

AV_PTP

Programmable Telemetry Processor

Remote Interface Library

Programmer's Guide

Version: 1.49

Date Prepared: June 27, 2001

Export Controls: Avtec's products and technology are controlled for export purposes by the U.S. Government pursuant to the Arms Export Control Act and the International Traffic in Arms Regulations or the Export Administration Act and the Export Administration Regulations. It is the company's responsibility to comply with the applicable export laws and regulations before engaging in export activities, including engaging in exports, reexports, or the disclosure of technical data in the U.S. to foreign persons, a deemed export.



TM 01-014

AV_PTP

Programmable Telemetry Processor

Remote Interface Library

Programmer's Guide

Copyright © 2001 Avtec Systems, Inc. All rights reserved. No part of this document may be reproduced without prior written permission from Avtec Systems.

The information in this document has been fully reviewed and is believed to be entirely reliable. AVTEC reserves the right, however, to modify any products herein to improve reliability, function or design. AVTEC does not assume any liability arising out of the application or use of any product or circuit described herein. AVTEC does not convey any license under its patent rights or the rights of others.

Support

Technical Support

This manual provides engineers with integration and programming information necessary to develop a system using the Avtec PTP for Windows. The Avtec PTP for Windows is a PC-based telemetry and command processing system that supports both CCSDS and TDM data formats. Technical Support is available from Avtec Systems, Inc.

Avtec Systems, Inc.
Attn: Avtec PTP for Windows Technical Support
10530 Rosehaven St., Suite 300
Fairfax, VA 22030-2840
(703) 273-2211
support@avtec.com

Customer Service

Direct non-technical questions and business-related matters to Customer Support at the same number.

REVISION HISTORY

REV.	DESCRIPTION	DATE	APPD.
-00	Baseline release.	7-23-01	7-23-01

TABLE OF CONTENTS

<u>DOCUMENT SCOPE</u>	IX
<u>HOW TO USE THIS MANUAL</u>	IX
<u>WHAT'S NEW SINCE VERSION 1.40</u>	X
<u>New Functions:</u>	x
<u>New Error Responses:</u>	x
<u>New Data Types:</u>	x
<u>RELATED DOCUMENTATION</u>	XI
<u>Additional Related Documentation</u>	xi
<u>CHAPTER 1 OVERVIEW</u>	1-1
SECTION 1 INTRODUCTION	1-1
SECTION 2 AVTEC PTP FOR WINDOWS ARCHITECTURE	1-1
SECTION 3 THEORY OF OPERATION	1-3
<u>Avtec PTP for Windows Software</u>	1-3
<u>The Avtec PTP for Windows Desktop</u>	1-4
<u>Avtec PTP for Windows Module Functions</u>	1-4
SECTION 4 FRONT-END TELEMETRY AND COMMAND PROCESSOR	1-5
<u>CHAPTER 2 REMOTE INTERFACE LIBRARY</u>	2-1
SECTION 1 OVERVIEW	2-1
SECTION 2 ERROR RESPONSES	2-8
SECTION 3 DATA TYPES	2-9
SECTION 4 BYTE ALIGNMENT	2-10
SECTION 5 FUNCTION DETAILS	2-11
<u>ptp_DFswap</u>	2-11
<u>ptp_Fswap</u>	2-11
<u>ptp_LLswap</u>	2-11
<u>ptp_Lswap</u>	2-12
<u>ptp_swap</u>	2-12
<u>ptpBeginMultiCastService</u>	2-12
<u>ptpCallFileCleanup</u>	2-13
<u>ptpCancelUserAlarmCallBackEx</u>	2-14
<u>ptpConnectModuleData</u>	2-14
<u>ptpConnectModuleDataEx</u>	2-14
<u>ptpConnectModuleEvent</u>	2-15
<u>ptpConnectModuleEventEx</u>	2-15
<u>ptpConnectToMonitor</u>	2-15
<u>ptpConnectToServer</u>	2-16
<u>ptpCopyMulticastBuffer</u>	2-16
<u>ptpCopyResponseBuffer</u>	2-17
<u>ptpCopyResponseBufferEx</u>	2-17
<u>ptpCreateDefaultPathEX</u>	2-17
<u>ptpCreateNewConfiguration</u>	2-18
<u>ptpCreateNewConfigurationEx</u>	2-18
<u>ptpCreateNewConfigurationAllServers</u>	2-18
<u>ptpDisableAllModules</u>	2-18
<u>ptpDisableAllModulesEx</u>	2-18
<u>ptpDisableAllModulesAllServers</u>	2-18
<u>ptpDisableModule</u>	2-19
<u>ptpDisableModuleEx</u>	2-19
<u>ptpDisablePingEx</u>	2-19
<u>ptpDisconnectFromMonitorEx</u>	2-19

<i>ptpDisconnectFromServer</i>	2-20
<i>ptpDisconnectFromServerEx</i>	2-20
<i>ptpDisconnectFromServerAllServers</i>	2-20
<i>ptpEnableAllModules</i>	2-20
<i>ptpEnableAllModulesEx</i>	2-20
<i>ptpEnableAllModulesAllServers</i>	2-20
<i>ptpEnableKeepAliveEx</i>	2-20
<i>ptpEnableModule</i>	2-21
<i>ptpEnableModuleEx</i>	2-21
<i>ptpEnableNameServer</i>	2-21
<i>ptpEnablePingEx</i>	2-21
<i>ptpEnableSecureBufferMode</i>	2-22
<i>ptpEnableSecureBufferModeEx</i>	2-22
<i>ptpGetConfiguration</i>	2-23
<i>ptpGetConfigurationEx</i>	2-23
<i>ptpGetDefaultPathEx</i>	2-23
<i>ptpGetDesktopIdEx</i>	2-24
<i>ptpGetDesktopName</i>	2-24
<i>ptpGetDesktopNameEx</i>	2-24
<i>ptpGetDesktopTTL</i>	2-24
<i>ptpGetDesktopTTLEx</i>	2-24
<i>ptpGetDriveSpace</i>	2-25
<i>ptpGetEnableStatus</i>	2-25
<i>ptpGetEnableStatusEx</i>	2-25
<i>ptpGetHex</i>	2-26
<i>ptpGetKeepAliveStatusEx</i>	2-26
<i>ptpGetLong</i>	2-26
<i>ptpGetModuleConnections</i>	2-27
<i>ptpGetModuleConnectionsEx</i>	2-27
<i>ptpGetModuleEnableStatus</i>	2-27
<i>ptpGetModuleEnableStatusEx</i>	2-27
<i>ptpGetModuleName</i>	2-28
<i>ptpGetModuleNameEx</i>	2-28
<i>ptpGetModuleState</i>	2-28
<i>ptpGetModuleStateEx</i>	2-28
<i>ptpGetModuleTLL</i>	2-29
<i>ptpGetModuleTTLEx</i>	2-29
<i>ptpGetModuleVersion</i>	2-29
<i>ptpGetModuleVersionEx</i>	2-29
<i>ptpGetPingStateEx</i>	2-30
<i>ptpGetResources</i>	2-30
<i>ptpGetRILVersion</i>	2-31
<i>ptpGetServerVersionEx</i>	2-32
<i>ptpGetStationId</i>	2-32
<i>ptpGetStationIdEx</i>	2-32
<i>ptpGetStationTime</i>	2-32
<i>ptpGetStationTimeEx</i>	2-32
<i>ptpGetText</i>	2-33
<i>ptpGetTimeStateEx</i>	2-33
<i>ptpGetTimeStatusEx</i>	2-34
<i>ptpGetUnknownModuleStateEx</i>	2-35
<i>ptpHasDesktopChangedEx</i>	2-35
<i>ptpHideServerEx</i>	2-36
<i>ptpKillProcess</i>	2-36
<i>ptpKillScheduleThread</i>	2-36

