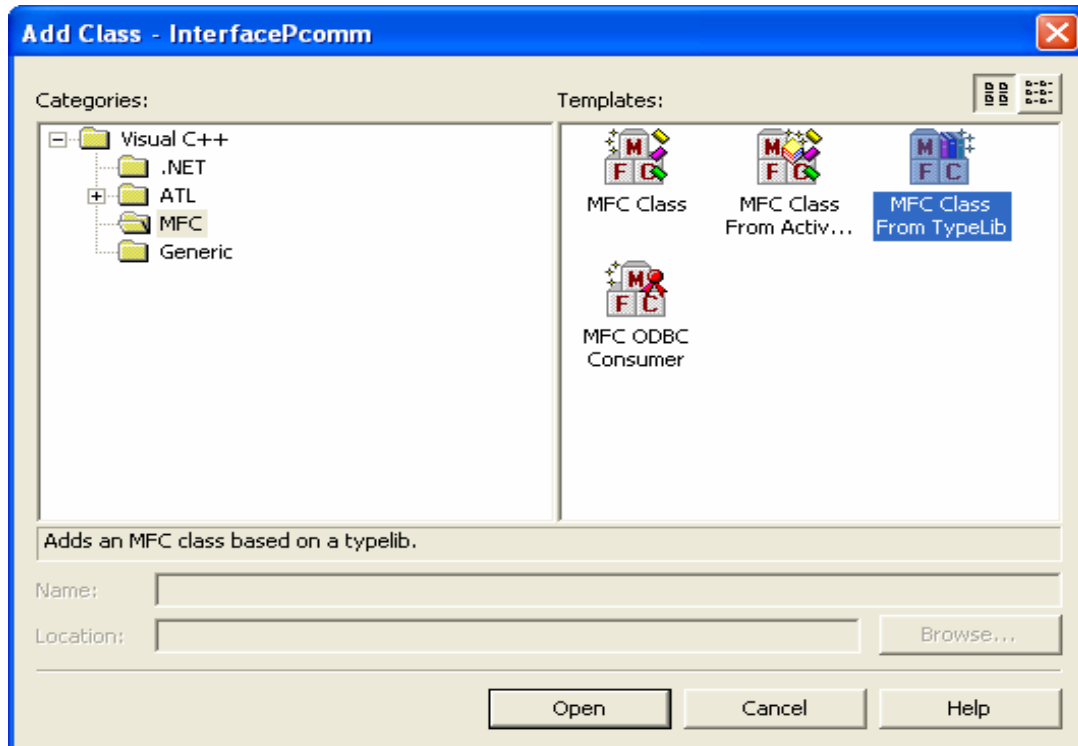
1. Create New MFC project in Microsoft Visual Studio .NET 2003. Name the project as InterfacePcomm.



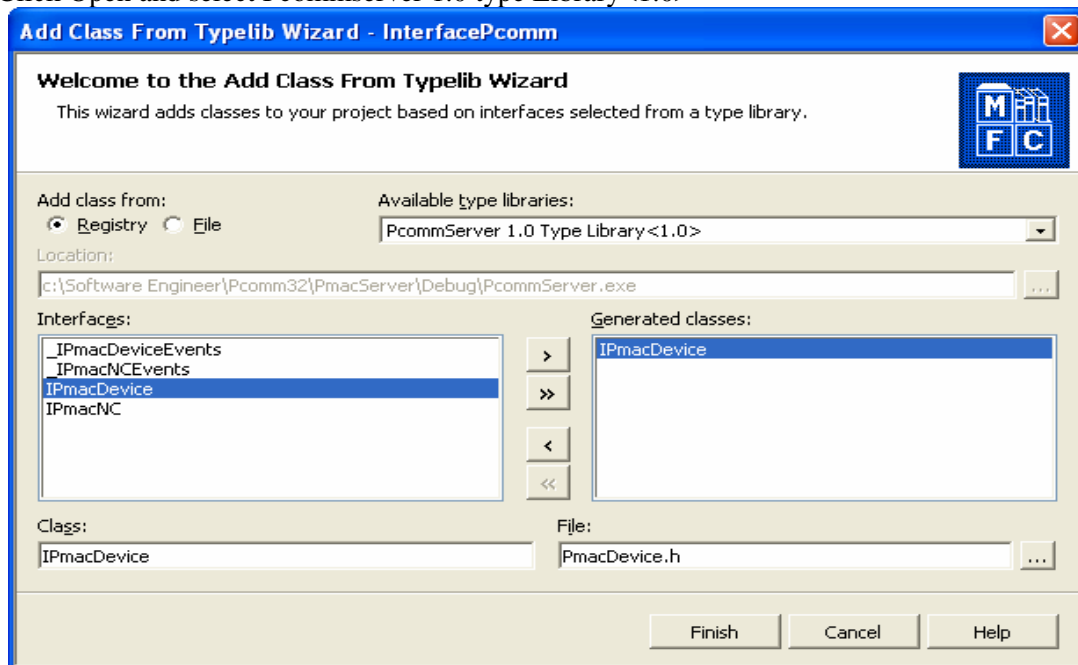
2. Select Dialog based Application and click Finish.



3. Add New Class from type Library.



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



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;
```

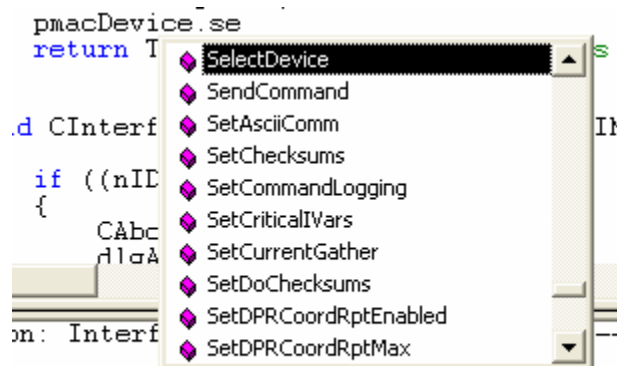
```
CoInitialize(NULL);
```

```
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.



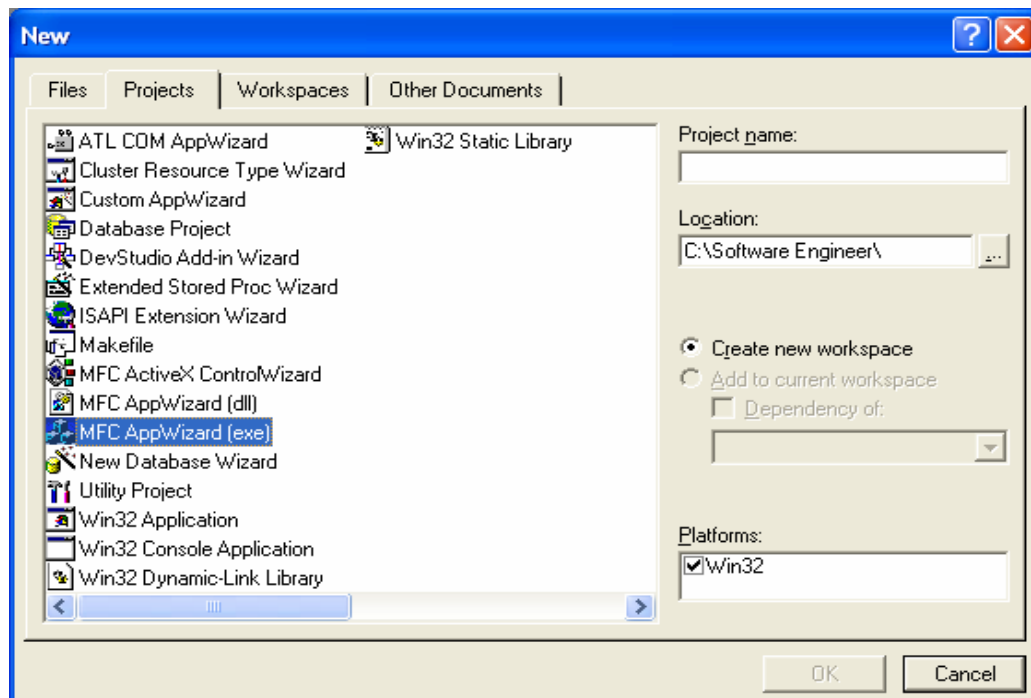
For Example : Add this code to open SelectDevice Dialog box.