<i>ptpListServers</i>	2-37
<i>ptpLoadModule</i>	2-38
<i>ptpLoadModuleEx</i>	2-38
<i>ptpLoadProfile</i>	2-39
<i>ptpLoadProfileEx</i>	2-39
<i>ptpLoadProfileAllServers</i>	2-39
<i>ptpNoOp</i>	2-39
<i>ptpNoOpEx</i>	2-39
<i>ptpNoOpAllServers</i>	2-39
<i>ptpPollCustomStatusEx</i>	2-39
<i>ptpPollModuleState</i>	2-41
<i>ptpPollModuleStateEx</i>	2-41
<i>ptpPollUnknownModuleStateEx</i>	2-41
<i>ptpPostMessage</i>	2-42
<i>ptpPostMessageEx</i>	2-42
<i>ptpQueryAvailableModules</i>	2-42
<i>ptpQueryAvailableModulesEx</i>	2-42
<i>ptpQueryDirectory</i>	2-42
<i>ptpQueryDirectoryEx</i>	2-42
<i>ptpQueryExistingLogs</i>	2-43
<i>ptpQueryExistingLogsEx</i>	2-43
<i>ptpQueryOpenWindows</i>	2-43
<i>ptpQuerySavedProfiles</i>	2-44
<i>ptpQuerySavedProfilesEx</i>	2-44
<i>ptpReboot</i>	2-44
<i>ptpRegisterCallBack</i>	2-45
<i>ptpRegisterCustomStatusEx</i>	2-45
<i>ptpRegisterUserAlarmCallBackEx</i>	2-46
<i>ptpRemoveModuleData</i>	2-47
<i>ptpRemoveModuleDataEx</i>	2-47
<i>ptpRemoveModuleEvent</i>	2-47
<i>ptpRemoveModuleEventEx</i>	2-47
<i>ptpSaveProfile</i>	2-48
<i>ptpSaveProfileEx</i>	2-48
<i>ptpScheduleFileCleanup</i>	2-48
<i>ptpSendDesktopCommand</i>	2-49
<i>ptpSendDesktopCommandEx</i>	2-49
<i>ptpSendModuleCommand</i>	2-50
<i>ptpSendModuleCommandEx</i>	2-50
<i>ptpSetActivityDelay</i>	2-50
<i>ptpSetActivityDelayEX</i>	2-50
<i>ptpSetConfiguration</i>	2-50
<i>ptpSetConfigurationEx</i>	2-50
<i>ptpSetCustomStatusEx</i>	2-51
<i>ptpSetDefaultPathEx</i>	2-52
<i>ptpSetDesktopTTL</i>	2-53
<i>ptpSetDesktopTTLEx</i>	2-53
<i>ptpSetDesktopTTLAllServers</i>	2-53
<i>ptpSetModuleState</i>	2-53
<i>ptpSetModuleStateEx</i>	2-53
<i>ptpSetModuleTTL</i>	2-54
<i>ptpSetModuleTTLEx</i>	2-54
<i>ptpSetMulticastAddress</i>	2-54
<i>ptpSetMulticastAddressEx</i>	2-54
<i>ptpSetMulticastAddressAllServers</i>	2-54

<i>ptpSetRILSocketTimeOut</i>	2-55
<i>ptpSetStationTimeEx</i>	2-55
<i>ptpSetStatusDelay</i>	2-55
<i>ptpSetStatusDelayEx</i>	2-55
<i>ptpSetTimeStateEx</i>	2-56
<i>ptpSpawn</i>	2-56
<i>ptpUnloadModule</i>	2-57
<i>ptpUnloadModuleEx</i>	2-57
<i>ptpWinCls</i>	2-57
<i>ptpWinGotoxy</i>	2-57
<i>ptpZeroAllModuleCounters</i>	2-58
<i>ptpZeroAllModuleCountersEx</i>	2-58
<i>ptpZeroAllModuleCountersAllServers</i>	2-58
<i>ptpZeroModuleCounters</i>	2-58
<i>ptpZeroModuleCountersEx</i>	2-58
CHAPTER 3 PROGRAMMER'S EXAMPLES	3-1
<i>EXAMPLE 1</i>	3-1
APPENDIX A:	A-1
<i>ACRONYMS AND ABBREVIATIONS</i>	A-1

FIGURES

Figure 1-1: PC-based Front-end Processor Block Diagram	1-2
Figure 1-2: Module Connections for a Front-End Processor	1-5
Figure 2-1: AV_PTP Development Environment	2-1
Figure 3-1: Demo Program Telemetry Data Flow Configuration (TestAvptp.cpp)	3-2
Figure 3-2: Demo Program Command Data Flow Configuration (TestAvptp.cpp)	3-5
Figure 3-3: TestAvptp Program Flow Chart	3-8

TABLES

Table 2-1: AV_PTP Library Functions	2-1
Table 2-2: PB-4 Format	2-33

DOCUMENT SCOPE

This manual provides application programmers with the information necessary to write applications using the Avtec Programmable Telemetry Processor (PTP) for Windows. The Avtec PTP for Windows is a flexible and powerful satellite telemetry and command front-end system. The Avtec PTP for Windows accepts multiple telemetry streams and outputs time-tagged frame or packet data over a network to workstations in a distributed telemetry analysis system. The Avtec PTP for Windows also supports high speed data logging for store and forward operation. The system includes a command gateway that accepts input from a networked command processor and outputs to the uplink. The Avtec PTP for Windows is controlled and monitored using the base or Ex (extended) functions in the AV_PTP Remote Interface Library. The AV_PTP Remote Interface Library communicates with the Avtec PTP for Windows system through network sockets.

HOW TO USE THIS MANUAL

This manual is grouped into three chapters:

Chapter 1: Overview

Summarizes the Avtec PTP for Windows system.

Chapter 2: Remote Interface Library

Describes the relationship among the user's application program, the AV_PTP function calls, the network socket interface, and the Avtec PTP for Windows system. **Chapter 2** also describes each routine in the AV_PTP Remote Interface Library.

Chapter 3: Programmer's Examples

Describes two sample programs that demonstrate the use of AV_PTP.

Appendix A lists the acronyms and abbreviations used in this manual.

WHAT'S NEW SINCE VERSION 1.40

NEW FUNCTIONS:

- ptpCallFileCleanup
- ptpEnableKeepAliveEx
- ptpGetDesktopIdEx
- ptpGetDriveSpace
- ptpGetKeepAliveStatusEx
- ptpGetModuleVersion
- ptpGetModuleVersionEx
- ptpGetResources
- ptpGetRILVersion
- ptpGetServerVersionEx
- ptpGetTimeStateEx
- ptpGetTimeStatusEx
- ptpKillScheduleThread
- ptpPollCustomStatusEx
- ptpRegisterCustomStatusEx
- ptpScheduleFileCleanup
- ptpSetCustomStateEx
- ptpSetStationTimeEx
- ptpSetTimeStateEx

NEW ERROR RESPONSES:

```
#define PTP_SERVERCRCERR
```

-20

NEW DATA TYPES:

All Operating Systems (i.e. NT and Solaris)

```
#define PTP_INT          signed long           /* 32-bit signed integer */
#define PTP_SHORT         signed short          /* 16-bit signed integer */
#define PTP_CHAR          signed char           /* 8-bit signed integer */
#define PTP_LONG          signed long           /* 32-bit signed integer */
#define PTP_BOOL          int                  /* */
```

Non Window Operating Systems (i.e. Solaris)

```
#define HANDLE          void*
#define BOOL             PTP_BOOL
#define UCHAR            PTP_8
#define USHORT           PTP_16
#define WORD             PTP_16
#define ULONG            PTP_32
#define DWORD            PTP_32
#define LPVOID           PTP_32
#define ULONGLONG        PTP_64
```

RELATED DOCUMENTATION

The following documents contain information you may find useful to reference as you read this manual:

- 1) Avtec Systems, Inc. *AT-HSIO2 High Speed Serial I/O Board with Frame Synchronizer Functional Description*. Fairfax, Virginia, 1995.
- 2) Avtec Systems, Inc. *AV_FSTS Frame Synchronizer/Telemetry Simulator Dynamic Link Library for Windows NT Programmer's Guide*. Fairfax, Virginia, 1997.
- 3) Avtec Systems, Inc. *MONARCH-E Frame Synchronizer/Telemetry Simulator with Reed-Solomon Encoder/Decoder Functional Description*. Fairfax, Virginia, 1997.
- 4) Avtec Systems, Inc. *Avtec Programmable Telemetry Processor for Windows User's Guide*. Fairfax, Virginia, 2001.

ADDITIONAL RELATED DOCUMENTATION

- 1) Apogee Labs, Inc. *Model ISA-PSK PSK Modulator User's Manual*. North Wales, Pennsylvania, 1996.
- 2) Apogee Labs, Inc. *Model ISA-STG2 Synchronized Time Generator User's Manual*. North Wales, Pennsylvania, 1996.
- 3) Aydin Corporation, *PC329 Tunable BPSK Demodulator Installation and Operation Manual*. Horsham, Pennsylvania, 1996.
- 4) Aydin Corporation, *PC335 Bit Synchronizer Installation and Operation Manual*. Horsham, Pennsylvania, 1996.
- 5) GDP Space Systems, *PSK004 ISA-BUS BPSK Modulator User's Manual*. Horsham, Pennsylvania, 1995.
- 6) Microdyne Corporation. *Model PCR-2000 Telemetry PC Receiver Operations Manual*. Ocala, Florida, January 1996.
- 7) Veda Systems, Inc. *DB200 Convolutional Decoder Module Technical Reference Manual*. California, Maryland, September 1994.
- 8) Veda Systems, Inc. *Series-2/10 and DBS200 PC Bit Synchronizer Card Technical Reference Manual*. California, Maryland, January 1996.

Chapter 1

Overview

SECTION 1

INTRODUCTION

AVTEC's Programmable Telemetry Processor (PTP) for Windows is a PC-based, multi-channel telemetry and command processing system. The Avtec PTP for Windows' unique data handling capabilities are ideal for a telemetry front-end system which performs data acquisition, real-time network transfer, and store and forward operations. Support for both time-division multiplexed TDM and CCSDS telemetry formats provides the flexibility to support multiple spacecraft with a single front-end system.

The Avtec PTP for Windows acts as a gateway that accepts multiple telemetry streams and outputs time-tagged frame or packet data over a network to workstations in a distributed satellite control and analysis system. The Avtec PTP for Windows also includes a command gateway that accepts input from the network and outputs serial commands to the uplink. Key elements of the Avtec PTP for Windows system are AVTEC's MONARCH-E PCI-based CCSDS/TDM Telemetry Processor/Simulator board, a network-based, distributed computing architecture, and the Windows NT/2000 operating system.

The Avtec PTP for Windows is useful throughout the lifecycle of a satellite system. During development, integration, and test, the front-end system can be used as a test tool in a distributed test environment. During operations, the system is installed at remote ground stations, with network links back to operations center(s) for telemetry and command processing and analysis.

The Avtec PTP for Windows can be controlled locally or remotely via the network using a socket interface. The use of standard network protocols (TCP/IP) facilitates integration with existing telemetry analysis and command processing systems.

SECTION 2

AVTEC PTP FOR WINDOWS ARCHITECTURE

The Avtec PTP for Windows is a PC-based system running under the Windows NT/2000 operating system. The hardware architecture is shown in **Figure 1-1**. The PC contains PCI and

ISA I/O busses as well as the standard PC peripherals and application specific interfaces. The Avtec PTP for Windows software controls and monitors all aspects of the PC configuration.

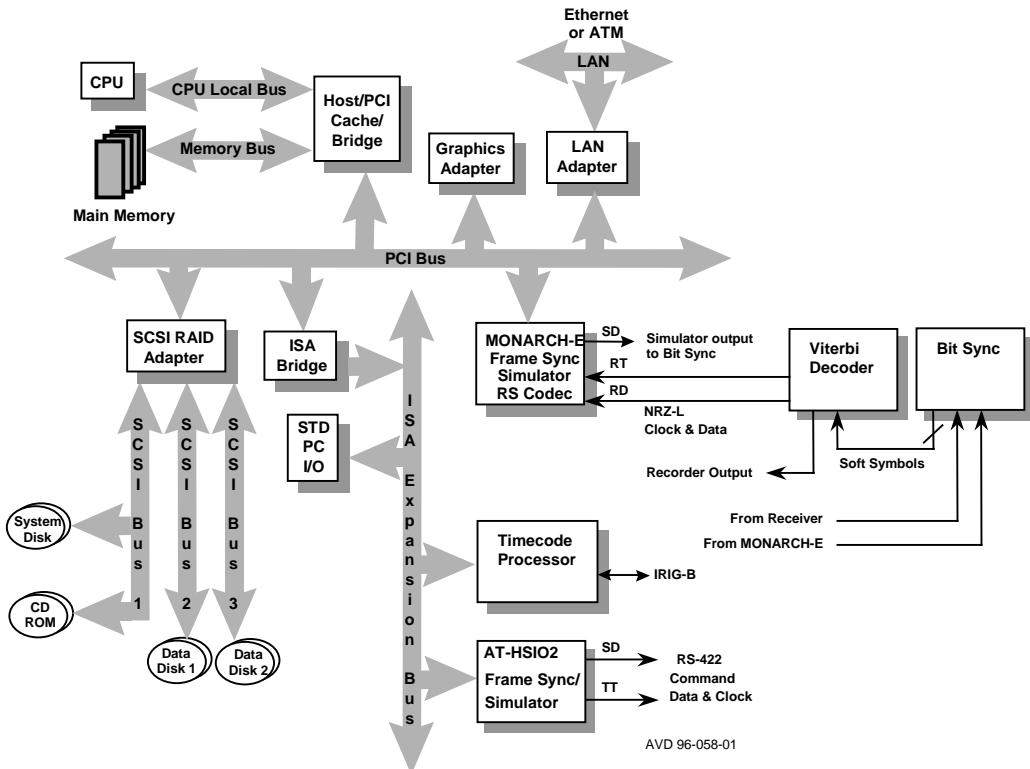


Figure 1-1: PC-based Front-end Processor Block Diagram

Standard PC interfaces supported by Avtec PTP for Windows include:

- Ethernet interface
- SCSI and IDE hard disks
- CD ROM drive
- COM Port
- LPT Port

Application-specific interfaces supported by the Avtec PTP for Windows include:

- AT-HSIO2 Frame Synchronizer/Telemetry Simulator
- MONARCH-E PCI Frame Synchronizer/Telemetry Simulator with Reed-Solomon Encoder/Decoder
- Bit Synchronizer (*Aydin PC 335 or Veda TSCN-300*) with Viterbi decoder
- Command BPSK Modulator (*Apogee ISA-PSK*)
- Tunable BPSK Modulator (*General Data Products PSK004*)

- Tunable Subcarrier BPSK Demodulator (*Aydin PC329*)
- IRIG-B and NASA-36 Time Code Processor (*Apogee ISA-STG2 or Odetics GPSync*)
- DataLynx Formatter
- DataLynx Receiver

The Avtec PTP for Windows acquires the telemetry data streams using the Frame Synchronizer portion of the AT-HSIO2 or MONARCH-E. Both the AT-HSIO2 and the MONARCH-E perform frame synchronization using an adaptive strategy, serial-to-parallel conversion, and time tagging. The MONARCH-E supports rates up to 25 Mbps with hardware support for derandomization, CRC, and Reed-Solomon decoding. The AT-HSIO2 supports data rates up to 2 Mbps. Both boards are more like network interface cards than traditional frame synchronizer/decom cards in that they provide a stream of frame data to the PC rather than individual raw measurements and tags. Because of this flexible stream architecture, the Avtec PTP for Windows can support both TDM framed telemetry as well as packetized telemetry.

The AT-HSIO2 and MONARCH-E are bi-directional boards which can also support Telemetry Simulation, Command Output, and Bit Error Rate Testing. The MONARCH-E Telemetry Simulator performs Reed-Solomon encoding, interleaving, CRC encoding, sync marker insertion, pseudo-randomization, and convolutional encoding at data rates up to 25 Mbps. The AT-HSIO2 outputs serial streams at up to 2 Mbps. The boards also support bit error rate testing and data quality monitoring for testing and pre-pass verification functions.

SECTION 3

THEORY OF OPERATION

AVTEC PTP FOR WINDOWS SOFTWARE

The Avtec PTP for Windows software consists of a Server application (PTP NT.EXE), a graphical user interface (CONSOLE.EXE), and the module dynamic link libraries (DLL). The Server controls the Avtec PTP for Windows hardware and performs all of the real-time data processing. The Console application provides an easy-to-use graphical interface for controlling the configuration and monitoring status of the Avtec PTP for Windows system.

The Console communicates with the Server through a TCP/IP socket connection. The Console can run locally on the Avtec PTP for Windows system or on a remote PC that is connected to the Avtec PTP for Windows system via a TCP/IP network. The Avtec PTP for Windows application can also be remotely controlled by a user application through the AV_PTP Remote Interface Library.

THE AVTEC PTP FOR WINDOWS DESKTOP

The user creates a data acquisition and processing solution using the Avtec PTP for Windows desktop. The Avtec PTP for Windows supports the acquisition, processing, and routing of multiple, simultaneous data streams using a combination of software modules, I/O boards, and PC peripherals. Data acquisition and processing is completely software configurable. The Avtec PTP for Windows operates four types of modules: Auxiliary I/O, Data I/O, Data Processing, and Encapsulation/Extraction. The user creates a desktop by loading a combination of modules to perform specific tasks.

Auxiliary I/O Modules control and monitor auxiliary I/O boards such as a Bit Synchronizer, a BPSK modulator, a BPSK demodulator, or a Time Code Processor. These modules do not transfer real-time data streams into or out of the system. Only control and status information is passed between the CPU and the auxiliary I/O board.

Data I/O Modules control the data I/O devices in the system, including network interfaces, file devices, and the AT-HSIO2 or MONARCH-E boards. The Data I/O Modules transfer real-time data streams into or out of the system using the application-specific interfaces and standard PC peripherals.

Data Processing Modules perform some software processing on data as it flows through the module. An example is the CCSDS Virtual Channel Processor (VCP) which filters frame data based on Virtual Channel ID (VCID) and data quality.

Encapsulation/Extraction Modules are data formatting modules that either create a data format or extract data from a format. In many cases, the modules provide spacecraft specific headers that can be added to a data unit.

AVTEC PTP FOR WINDOWS MODULE FUNCTIONS

Data I/O Modules and Data Processing Modules are “connected” together to perform real-time data processing tasks. The user selects the data flow between the various modules. Each Data Module has data inputs and outputs and an event input and output.

A module’s data output is connected to another module to pass data buffers from the first module to the second. For example, if the data output of the Serial Input Module is connected to the Recorder Module, the Serial Input Module will pass frame data from the I/O board to the Recorder Module each time it receives a frame on the serial data line. The data output of the Serial Input Module could also be connected to the Socket Module so that each frame of data is also transmitted to the network.

A module’s event output is connected to another module to signal an event to the second module. For example, the Serial Output Module provides an event output every time it completes writing a frame of data to the I/O board. This event output is used to tell another module to provide another buffer of data to the Serial Output Module. To playback a binary file to the serial output, the Playback Module’s data output is connected to the Serial Output Module and the Serial

Output Module's event output is connected to the Playback Module. The event connection provides a flow control between the two modules.

The following section describes how modules are used in a typical Avtec PTP for Windows application.

SECTION 4

FRONT-END TELEMETRY AND COMMAND PROCESSOR

The Avtec PTP for Windows' ability to input serial telemetry streams and route them in various ways makes it an ideal solution for front-end telemetry and command systems. In this application, the Avtec PTP for Windows acts as a gateway between a network and the spacecraft uplink/downlink. **Figure 1-2** shows the modules that are used in a front-end Telemetry and Command Processor. The underlying hardware architecture is shown in **Figure 1-1**.