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

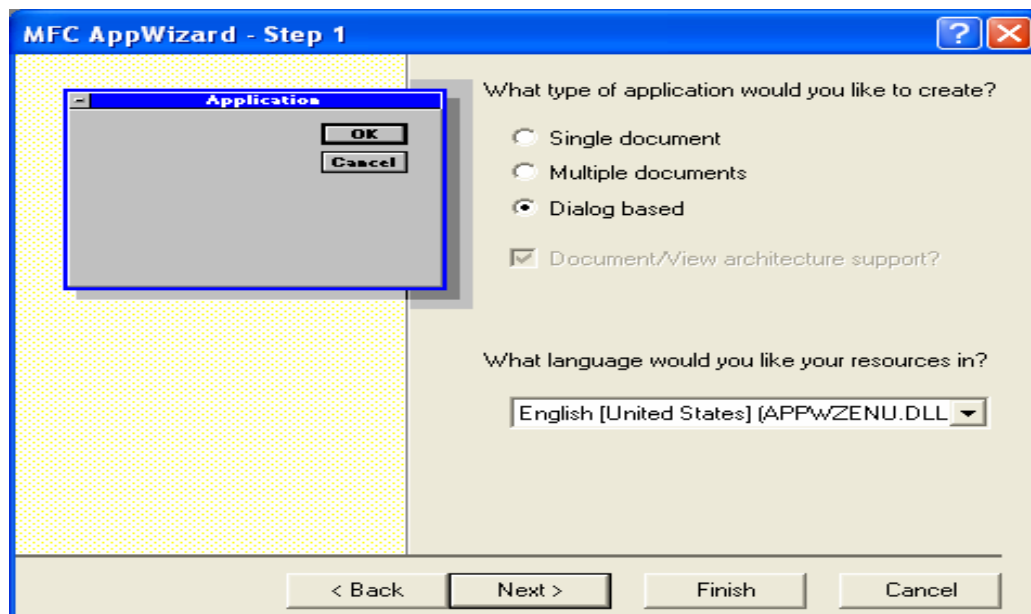
Compile the code and Run! The PmacSelect() dialogbox will appear. Test the PMAC and this completes the PcommServer Interface.

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

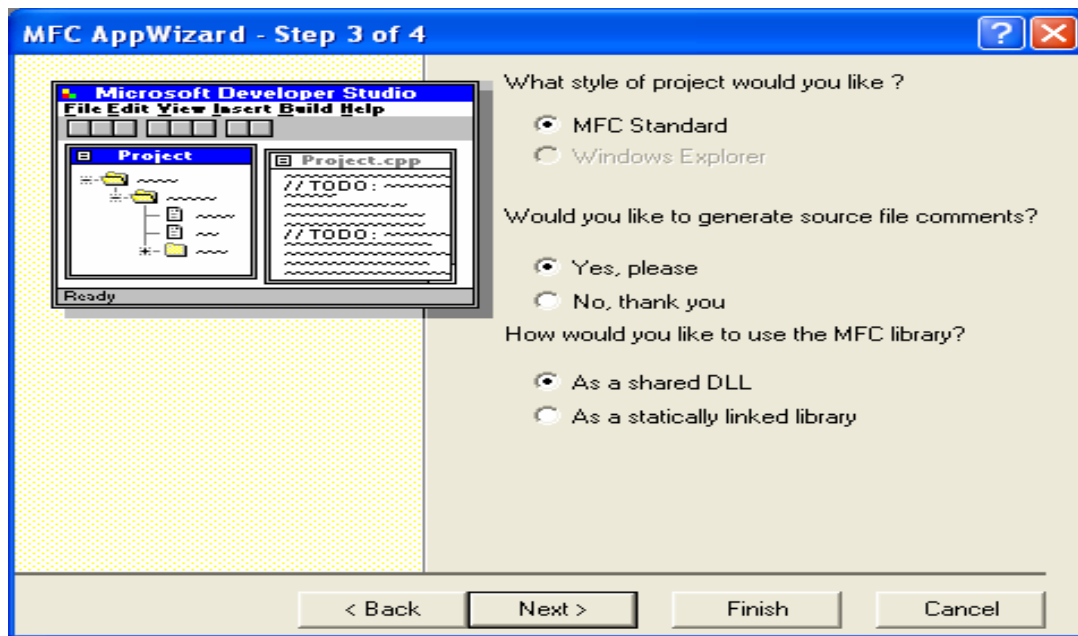
1. Create MFC project using AppWizard. Enter project name and click OK. For our example type InterfacePcomm as project Name.