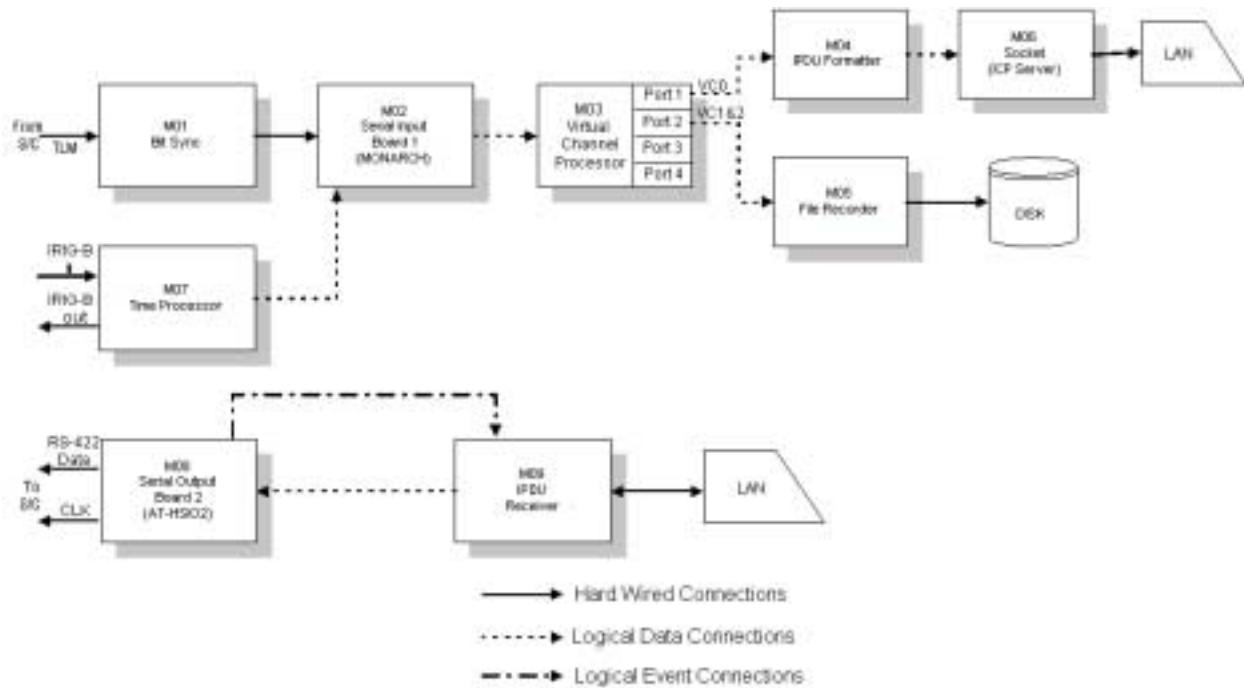


Figure 1-2: Module Connections for a Front-End Processor

The modules used for the telemetry data flow are listed below:

- Bit Synchronizer/Viterbi Decoder
- Serial Input
- Virtual Channel Processor

- Recorder
- Socket
- Delay
- ITCU Telemetry
- LEO-T TFDH Receiver
- SoftSync

The Avtec PTP for Windows accepts a serial telemetry stream using the Bit Synchronizer with Viterbi Decoder. The clock and data output from the Bit Synchronizer with Viterbi Decoder are wired to the MONARCH-E clock and data input. The MONARCH-E performs frame synchronization, derandomization, Reed-Solomon decoding, and time tagging. The Serial Input Module controls the operation of the MONARCH-E. The Serial Input Module's data output port is connected to the CCSDS VCP Module. The Serial Input Module passes telemetry frames with time tag and quality annotation to the CCSDS VCP module. The CCSDS VCP filters frames based on VCID and data quality. The VCP has multiple data output ports. Each data output port can be configured to output a particular subset of Virtual Channels. In this configuration, the VCP port 1 is configured to pass only frames with VCID 0 and is connected to the IPDU Formatter Module. The IPDU Formatter receives Virtual Channel Data Units (VCDU) from the VCP port 1 and encapsulates the VCDU within an IPDU header. The IPDU Formatter is connected to Socket Module that is configured as a TCP server. The Socket Module sends VCID 0 frames with IPDU header over the network in real time. The VCP port 2 is configured to pass only frames with VCID 1 or 2 and to discard frames with other VCIDs or with sequence errors. The VCP port 2 is connected to the Recorder Module. The VCID 1 and 2 frames output from VCP port 2 are sent to the Recorder module which stores the data to disk.

The modules used for the command data flow are listed below:

- IPDU Receiver
- Serial Output
- Command Verify
- ITCU Command
- Manual Command

The Avtec PTP for Windows accepts commands from the network using another instance of the IPDU Receiver. The IPDU Receiver is a special type of Socket module that supports IPDU encapsulated data. It is configured as a TCP server and accepts IPDU encapsulated Command Link Transmission Units (CLTU) from a network client. The IPDU Receiver data output is connected to the Serial Output Module. The IPDU receiver strips the IPDU header from accepted network packets and passes the command data to the Serial Output Module. The Serial Output module serializes the data using the AT-HSIO2 serial output. It is no longer necessary to set the Serial Output frame size to zero to support variable size command frames. With Avtec PTP for

Windows Version 1.49 and greater, the control flag can be set to PTP_OUT_COMMANDING_MODE which will accomplish the same thing.

Chapter 2

Remote Interface Library

SECTION 1

OVERVIEW

The Avtec PTP for Windows remote control and monitoring interface consists of two network sockets. A TCP/IP socket server provides a bi-directional byte stream connection between a single controller and the Avtec PTP for Windows system. An IP multicast socket is used for periodic status broadcast so that multiple host machines can receive from the multicast address to monitor the status of the Avtec PTP for Windows system.

AV_PTP is a functional interface between the user's application-level software and the Avtec PTP for Windows remote control socket interface. High level API calls remove the need for any knowledge of the underlying socket interface on the part of the application which communicates with the Avtec PTP for Windows system strictly on a function call basis. This not only reduces the burden on the programmer writing the application, but also enables the programmer to port the application to a different platform without rewriting application source code.

Figure 2-1 indicates the residence levels of the elements described above. The layers with no shading reside on the user's machine, and the *layers with shading* reside on the Avtec PTP for Windows system.

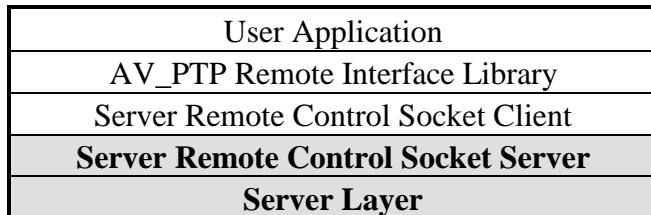


Figure 2-1: AV_PTP Development Environment

The AV_PTP remote interface functions are defined in the AV_PTP.H header file. This chapter contains a detailed description of each function contained in the AV_PTP Library as listed in **Table 2-1:**

Table 2-1: AV_PTP Library Functions

Routines	Description
ptp_DFSwap	Called to insure proper byte swapping of double float

Routines	Description
ptp_Fswap	Called to insure proper byte swapping for float
ptp_LLswap	Called to insure proper byte swapping of 64 bit int
ptp_Lswap	Called to insure proper byte swapping of long int
ptp_swap	Called to insure proper byte swapping of short int
ptpBeginMultiCastService*	Begins listening for Avtec PTP for Windows Server broadcasts
PtpCallFileCleanup^	Deletes files with user-specified attributes.
PtpCancelUserAlarmCallBackEx	Removes a user callback function.
PtpConnectModuleData	Connects the data output from one module port to the input of another module
PtpConnectModuleDataEx	Connects the data output from one module port to the input of another module
PtpConnectModuleEvent	Connects the event output from one module to the input of another module
PtpConnectModuleEventEx	Connects the event output from one module to the input of another module
PtpConnectToMonitor^	Establishes socket connection to an Avtec PTP for Windows Monitor
PtpConnectToServer	Establishes socket connection to an Avtec PTP for Windows Server
PtpCopyMulticastBuffer	Copy raw multicast status buffer.
PtpCopyResponseBuffer	Copy raw response message from last command sent
PtpCopyResponseBufferEx	Copy raw response message from last command sent
PtpCreateDefaultPathEx	Creates a new file system path
PtpCreateNewConfiguration	Creates a new Avtec PTP for Windows configuration (no modules loaded)
PtpCreateNewConfigurationEx	Creates a new Avtec PTP for Windows configuration (no modules loaded)
ptpCreateNewConfigurationAllServers	Creates a new Avtec PTP for Windows configuration (no modules loaded) on all connected servers
PtpDisableAllModules	Disables all loaded modules
PtpDisableAllModulesEx	Disables all loaded modules
ptpDisableAllModulesAllServers	Disables all loaded modules on all connected servers
PtpDisableModule	Disables a particular instance of a module
PtpDisableModuleEx	Disables a particular instance of a module
PtpDisablePingEx	Disable servers network connectivity test
PtpDisconnectFromMonitorEx^	Disconnect control socket from Avtec PTP for Windows Monitor
ptpDisconnectFromServer	Disconnect control socket from Avtec PTP for Windows Server
ptpDisconnectFromServerEx	Disconnect control socket from Avtec PTP for Windows Server
ptpDisconnectFromServerAllServers	Disconnect from all connected Servers
PtpEnableAllModules	Enables all loaded modules
ptpEnableAllModulesEx	Enables all loaded modules
ptpEnableAllModulesAllServers	Enables all loaded modules on all connected Servers

Routines	Description
PtpEnableKeepAliveEx	Enable or disable TCP keep alive for RIL messages.
PtpEnableModule	Enables a particular instance of a module
PtpEnableModuleEx	Enables a particular instance of a module
ptpEnableNameServer*	Enable name resolution for received multicast statuses
PtpEnablePingEx	Enable servers network connectivity test
ptpEnableSecureBufferMode	Causes Server to send unique copies of data buffers sent between modules
ptpEnableSecureBufferModeEx	Causes Server to send unique copies of data buffers sent between modules
PtpGetConfiguration	Retrieves the current Avtec PTP for Windows configuration
ptpGetConfigurationEx	Retrieves the current Avtec PTP for Windows configuration
PtpGetDefaultPathEx	Get the current file system path
PtpGetDesktopIdEx	Get the desktop and parent desktop id.
PtpGetDesktopName	Retrieve Avtec PTP for Windows Server desktop name
PtpGetDesktopNameEx	Retrieve Avtec PTP for Windows Server desktop name
PtpGetDesktopTLL	Get Time To Live for multicast status of all modules (default)
PtpGetDesktopTTLEx	Get Time To Live for multicast status of all modules (default)
PtpGetDriveSpace^	Get information about drive space.
PtpGetEnableStatus	Retrieve enable status of all modules
ptpGetEnableStatusEx	Retrieve enable status of all modules
PtpGetHex*	Converts keyboard hex input to long int
ptpGetKeepAliveStatusEx	Determine if TCP keep alive is enabled or disabled.
PtpGetLong*	Converts keyboard decimal input to long int
ptpGetModuleConnections	Future support
ptpGetModuleConnectionsEx	Future support
ptpGetModuleEnableStatus	Retrieve enable status of specified module
ptpGetModuleEnableStatusEx	Retrieve enable status of specified module
PtpGetModuleName	Retrieve specified module name
ptpGetModuleNameEx	Retrieve specified module name
PtpGetModuleState	Retrieves the current state (configuration) of a module
ptpGetModuleStateEx	Retrieves the current state of a module
PtpGetModuleTTL	Get Time To Live for multicast status of specific module
PtpGetModuleTTLEx	Returns current Time To Live of multicast status for selected module
ptpGetModuleVersion	Get the module version.
ptpGetModuleVersionEx	Get the module version.
PtpGetPingStateEx	Get enable state of servers network connectivity test
PtpGetResources^	Get the memory and system resource information.
PtpGetRILVersion	Get the current Remote Interface Library(RIL) version.

Routines	Description
ptpGetServerVersionEx	Get the current version of the Avtec PTP for Windows Server.
PtpGetStationId	Retrieves Avtec PTP for Windows Server station Id
ptpGetStationIdEx	Retrieves Avtec PTP for Windows Server station Id
ptpGetStationTime	Retrieve Avtec PTP for Windows Server PB4 time
ptpGetStationTimeEx	Retrieve Avtec PTP for Windows Server PB4 time
PtpGetText*	Receives keyboard input
ptpGetTimeStateEx	Gets the current configuration of the Apogee time board on Avtec PTP for Windows Server.
ptpGetTimeStatusEx	Get the time status.
ptpGetUnknownModuleStateEx	Same as ptpGetModuleStateEx except that structure size is returned
ptpHasDesktopChangedEx	See if Avtec PTP for Windows desktop has been modified by other inputs.
PtpHideServerEx	Hides or shows Avtec PTP for Windows Server window from the local users.
PtpKillProcess	Closes the program specified by ProgramName that is running on the Avtec PTP for Windows Server machine.
PtpKillScheduleThread^	Kill the thread started by the functions ptpBeginSchedule (under development) or ptsScheduleFileCleanup.
ptpListServers*	Retrieves a list of live Multicast Avtec PTP for Windows Servers
PtpLoadModule	Loads an instance of the specified module
PtpLoadModuleEx	Loads an instance of the specified module
PtpLoadProfile	Loads a previously saved configuration from the disk
PtpLoadProfileEx	Loads a previously saved configuration from the disk
PtpLoadProfileAllServers	Loads a previously saved configuration from the disk on all connected Servers
PtpNoOp	Places call to Avtec PTP for Windows Server to verify socket and Server still alive
PtpNoOpEx	Places call to Avtec PTP for Windows Server to verify socket and Server still alive
PtpNoOpAllServers	Places call to all connected Avtec PTP for Windows Servers to verify sockets and Servers are still alive
PtpPollCustomStatusEx	Retrieves custom status information.
PtpPollModuleState	Method to retrieve module state without incrementing command count
PtpPollModuleStateEx	Method to retrieve module state without incrementing command count
ptpPollUnknownModuleStateEx	Same as ptpPollModuleStateEx except that structure size is returned
PtpPostMessage*	Copy raw Avtec PTP for Windows command buffer to control socket
PtpPostMessageEx*	Copy raw Avtec PTP for Windows command buffer to control socket
PtpQueryAvailableModules	Retrieves a list of available modules
ptpQueryAvailableModulesEx	Retrieves a list of available modules
PtpQueryDirectory	Retrieves a list of directory files available