2. Select Dialog based Application and click Next.

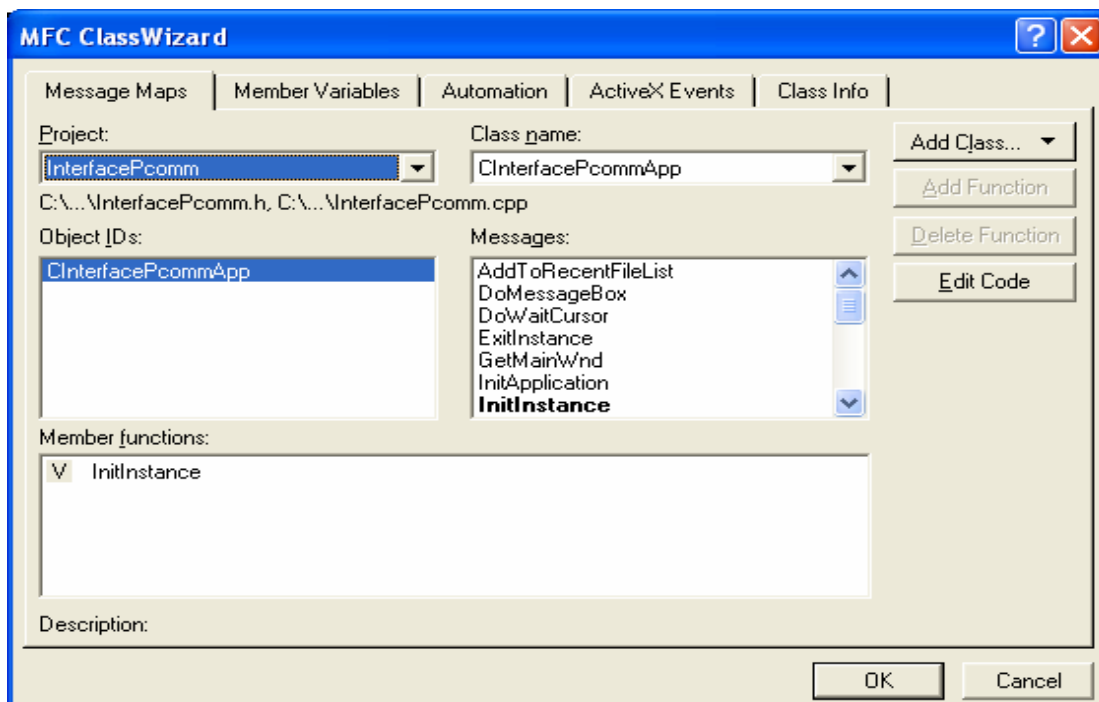


3. Create the project and make sure for MFC support.

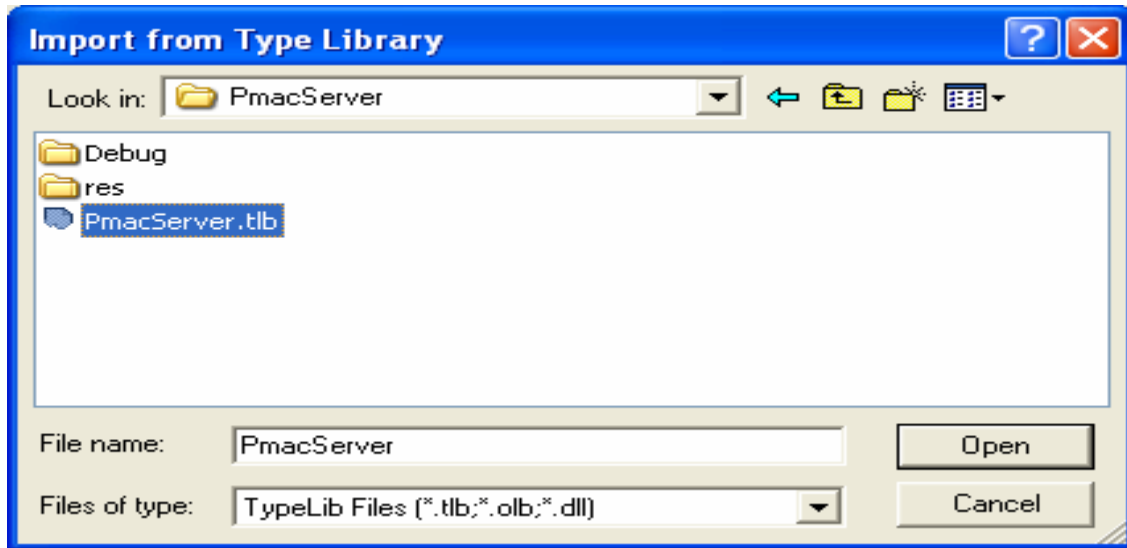


4. Click Finish to generate project template.

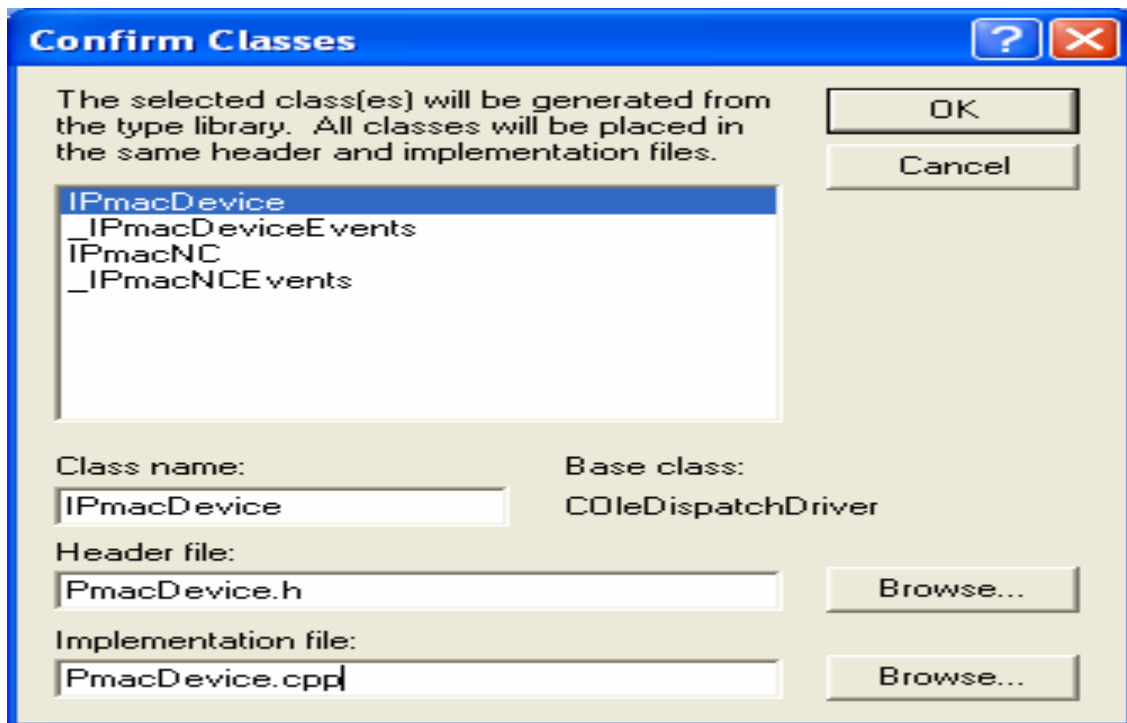
5. Add new class from type library.



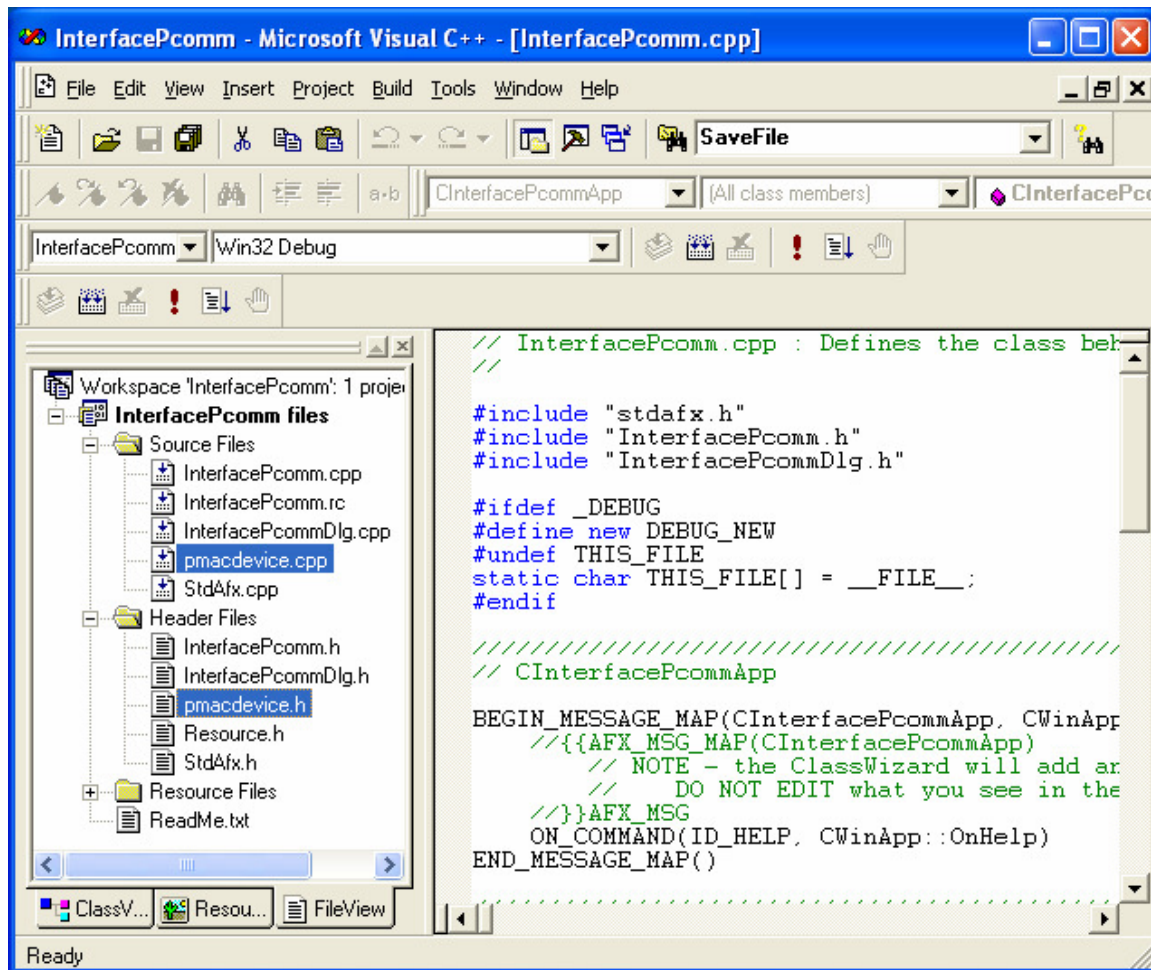
6. Locate, select and open Pmacserver.tlb or Pcommserver.Exe file.



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



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.



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;
```

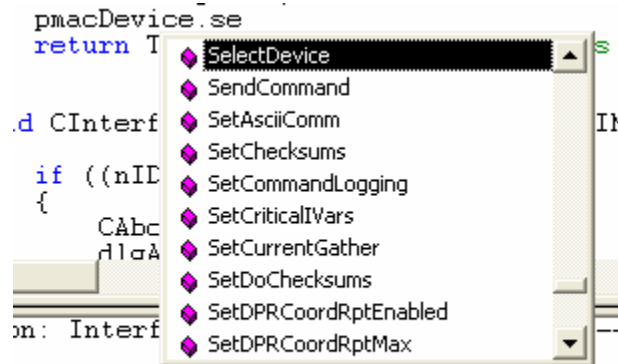
```
CoInitialize(NULL);
```

```
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.



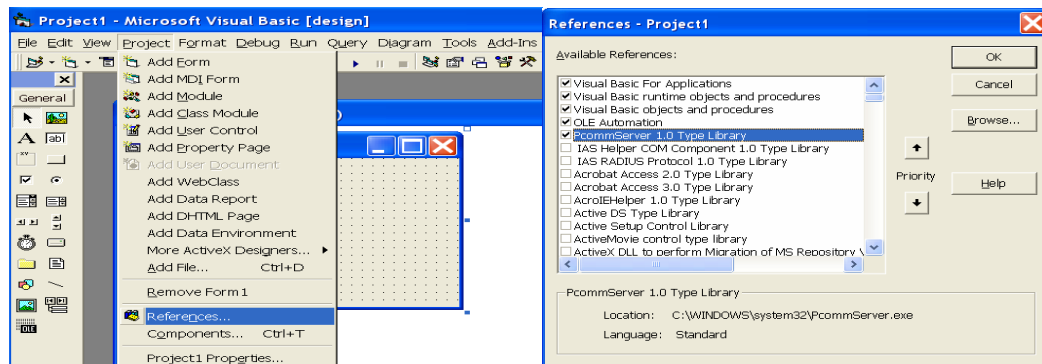
For Example : Add this code to open SelectDevice Dialog box.

```
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

1. Open a new project as standard application from the file menu.
2. From the project menu select References to locate PcommServer Library and enable the checkbox.



3. Add the following line at the top of your main form to get reference to PcommServer library and its methods.

```
Public pmacdoc As New PCOMMSERVERLib.PmacDevice
```

4. Following code gets you going.

```
Private Sub Command1_Click()  
    Dim dwDevice As Long  
    Dim status As Boolean  
    Call pmacdoc.SelectDevice(CLng(Form1.hWnd), dwDevice, status)  
End Sub
```

NOTE: In order to incorporate all the events, you must declare PcommServer instance by using following statement:

```
Private WithEvents PmacDeviceEvent As PCOMMSERVERLib.PmacDevice
```

Borland C++

C++ is yet another very powerful platform for C++ development and has been used for years by Delta Tau team for developing Pwin32Pro2 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.

Connecting to PcommServer interface using Borland C++

Each ConnctInterface() at the start of application must be is aassociated 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;
    }
    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 & 0xFFFFFFFF) // 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:

5. Configuration, Initialization, and Shutdown Functions
6. ASCII Communication Functions
7. Download Functions
8. DPR Real Time Fixed Data Buffer
9. DPR Variable Background Data Buffer Functions
10. DPR Binary Rotary Buffer Functions
11. DPR Numeric Read / Write Functions
12. 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.

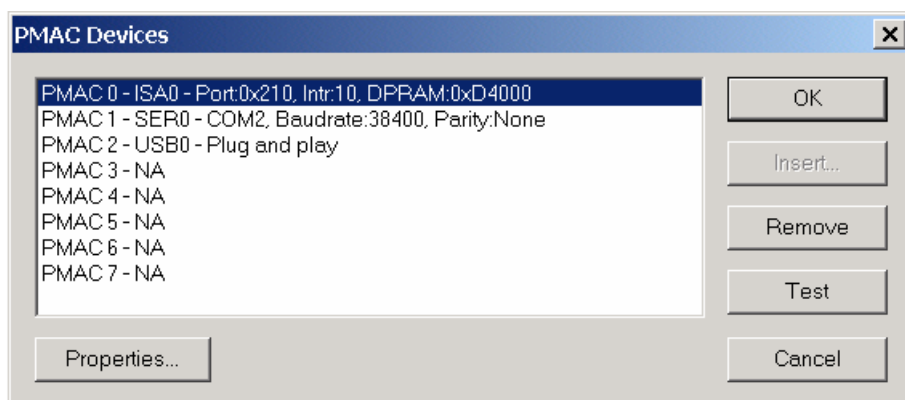
Role change between the return value and status word

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.



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.

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

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

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

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.

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.

```
[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;
```

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

`GetControlResponseEx()` sends a control character to PMAC and potentially returns the ASCII response from PMAC, similar to `GetResponseEx()`.

```
[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 total 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 DEVPMATYPE parameter.

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

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

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

Arguments

dwDevice Device number.

DEVPMATYPE Number corresponding to PmacType. See the PMACType Structure for updated PMAC Types.

DEVPMATYPE Enumeration

```
[Visual Basic]
Public Enum DEVPMATYPE
```

```
[C#]
public enum DEVPMATYPE
```

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

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.

```
[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;
```

Arguments

dwDevice Device number.
pVal DEVASCIIMODE.

DEVASCIIMODE Enumeration

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

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

put_USMonitoringEnabled() and put_USMonitoringPeriod() Methods

Use these methods 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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
put_USMonitoringEnabled(
    /* [in] */ long dwDevice,
    /* [in] */ VARIANT_BOOL newVal) = 0;
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE put_USMonitoringPeriod(
    /* [in] */ long dwDevice,
    /* [in] */ long newVal) = 0;
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE USMonitoringEnabled
    /* [in] */ long dwDevice,
    /* [in] */ VARIANT_BOOL newVal) = 0;
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE USMonitoringPeriod(
    /* [in] */ long dwDevice,
    /* [in] */ long newVal) = 0;
```

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 *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.

PmacReset() Method

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

```
[Visual Basic]  
Overrideable 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 not respond within 30 seconds then returns a timeout 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++]
virtual /* [helpstring][id] */ HRESULT get_DPRTesting(
    /*[in] */ long dwDevice,
    /*[out, retval] */ VARIANT_BOOL *pVal) = 0;
```

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 <i>macro</i> parameter is set to TRUE.
Filename.LOG	The status of the download at each stage is recorded when the <i>log</i> 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 <i>map</i> 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.

UPLOAD PMAC CONFIGURATION

Starting with **version 4.2.5.0** we support a comprehensive PMAC configuration backup function Upload().

DEVUPLOAD structure

```
typedef struct DEVUPLOAD
{
    VARIANT_BOOL single_backup;
    VARIANT_BOOL user_backup;
    VARIANT_BOOL coord_sys;
    VARIANT_BOOL important_register;
    VARIANT_BOOL motion;
    VARIANT_BOOL plc;
    VARIANT_BOOL comp_tables;
    VARIANT_BOOL plcc;
    VARIANT_BOOL pvar;
    VARIANT_BOOL qvar;
    VARIANT_BOOL mvar;
    VARIANT_BOOL ivar;
    VARIANT_BOOL option_16;
    VARIANT_BOOL option_16e;
    VARIANT_BOOL macro;
    VARIANT_BOOL racro_ring_order;
}DEVUPLOAD;
```

Upload() Method

This function uploads complete or partial PMAC configuration to the string specified. DEVUPLOAD is a structure containing all the variables, programs and other configuration parameters to be backed up.

For multi-file backup option the function executes separately for each option selected and saves data in separate files later uploads.

[Visual Basic]

```
Public Sub Upload( _
    ByVal dwDevice As Integer, _
    ByRef pUpload As E:DEVUPLOAD, _
    ByVal pData As String, _
    ByVal pStatus As Integer, _
)
```

[C#]

```
virtual public void Upload(
    int dwDevice,
    ByRef pUpload As E:DEVUPLOAD,
    String pData,
    int pStatus
);
```

[C++]

```
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE Upload(
    /* [in] */ long dwDevice,
    /* [in] */DEVUPLOAD pUpload,
```

```
/* [out] */ BSTR* pData,  
/* [out] */ long *pStatus) = 0;
```

Arguments

dwDevice	Device number.
pUpload	Complete upload structure
pData	Pointer to Data..
pStatus	Complete status structure.

GETTING PMAC STATUS

See section “Data Types, Structures, Callbacks, and Constants” for detailed Motor, CS and Global structures

Notes:

13. All of the above structures are listed in detail at the end of this manual.
 14. A detailed VB.NET example project “PmacDeviceStatus” is provided for users’ reference.
 15. 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.
-

Status Query functions

Following methods provide detailed Global, Servo and Coordinate Systems status structures for all PMAC types.