Routines	Description
ptpQueryDirectoryEx	Retrieves a list of directory files available
ptpQueryExistingLogs	Retrieves a list of log files available for playback
ptpQueryExistingLogsEx	Retrieves a list of log files available for playback
PtpQueryOpenWindows^	List all applications running on Server pc.
ptpQuerySavedProfiles	Retrieves a list of previously saved configurations
ptpQuerySavedProfilesEx	Retrieves a list of previously saved configurations
ptpReboot^	Shuts down all apps and reboots Server pc
PtpRegisterCallBack*	Registers local function address for multicast servers to call back
ptpRegisterCustomStatusEx	Request Avtec PTP for Windows Server to begin sending custom multicast status messages.
ptpRegisterUserAlarmCallBackEx	Add a user registered function to be called in the event of desktop changes
PtpRemoveModuleData	Remove data connection between two modules
ptpRemoveModuleDataEx	Remove data connection between two modules
ptpRemoveModuleEvent	Remove event connection between two modules
ptpRemoveModuleEventEx	Remove event connection between two modules
PtpSaveProfile	Saves the current Avtec PTP for Windows configuration to the disk
PtpSaveProfileEx	Saves the current Avtec PTP for Windows configuration to the disk
PtpScheduleFileCleanup^	Continuously call the function ptpCallFileCleanup at scheduled times.
ptpSendDesktopCommand	Future support
ptpSendDesktopCommandEx	Future support
ptpSendModuleCommand	Future support
ptpSendModuleCommandEx	Future support
PtpSetActivityDelay	Set time out of module Activity flags
ptpSetActivityDelayEx	Set time out of module Activity flags
PtpSetConfiguration	Set Avtec PTP for Windows configuration parameters
ptpSetConfigurationEx	Set Avtec PTP for Windows configuration parameters
PtpSetCustomStatusEx	Sets the module configuration via a custom status structure.
PtpSetDefaultPathEx	Sets the file system working path for data logging etc...
ptpSetDesktopTTL	Set Time To Live for multicast status messages from all modules in a desktop.
ptpSetDesktopTTLEx	Set Time To Live for multicast status of all modules (default)
ptpSetDesktopTTLAllServers	Set Time To Live for multicast status of all modules on all servers (default)
ptpSetModuleState	Sets a module's state (configuration)
ptpSetModuleStateEx	Sets a module's state
ptpSetModuleTTL	Set Time To Live for multicast status of specific module
ptpSetModuleTTLEx	Set Time To Live for multicast status of specific module
ptpSetMulticastAddress	Set multicast class D address (can also be set to class C)

Routines	Description
ptpSetMulticastAddressEx	Set multicast class D address (can also be set to class C)
ptpSetMulticastAddressAllServers	Set multicast class D address on all servers (can also be set to class C)
ptpSetRILSocketTimeOut	Sets network RIL socket timeout duration for all connections
ptpSetStationTimeEx	Sets the Apogee time board on Avtec PTP for Windows Server to the indicated time
ptpSetStatusDelay	Sets rate of Server multicast messages.
ptpSetStatusDelayEx	Sets rate of Server multicast messages.
ptpSetTimeStateEx	Sets the current configuration of the Apogee time board on Avtec PTP for Windows Server.
ptpSpawn^	Begins specified program on Server
ptpUnloadModule	Test and resynch Apogee time board
ptpUnloadModuleEx	Removes a particular instance of a module
ptpWinCls*	Clears window of text application
PtpWinGotoxy	Position text cursor in window of text application at location x, y.
ptpZeroAllModuleCounters	Zeroes the status counters for all loaded modules.
ptpZeroAllModuleCountersEx	Zeroes the status counters for all loaded modules.
ptpZeroAllModuleCountersAllServers	Zeroes the status counters for all loaded modules.
ptpZeroModuleCounters	Zeroes the status counters for the module instance specified by module.
ptpZeroModuleCountersEx	Zeroes the status counters for the module instance specified by module.

Functions marked with an * are only supported by the Win32 version of AV_PTP.

Functions marked with an ^ require that the Avtec PTP for Windows Monitor program is running on the Avtec PTP for Windows Server machine.

Ex function calls support multi-server connections whereas non ex functions support only one connection. The function ptpConnectToServer must be called before sending any other commands to the Avtec PTP for Windows. The PTP_RETURN value should be cast to the PTP_16 and used for the socket id parameter for the ex functions. Some ex functions do not have an non-ex counterpart. Ex function calls are available to single server users by passing the handle returned from the function ptpConnectToServer.

The ptp_swap functions only swap the byte order of the data for operating systems that have the macro BIG_ENDIAN defined. This allows programmers to use the same function calls regardless of the target platform. The macro BIG_ENDIAN is automatically defined in the Solaris Avtec PTP for Windows Remote Interface Library (av_ptp.a) since this is a BIG_ENDIAN operating system. It is not defined in the NT/2000 Avtec PTP for Windows Remote Interface Library (Av_PTP.dll) since this is a little endian machine.

The module handle is always its position in the desktop. Since there can be no more than 32 modules, PTP_8 is big enough to contain the module handle. The PTP_Config structure defines

module handle as a PTP_8 which is used in the functions ptpNetConfiguration, ptpNetConfiguration Ex, ptpNetGetConfiguration, and ptpNetGetConfigurationEx. Other functions define module handle as a PTP_32 but only 1 byte of the 4 is used.

SECTION 2

ERROR RESPONSES

Each AV_PTP function returns one of the following error responses if an error occurs.

/* Error Responses */	
#define PTP_GENERICERROR	-1
#define PTP_NOTCONNECTED	-7
#define PTP_CONNECTIONFAILED	-8
#define PTP_ALREADYCONNECTED	-9
#define PTP_INVALIDCOMMANDTYPEERR	-10
#define PTP_INVALIDFUNCTIONCALLERR	-10
#define PTP_INVALIDMODULEHANDLEERR	-11
#define PTP_MODULEHASNOREMOTECONTROLERR	-12
#define PTP_FILENOFOUNDERR	-13
#define PTP_TOOMANYMODULESERR	-14
#define PTP_INVLAIMODULEERR	-15
#define PTP_MODULEHASNOSTATUS	-16
#define PTP_MEMCOPYERR	-17
#define PTP_CRCERR	-18
#define PTP_MALLOC_ERR	-19
#define PTP_SERVERCRCERR	-20

The error PTP_NOTCONNECTED will be returned if the Avtec PTP for Windows was not connected before the command was sent. On the other hand, the error PTP_CONNECTIONFAILED will be returned if the Avtec PTP for Windows lost its connection while the command was being sent.

The error PTP_FILENOFOUNDERR will be returned if the file or the directory was not found.

The errors PTP_INVALIDCOMMANDTYPEERR and PTP_INVALIDFUNCTIONCALLERR both are equivalent in meaning and numeric value. There is a distinction only to support legacy code.