```
GetTurboMotorStatusEx()  
GetNTMotorStatusEx()  
GetTurboGlobalStatusEx()  
GetNTGlobalStatusEx()  
GetTurboCoordinateStatusEx()  
GetNTCoordinateStatusEx()
```

GetTurboMotorStatusEx() and GetNTMotorStatusEx() Methods

```
[Visual Basic]  
Sub GetTurboMotorStatusEx( _  
    ByVal dwDevice As Integer, _  
    ByVal lMotor As Integer, _  
    ByRef pStatus As E:DEVMSTURBO, _  
    ByRef pbSuccess As Boolean _  
)
```

```
[Visual Basic]  
Sub GetNTMotorStatusEx( _  
    ByVal dwDevice As Integer, _  
    ByVal lMotor As Integer, _  
    ByRef pStatus As E:DEVMSNONTURBO, _  
    ByRef pbSuccess As Boolean _  
)
```

```
[C#]  
virtual public void GetTurboMotorStatusEx(  
    int dwDevice,  
    int lMotor,  
    out E:DEVMSTURBO pStatus  
    out bool pbSuccess  
) ;
```

```
[C#]
virtual public void GetNTMotorStatusEx(
    int dwDevice,
    int lMotor,
    out E:DEVMSNONTURBO pStatus
    out bool pbSuccess
);
```

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

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

Arguments

dwDevice Device number.

lMotor Motor number-1, Motor index starts at 0.

pStatus Complete status structure.

pbSuccess Pointer to successful completion of function.

GetTurboCoordinateStatusEx() and GetNTCoordinateStatusEx Methods

```
[Visual Basic]
Sub GetTurboCoordinateStatusEx( _
    ByVal dwDevice As Integer, _
    ByVal dwCoord As Integer, _
    ByRef pStatus As E:DEVCSSTURBO, _
    ByRef pbSuccess As Boolean _
)
```

```
[Visual Basic]
Sub GetNTCoordinateStatusEx( _
    ByVal dwDevice As Integer, _
    ByVal dwCoord As Integer, _
    ByRef pStatus As E:DEVCSNONTURBO, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
virtual public void GetTurboCoordinateStatusEx(
    int dwDevice,
    int dwCoord,
```

```
    out E:DEVCSURBO pStatus
    out bool pbSuccess
);
```

```
[C#]
virtual public void GetNTCoordinateStatusEx(
    int dwDevice,
    int dwCoord,
    out E:DEVCSNONTURBO pStatus
    out bool pbSuccess
);
```

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

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

Arguments

dwDevice	Device number.
dwCoord	CS number -1, CS index starts at 0.
pStatus	Complete status structure.
pbSuccess	Pointer to successful completion of function.

GetTurboGlobalStatusEx() and GetNTGlobalStatusEx() Methods

```
[Visual Basic]
Sub GetTurboGlobalStatusEx( _
    ByVal dwDevice As Integer, _
    ByRef pStatus As E:DEVGSTURBO, _
    ByRef pbSuccess As Boolean _
)
```

```
[Visual Basic]
Sub GetNTGlobalStatusEx( _
    ByVal dwDevice As Integer, _
    ByRef pStatus As E:DEVGSNONTURBO, _
    ByRef pbSuccess As Boolean _
)
```

```
[C#]
virtual public void GetTurboGlobalStatusEx(
    int dwDevice,
    out E:DEVGSTURBO pStatus
    out bool pbSuccess
);
```

```
[C#]
virtual public void GetNTGlobalStatusEx(
    int dwDevice,
    out E:DEVGSNONTURBO pStatus
    out bool pbSuccess
);
```

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

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

Arguments

dwDevice	Device number.
pStatus	Complete status structure.
pbSuccess	Pointer to successful completion of function.

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

Arguments

dwDevice Device number.

pbSuccess Pointer to successful completion of function.

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

Arguments

dwDevice Device number.

pSources Pointer to sources

pSamples Pointer to sources.

pbSuccess Pointer to successful completion of function.

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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetGatherSamples(
    /* [in] */ long dwDevice,
    /* [in] */ long sourceNum,
    /* [out] */ VARIANT *pVariant,
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

Arguments

dwDevice	Device number.
sourceNum	Pointer to source number
pVariant	Pointer to variant for data collection.
pbSuccess	Pointer to successful completion of function.

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

Arguments

dwDevice	Device number.
pVal	Pointer to Number of sources.
pbSuccess	Pointer to successful completion of function.

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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE SetGather(
    /* [in] */ long dwDevice,
    /* [in] */ long num,
    /* [in] */ BSTR str,
    /* [in] */ VARIANT_BOOL bEnable,
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetGather(
    /* [in] */ long dwDevice,
    /* [in] */ long num,
    /* [in,out] */ BSTR *pStr,
```

```
/* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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++]

```
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE SetQuickGather(
    /* [in] */ long dwDevice,
    /* [in] */ long lComMask,
    /* [in] */ long lEncMask,
    /* [in] */ long lDacMask,
    /* [in] */ long lCurMask,
    /* [in] */ VARIANT_BOOL bEnable,
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE  
SetQuickGatherWithDirectCurrent(  
    /* [in] */ long dwDevice,  
    /* [in] */ long lComMask,  
    /* [in] */ long lEncMask,  
    /* [in] */ long lDacMask,  
    /* [in] */ long lCurMask,  
    /* [in] */ VARIANT_BOOL bEnable,  
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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

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++]
```

```
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE ClearGather(
    /* [in] */ long dwDevice) = 0;
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE ClearGatherData(
    /* [in] */ long dwDevice) = 0;
```

Arguments

dwDevice Device number.

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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE InitGather(
    /* [in] */ long dwDevice,
    /* [in] */ long size,
    /* [in] */ double msec,
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

Arguments

dwDevice Device number.

size Whether to collect the data or not?

msec time in milliseconds.

pbSuccess Pointer to successful completion of function

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

Arguments

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

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++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetGatherPoint(  
    /* [in] */ long dwDevice,  
    /* [in] */ long sourceNum,  
    /* [in] */ long sampleNum,  
    /* [out] */ double *pVal,  
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

Arguments

dwDevice	Device number.
sourceNum	number of sources to collect.

sampleNum	number of samples.
pVal	Pointer to value.
pbSuccess	Pointer to successful completion of function.

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

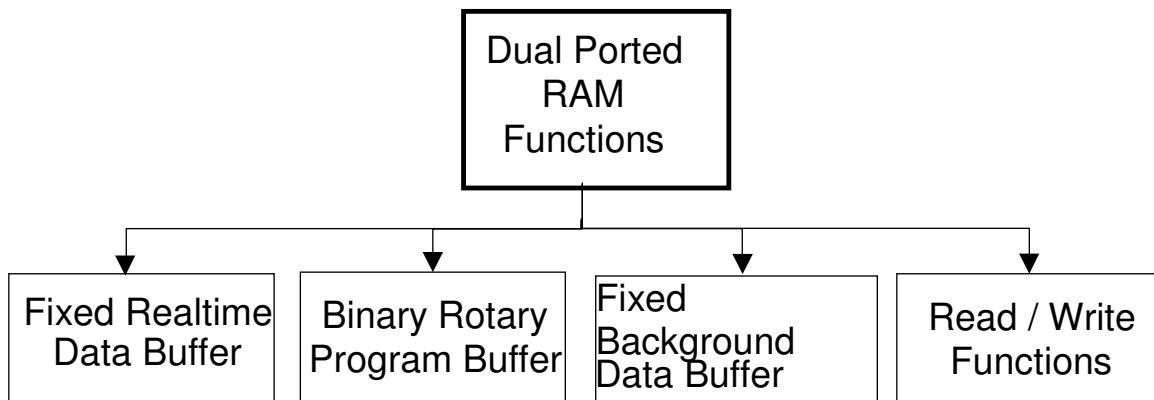
PCOMMSERVER DPR FEATURES

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 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

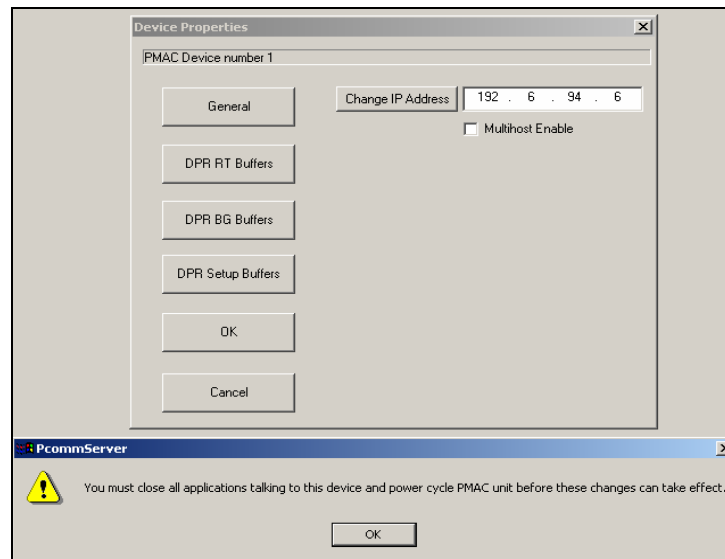
For both RealTime and Background Data buffers the initialization is now done via the properties tab the PmacSelect() dialog. Please see that following properties tab which explains on turning ON/OFF both these buffers, setting up the motor mask and setting the monitor period for these data reporting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

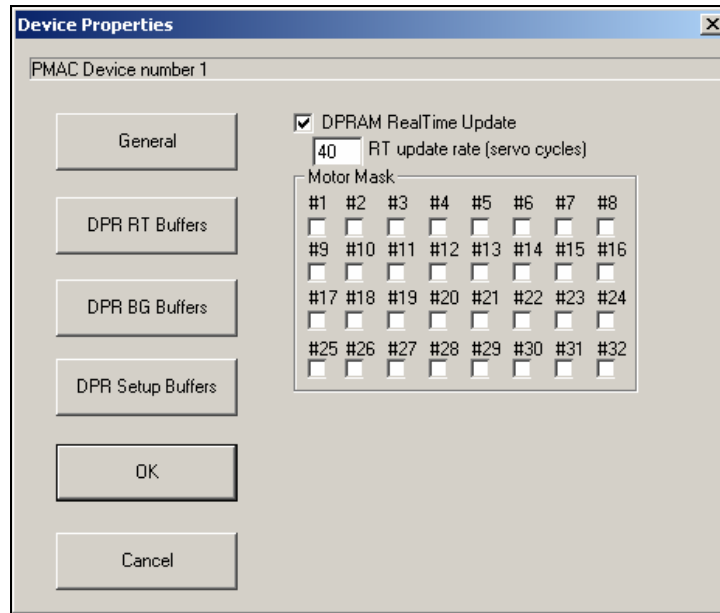
General properties include setting Interrupts enable/disable checkbox and IP address change as well as Multihost check box.



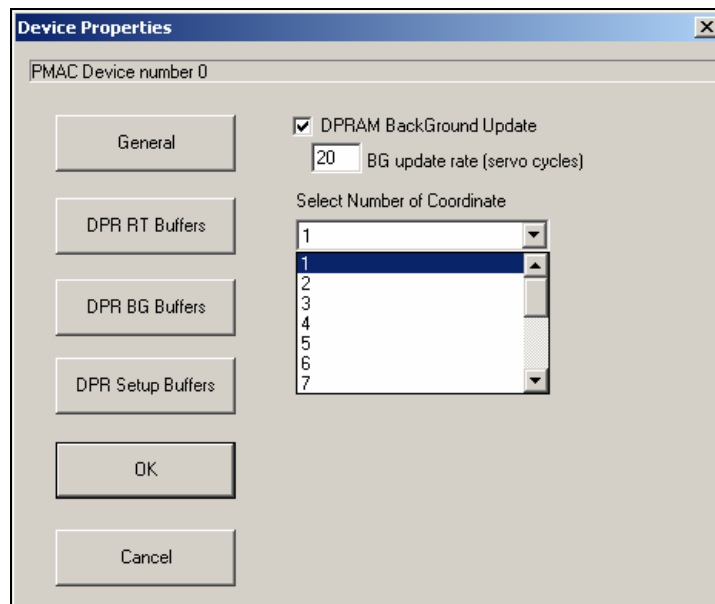
Note:

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.



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 update data in the DPRAM.



DPR Setup Buffers provide setup screen for Binary Rotary buffers.

The image shows a 'Device Properties' dialog box for 'PMAC Device number 1'. It has a sidebar with buttons for 'General', 'DPR RT Buffers', 'DPR BG Buffers', 'DPR Setup Buffers' (which is selected), 'OK', and 'Cancel'. The main area is titled 'DPR Setup Buffers' and contains the following fields:

- Gather Size: 0
- Variable Read Size: 384
- Variable Write Size: 32
- Bin. Rot. Buffer Sizes table:

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

GetCommandedPos()

```
[Visual Basic]
Sub GetCommandedPos( _
    ByVal dwDevice As Integer, _
    ByVal mtr (Actual-1) As Integer, _
    ByVal scale As Double, _
    ByRef pVal As Double _
)
```

```
[C#]
void GetCommandedPos(
    int dwDevice,
    int mtr,
    Double scale,
    out Double pVal
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetCommandedPos(
    /* [in] */ long dwDevice,
    /* [in] */ long mtr,
    /* [in] */ double scale,
    /* [out] */ double *pVal) = 0;
```

GetNetActualPosition()

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

```
[C#]
virtual public void GetNetActualPosition(
    int dwDevice,
    int mtr,
    Double scale,
    out Double pVal
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetNetActualPosition(
    /* [in] */ long dwDevice,
    /* [in] */ long mtr,
```



```
/* [in] */ double scale,  
/* [out] */ double *pVal) = 0;
```

GetFollowError()

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

```
[C#]  
virtual public void GetFollowError(  
    int dwDevice,  
    int mtr,  
    Double scale,  
    out Double pVal  
);
```

```
[C++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetFollowError(  
    /* [in] */ long dwDevice,  
    /* [in] */ long mtr,  
    /* [in] */ double scale,  
    /* [out] */ double *pVal) = 0;
```

GetVelocity()

```
[Visual Basic]  
Sub GetVelocity( _  
    ByVal dwDevice As Integer, _  
    ByVal mtr As Integer, _  
    ByVal scale As Double, _  
    ByRef pVal As Double _  
)
```

```
[C#]  
void GetVelocity(  
    int dwDevice,  
    int mtr,  
    Double scale,  
    out Double pVal  
);
```

```
[C++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetVelocity(  
    /* [in] */ long dwDevice,
```

```

/* [in] */ long mtr,
/* [in] */ double scale,
/* [out] */ double *pVal) = 0;

```

GetMasterPos()

[Visual Basic]

```

Overridable Public Sub GetMasterPos( _
    ByVal dwDevice As Integer, _
    ByVal mtr As Integer, _
    ByVal scale As Double, _
    ByRef pVal As Double _
) Implements !PmacDevice.GetMasterPos

```

[C#]

```

virtual public void GetMasterPos(
    int dwDevice,
    int mtr,
    Double scale,
    out Double pVal
);

```

[C++]

```

virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetMasterPos(
    /* [in] */ long dwDevice,
    /* [in] */ long mtr,
    /* [in] */ double scale,
    /* [out] */ double *pVal) = 0;

```

GetCompensationPos()

[Visual Basic]

```

Sub GetCompensationPos( _
    ByVal dwDevice As Integer, _
    ByVal mtr As Integer, _
    ByVal scale As Double, _
    ByRef pVal As Double _
)

```

[C#]

```

void GetCompensationPos(
    int dwDevice,
    int mtr,
    Double scale,
    out Double pVal
);

```

[C++]

```

virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetCompensationPos(
    /* [in] */ long dwDevice,

```

```
/* [in] */ long mtr,  
/* [in] */ double scale,  
/* [out] */ double *pVal) = 0;
```

USING THE DPR BACKGROUND FIXED DATA BUFFER

GetTargetPos()

```
[Visual Basic]
Sub GetTargetPos( _
    ByVal dwDevice As Integer, _
    ByVal mtr As Integer, _
    ByVal scale As Double, _
    ByRef pVal As Double _
)
```

```
[C#]
void GetTargetPos(
    int dwDevice,
    int mtr,
    Double scale,
    out Double pVal
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetTargetPos(
    /* [in] */ long dwDevice,
    /* [in] */ long mtr,
    /* [in] */ double scale,
    /* [out] */ double *pVal) = 0;
```

GetBiasPos()

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

```
[C#]
virtual public void GetBiasPos(
    int dwDevice,
    int mtr,
    Double scale,
    out Double pVal
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetBiasPos(
    /* [in] */ long dwDevice,
```

```
/* [in] */ long mtr,  
/* [in] */ double scale,  
/* [out] */ double *pVal) = 0;
```

GetFeedRateAndMode()

```
[Visual Basic]  
Overridable Public Sub GetFeedRateAndMode( _  
    ByVal dwDevice As Integer, _  
    ByVal dwCoord As Integer, _  
    ByRef pRate As Double, _  
    ByRef pMode As Boolean _  
) Implements !PmacDevice.GetFeedRateAndMode
```

```
[C#]  
virtual public void GetFeedRateAndMode(  
    int dwDevice,  
    int dwCoord,  
    out Double pRate,  
    out bool pMode  
);
```

```
[C++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE GetFeedRateAndMode(  
    /* [in] */ long dwDevice,  
    /* [in] */ long dwCoord,  
    /* [out] */ double *pRate,  
    /* [out] */ VARIANT_BOOL *pMode) = 0;
```

GetProgramMode()

```
[Visual Basic]  
Sub GetProgramMode( _  
    ByVal dwDevice As Integer, _  
    ByVal dwCoord As Integer, _  
    ByRef pVal As E:DEVPROGRAMMODE _  
)
```

```
[C#]  
void GetProgramMode(  
    int dwDevice,  
    int dwCoord,  
    out E:DEVPROGRAMMODE pVal  
);
```

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

DEVPROGRAMMODE enum

```
typedef enum DEVPROGRAMMODE
{
    DEV_PRG_STOP,
    DEV_PRG_RUN,
    DEV_PRG_STEP,
    DEV_PRG_HOLD,
    DEV_PRG_JOGHOLD,
    DEV_PRG_JOGSTOP
} DEVPROGRAMMODE;
```

GetProgramMotionMode()

```
[Visual Basic]
Overridable Public Sub GetProgramMotionMode( _
    ByVal dwDevice As Integer, _
    ByVal dwCoord As Integer, _
    ByRef pVal As E:DEVOTIONMODE _
) Implements !PmacDevice.GetProgramMotionMode
```

```
[C#]
virtual public void GetProgramMotionMode(
    int dwDevice,
    int dwCoord,
    out E:DEVOTIONMODE pVal
);
```

Implements

!PmacDevice.GetProgramMotionMode

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

DEVOTIONMODE enum

```
typedef enum DEVOTIONMODE
{
    DEV_MOTIONMODE_LINEAR,
    DEV_MOTIONMODE_RAPID,
    DEV_MOTIONMODE_CIRCW,
    DEV_MOTIONMODE_CIRCCW,
    DEV_MOTIONMODE_SPLINE,
    DEV_MOTIONMODE_PVT
} DEVOTIONMODE;
```

USING 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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRRotBufClr(
    /* [in] */ long dwDevice,
    /* [in] */ long bufnum ) = 0;
```

Arguments

dwDevice Device number.

bufnum Which of the two rotary buffers to reference.

DPRSetRotBuf() Method

Once initialized the *DPRSetRotBuf()* 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.

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

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

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

Arguments

dwDevice Device number.

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

DPRAsciiStrToRotEx() Method

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.

```
[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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRAsciiStrToRotEx(
    /* [in] */ long dwDevice,
    /* [in] */ BSTR inpstr,
    /* [in] */ long bufnum,
    /* [in] */ VARIANT_BOOL bSendRemaining,
    /* [out] */ long *pStatus) = 0;
```

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_070	-70	RS274 to BIN DPROT DPR or PMAC Rotary Buffer Size Err
IDS_ERR_063	-63	"RS274 to BIN DPROT Integer number out of range"
IDS_ERR_062	-62	"RS274 to BIN DPROT Illegal Command or Format in string"
IDS_ERR_061	-61	"RS274 to BIN DPROT Unable to convert string to float number"
IDS_ERR_060	-60	"RS274 to BIN DPROT Unable to pack floating point number"
IDS_ERR_059	-59	"RS274 to BIN DPROT Unable to allocate memory"
DprNOK	-1	DPR Binary Rotary Buffer Not present or not configured.
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. The example program is available for VB, C#, C++ for most versions of Visual Studio.

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

```
DPRGetFixedDouble()
```

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++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRGetMem(  
    /* [in] */ long dwDevice,  
    /* [in] */ long offset,  
    /* [in] */ long bytes,  
    /* [out] */ VARIANT *pVal,  
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRSetMem(  
    /* [in] */ long dwDevice,  
    /* [in] */ long offset,  
    /* [in] */ long bytes,  
    /* [in] */ VARIANT Val,  
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRGetShort(
    /* [in] */ long dwDevice,
    /* [in] */ long address,
    /* [out] */ short* pVal,
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRSetShort(
    /* [in] */ long dwDevice,
    /* [in] */ long address,
    /* [in] */ short newVal,
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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

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

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++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRGetFloat(  
    /* [in] */ long dwDevice,  
    /* [in] */ long offset,  
    /* [out] */ float *pVal,  
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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++]  
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRSetFloat(  
    /* [in] */ long dwDevice,  
    /* [in] */ long offset,  
    /* [in] */ float newVal,  
    /* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

Arguments

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

The "Dual Word" Conversion function converts data that is placed in DPR by one of its 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.

DPRGetFixedDouble() Method

This method replaces the old DPRLFixed function.

This method reads 2 32-bit words from DPRAM holding 24-bits each of the 48-bit word. Then DPRLFixed() converts the data (long array) to a meaningful numeric value. In PcommServer library, DPRGetFixedDouble() method executes the following sequence:

```
long d[2] = {0};
(theApp.DPRGetMem(dwDevice,address,sizeof(long) * 2,d) != NULL) ?
    *pbSuccess = VARIANT_TRUE : *pbSuccess = VARIANT_FALSE;
if(pbSuccess)
    *pVal = theApp.DPRLFixed(d,1.0); // Second parameter passed to DPRLFixed() is the scale factor.
else
    *pVal = 0.0;
return S_OK;
```

whereas DPRLFixed() executes:

```
double CPmac::DPRLFixed( long    d[], double scale )
{
    // 48-bit fixed point word:
    // 2 32 bit    words holding 24 bits each of the 48 bit words.

    double result;

    result = ( double ) ( d[0] & 0x0FFFFFFF ) +
        (( double ) ( d[1] )) * 16777216.0;
    return result * scale;
}
```

[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++]

```
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRGetFixedDouble(
    /* [in] */ long dwDevice,
    /* [in] */ long address,
    /* [out] */ double* pVal,
```



```
/* [out] */ VARIANT_BOOL *pbSuccess) = 0;
```

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 for user convenience only.

***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++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE DPRFloat(
    /* [in] */ __int64 d,
    /* [in] */ double scale,
    /* [out] */ double *pVal) = 0;
```

Arguments

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

***strto32f()* Method**

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

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

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE strto32f(
    /* [in] */ BSTR str,
    /* [out] */ double *pVal) = 0;
```

strtod32dp() Method

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

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

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE strtod32dp(
    /* [in] */ BSTR str,
    /* [out] */ double *pVal) = 0;
```

strtod24() Method

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

```
[C#]
void strtod24(
    string str,
    out Double pVal
);
```

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE strtod24(
    /* [in] */ BSTR str,
    /* [out] */ double *pVal) = 0;
```

strtod48f() Method

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

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

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE strtod48f(
    /* [in] */ BSTR str,
    /* [out] */ double *pVal) = 0;
```

strtod48l() Method

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

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

```
[C++]
virtual /* [helpstring][id] */ HRESULT STDMETHODCALLTYPE strtod48l(
    /* [in] */ BSTR str,
    /* [out] */ double *pVal) = 0;
```

DATA TYPES, STRUCTURES, CALLBACKS, AND CONSTANTS

GLOBALSTATUS for TURBO and 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_adrsd  : 1; // 0
  USHORT all_adrsd   : 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; // 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_ERROR         & 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_adrsd : 1;      // 0
USHORT all_adrsd : 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 earon_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

Motion

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

MOTIONMODE

```
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_seq_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 padding               : 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      & 0x8000000000000000")
cpp_quote ("#define CST_Z_AXIS_INCREMENT_MODE        & 0x4000000000000000")
cpp_quote ("#define CST_Y_AXIS_USED_IN_FEEDRATE      & 0x2000000000000000")
cpp_quote ("#define CST_Y_AXIS_INCREMENT_MODE        & 0x1000000000000000")
cpp_quote ("#define CST_X_AXIS_USED_IN_FEEDRATE      & 0x0800000000000000")
cpp_quote ("#define CST_X_AXIS_INCREMENT_MODE        & 0x0400000000000000")
cpp_quote ("#define CST_W_AXIS_USED_IN_FEEDRATE      & 0x0200000000000000")
cpp_quote ("#define CST_W_AXIS_INCREMENT_MODE        & 0x0100000000000000")
cpp_quote ("#define CST_V_AXIS_USED_IN_FEEDRATE      & 0x0080000000000000")
cpp_quote ("#define CST_V_AXIS_INCREMENT_MODE        & 0x0040000000000000")
cpp_quote ("#define CST_U_AXIS_USED_IN_FEEDRATE      & 0x0020000000000000")
cpp_quote ("#define CST_U_AXIS_INCREMENT_MODE        & 0x0010000000000000")
cpp_quote ("#define CST_C_AXIS_USED_IN_FEEDRATE      & 0x0008000000000000")
cpp_quote ("#define CST_C_AXIS_INCREMENT_MODE        & 0x0004000000000000")
```

```

cpp_quote ("#define CST_B_AXIS_USED_FEEDRATE          & 0x00020000000000000000")
cpp_quote ("#define CST_B_AXIS_INCREMENT_MODE          & 0x00010000000000000000")
cpp_quote ("#define CST_A_AXIS_USED_IN_FEEDRATE            & 0x00008000000000000000")
cpp_quote ("#define CST_A_AXIS_INCREMENT_MODE                & 0x00004000000000000000")
cpp_quote ("#define CST_RADIUS_VEC_INCR_MODE                 & 0x00002000000000000000")
cpp_quote ("#define CST_CONTINUOUS_MOTION_REG                  & 0x00001000000000000000")
cpp_quote ("#define CST_MOVE_SPEC_BY_TIME                      & 0x00000800000000000000")
cpp_quote ("#define CST_CONTINUOUS_MOTION_MODE                  & 0x00000400000000000000")
cpp_quote ("#define CST_SINGLE_STEP_MODE                        & 0x00000200000000000000")
cpp_quote ("#define CST_RUNNING_PROGRAM                         & 0x00000100000000000000")
cpp_quote ("#define CST_LOOKAHEAD_IN_PROGRES                   & 0x00000080000000000000")
cpp_quote ("#define CST_RUN_TIME_ERROR                          & 0x00000040000000000000")
cpp_quote ("#define CST_INTERNAL_MOVE_IN_STACK                 & 0x00000020000000000000")
cpp_quote ("#define CST_AMP_FAULT_ERROR                         & 0x00000010000000000000")
cpp_quote ("#define CST_FATAL_FOLLOWING_ERROR                  & 0x00000008000000000000")
cpp_quote ("#define CST_WARNING_FOLLOWING_ERROR                 & 0x00000004000000000000")
cpp_quote ("#define CST_IN_POSITION                            & 0x00000002000000000000")
cpp_quote ("#define CST_ROTARY_BUFFER_FULL                     & 0x00000001000000000000")
cpp_quote ("#define CST_DELAYED_CALC_FLAG                      & 0x00000000800000000000")
cpp_quote ("#define CST_END_OF_BLOCK_STOP_PROGS                 & 0x00000000400000000000")
cpp_quote ("#define CST_INITIAL_SYNC_M_VAR_ONE_SHOT            & 0x00000000200000000000")
cpp_quote ("#define CST_INITIAL_DWELL_MOVE_BUFFERED             & 0x00000000100000000000")
cpp_quote ("#define CST_CTTR_COMP_OUTSIDE_CORNER                & 0x00000000080000000000")
cpp_quote ("#define CST_CTTR_COMP_STOP_REQ                      & 0x00000000040000000000")
cpp_quote ("#define CST_CTTR_COMP_MOVE_BUFFERED                 & 0x00000000020000000000")
cpp_quote ("#define CST_PRE_JOG_MOVE_IN_PROGS                   & 0x00000000010000000000")
cpp_quote ("#define CST_SEGMENTED_MOVE_IN_PROGS                 & 0x00000000008000000000")
cpp_quote ("#define CST_SEGMENT_ACCEL_FLAG                      & 0x00000000004000000000")
cpp_quote ("#define CST_SEGMENT_STOP_REQUEST                    & 0x00000000002000000000")
cpp_quote ("#define CST_PVT_SPLINE_MOVE_MODE                    & 0x00000000001000000000")
cpp_quote ("#define CST_CUTTER_COMP_LEFT                       & 0x00000000000800000000")
cpp_quote ("#define CST_CUTTER_COMP_ON                          & 0x00000000000400000000")
cpp_quote ("#define CST_CCW_CIRCLE_RAPID_MOVE_MODE              & 0x00000000000200000000")
cpp_quote ("#define CST_CIRCLE_SPLINE_MOVE_MODE                 & 0x00000000000100000000")
cpp_quote ("#define CST_LHB_WRAP                                & 0x00000000000080000000")
cpp_quote ("#define CST_INTERNAL_Y22                            & 0x00000000000040000000")
cpp_quote ("#define CST_INTERNAL_Y21                            & 0x00000000000020000000")
cpp_quote ("#define CST_INTERNAL_Y20                            & 0x00000000000010000000")
cpp_quote ("#define CST_LHB_SYNC_M_VAR_OVERFLOW                 & 0x00000000000008000000")
cpp_quote ("#define CST_LHB_DIRECTION                           & 0x00000000000004000000")
cpp_quote ("#define CST_LHB_STOP                                & 0x00000000000002000000")
cpp_quote ("#define CST_LHB_CHANGE                              & 0x00000000000001000000")
cpp_quote ("#define CST_LHB_LAST_SEGMENT                        & 0x00000000000000800000")
cpp_quote ("#define CST_LHB_RECALCULATE                          & 0x00000000000000400000")
cpp_quote ("#define CST_LHB_FLUSH                                & 0x00000000000000200000")
cpp_quote ("#define CST_LHB_LAST_MOVE                           & 0x00000000000000100000")
cpp_quote ("#define CST_LHB_SINGLE_SEGMENT_REQUEST               & 0x00000000000000080000")
cpp_quote ("#define CST_LHB_CHANGE_REQUEST                      & 0x00000000000000040000")
cpp_quote ("#define CST_LHB_MOVE_REQUEST                        & 0x00000000000000020000")
cpp_quote ("#define CST_LHB_DIRECTION_REQUEST                   & 0x00000000000000010000")
cpp_quote ("#define CST_RESERVED_Y7                             & 0x00000000000000008000")
cpp_quote ("#define CST_RESERVED_Y6                             & 0x00000000000000004000")
cpp_quote ("#define CST_RESERVED_Y5                             & 0x00000000000000002000")
cpp_quote ("#define CST_RESERVED_Y4                             & 0x00000000000000001000")
cpp_quote ("#define CST_RADIUS_ERROR                            & 0x00000000000000000800")
cpp_quote ("#define CST_PROGRAM_RESUME_ERROR                     & 0x00000000000000000400")
cpp_quote ("#define CST_DESIRED_POSITION_LIMIT_STOP              & 0x00000000000000000200")
cpp_quote ("#define CST_IN_PROGRAM_PMATCH                       & 0x00000000000000000100")

```

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;
```

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;
```

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_IN_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_IN_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_IN_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_FVT_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_TRIGGER_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_LIMIT_SET & 0x200000000000")
cpp_quote ("#define MSNT_HANDWHEEL_ENABLE & 0x100000000000")
cpp_quote ("#define MSNT_PHASED_MOTOR & 0x080000000000")
cpp_quote ("#define MSNT_OPEN_LOOP_MODE & 0x040000000000")
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           & 0x0000000000400")
cpp_quote ("#define MSNT_RESERVED_Y9             & 0x0000000000200")
cpp_quote ("#define MSNT_PHASING_SEARCH_ERROR     & 0x0000000000100")
cpp_quote ("#define MSNT_TRIGGER_MOVE             & 0x0000000000080")
cpp_quote ("#define MSNT_INTEG_FATAL_FOLLOW_ERR   & 0x0000000000040")
cpp_quote ("#define MSNT_I2T_AMP_FAULT_ERROR      & 0x0000000000020")
cpp_quote ("#define MSNT_BACKLASH_DIRECTION_FLAG  & 0x0000000000010")
cpp_quote ("#define MSNT_AMP_FAULT_ERROR           & 0x0000000000008")
cpp_quote ("#define MSNT_FATAL_ERROR_EXCEEDED     & 0x0000000000004")
cpp_quote ("#define MSNT_WARNING_ERROR_EXCEEDED   & 0x0000000000002")
cpp_quote ("#define MSNT_IN_POSITION_TRUE         & 0x0000000000001")
```


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

ASCII Communication Functions	39	Fixed Real Time Data buffer	71
Binary Rotary Buffer	35, 71, 72, 83, 84, 85	GLOBALSTATUS	96, 99
Configuration Functions	36	MOTION	99, 101
Control Panel	9	MOTIONMODE	99
Data Types, Structures, Callbacks, and Constants...	96	PROGRAM	i, 100, 102
DPR Background Fixed Data Buffer	80	Read/Write Functions	72
DPR Numeric Read And Write	86	Shutting Down Communication	28
Dual Word Conversion.....	86	Standard Read/Write.....	86
Fixed Background Data Buffer.....	71	Time.....	35, 71, 73, 76, 96, 103