SECTION 3

DATA TYPES

The AV_PTP library is supported on both Win32 and Solaris platforms. In order to avoid platform dependent type definitions, AV_PTP uses the following types in the definition of the AV_PTP functions and structures.

```
/*********************************************
/* General AV_PTP definitions
*/
/********************************************

#ifndef _WIN32
#define HANDLE void*
#endif

#define PTP_INT          signed long      /* 32-bit signed integer */
#define PTP_SHORT        signed short     /* 16-bit signed integer */
#define PTP_CHAR         signed char      /* 8-bit signed integer */
#define PTP_RETURN        signed long      /* 32-bit signed integer */
#define PTP_LONG          signed long      /* 32-bit signed integer */
#define PTP_32            unsigned long     /* 32-bit unsigned integer */
#define PTP_16            unsigned short    /* 16-bit unsigned integer */
#define PTP_8             unsigned char     /* 8-bit unsigned integer */
#define PTP_FLOAT          float           /* 32-bit float */
#define PTP_DFLOAT         double          /* 64-bit float */
#define PTP_BOOL           int             /* */

#ifdef _WIN32
#define PTP_64            ULONGLONG       /* 64-bit unsigned integer */
#define PTP_LRETURN        LONGLONG        /* 64-bit signed integer */
#else
#define PTP_64            unsigned long long /* 64-bit unsigned integer */
#define PTP_LRETURN        signed long long /* 64-bit signed integer */
#define BOOL              PTP_BOOL
#define UCHAR             PTP_8
#define USHORT            PTP_16
#define WORD              PTP_16
#define ULONG             PTP_32
#define DWORD             PTP_32
#define LPVOID            PTP_32
#define ULONGLONG         PTP_64
#endif
#endif
```

SECTION 4

BYTE ALIGNMENT

The Avtec PTP for Windows remote interface library must be compiled with a default byte packing of 8 bytes. If a different packing is used, then member variables can be misaligned.

Borland is one compiler that defaults to a byte packing of 4 bytes. This can cause structures that contain the PTP_64 members to be misaligned if the structure has not been manually padded to ensure that all PTP_64 are on 8 byte boundaries. A Borland compiler setting can be changed on the compiler to force a default packing of 8 bytes. Avtec has found that the cc, gcc, and Visual C++ compilers have a default packing of 8 bytes.

Please contact Avtec technical support if you think you are experiencing a byte alignment problem. A byte alignment mismatch can cause the setting of one member variable of a structure to cause a different member variable to be changed of that same structure. This problem can occur if the member variable being set comes anywhere after a member variable that has a size greater than the default packing of that compiler. An example of this situation would be a member variable that comes several variables after a variable with a size of 8 bytes using a compiler that has a default packing of 4 bytes. The 8 byte member would be 4 byte aligned with this compiler which may or may not also be 8 byte aligned like the Avtec PTP for Windows library.

Avtec provides a utility test program (ByteAlignment.c) that can be compiled and run on any platform using any compiler. This program will determine the default byte alignment of the compiler and will warn if it is different than 8 bytes.

SECTION 5

FUNCTION DETAILS

PTP_DFSWAP

Syntax

```
PTP_DFLOAT ptp_DFSwap(PTP_DFLOAT source)
```

Purpose

Prepares 64 bit float for transmission over socket connection.

Return Value

Returns converted 64 bit float value.

Discussion

The user calls this function to prepare 64 bit float for transmission across socket connections.

PTP_FSWAP

Syntax

```
PTP_FLOAT ptp_Fswap(PTP_FLOAT source)
```

Purpose

Prepares 32 bit float for transmission over socket connection.

Return Value

Returns converted 32 bit float value.

Discussion

The user calls this function to prepare 32 bit float for transmission across socket connections.

PTP_LL_SWAP

Syntax

```
PTP_64 ptp_LLswap(PTP_64 source)
```

Purpose

Prepares 64 bit long integers for transmission over socket connection.

Return Value

Returns converted 64 bit long integer value.

Discussion

The user calls this function to prepare 64 bit long integers for transmission across socket connections.

PTP_LSWAP***Syntax***

```
PTP_32 ptp_Lswap(PTP_32 source)
```

Purpose

Prepares long integers for transmission over socket connection.

Return Value

Returns converted long integer value.

Discussion

The user calls this function to prepare long integers for transmission across socket connections.

PTP_SWAP***Syntax***

```
PTP_16 ptp_swap(PTP_16 source)
```

Purpose

Prepares short integers for transmission over socket connection.

Return Value

Returns converted short integer value.

Discussion

The user calls this function to prepare short integers for transmission across socket connections.

PTPBEGINMULTICASTSERVICE***Syntax***

```
PTP_RETURN ptpBeginMultiCastService(char *Address, PTP_32 Port)
```

Purpose

Activates the function to begin listening for Avtec PTP for Windows broadcasts on address “Address” and port number “Port.” Defaults to address 233.3.3.3 and port 3000.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

The user can define which multicast address to listen for and which port to listen on. Setting “Address” to NULL and “Port” to 0 loads default values. Call to this function executes a thread that maintains a table of Avtec PTP for Windows Server addresses and their control port IDs that are alive on the network. Use **ptpListServers** to get a listing of the Avtec PTP for Windows Server table. This function is only supported for Win32.

PTPCALLFILECLEANUP**Syntax**

```
PTP_RETURN ptpCallFileCleanup(char *ptpAddress,
                           PTP_MONITOR_FILECLEANUPVARS *InfoInOut,
                           int size)
```

Purpose

Connects to **ptpMonitor** at **ptpAddress** and does an immediate delete of files based on **InfoInOut**. Size should be set to sizeof(**InfoInOut**).

The following input parameters are required:

- **ptpAddress** is text, i.e. "127.0.0.1".
- **InfoInOut.FilterPath** defines the path to search, for example C:\ptp_user\mypath\
- **InfoInOut.Filter** defines the files to perform the cleanup to, for example *.Extension or *.*
- **InfoInOut.AgeInSecs** delete files that are older than this time in second. For example, 3600 (60*60) for 1 hour or older, or 604800 (60*60*24*7) for 1 week or older.

Parameters

```
typedef struct _PTP_MONITOR_FILECLEANUPVARS
{
    PTP_MONITOR_TIMERVARS TimerVars;
    char                  FilterPath[256];
    char                  Filter[256];
    long                 AgeInSecs;
    UCHAR                Pad[32];
} PTP_MONITOR_FILECLEANUPVARS;

typedef struct _PTP_MONITOR_TIMERVARS
{
    char      TimerName[256];
    HANDLE   hTimer;
```

```

PTP_64    fTime;
ULONG     TimeOut;
BOOL      ThreadRunning;
BOOL      ThreadEnable;
BOOL      RunOnce;
UCHAR     Pad[32];
} PTP_MONITOR_TIMERVARS;

```

Return Value

A nonnegative value indicates success and the number of files deleted. A negative value indicates an error.

PTPCANCELUSERALARMCALLBACKEX***Syntax***

```
PTP_RETURN ptuCancUserAlarmCallBackEx(PTP_16 Id)
```

Purpose

Cancels a user callback function.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

See **ptpRegisterUserAlarmCallBackEx** and **ptpHasDesktopChangedEx**.

PTPCONNECTMODULEDATA**PTPCONNECTMODULEDATAEX*****Syntax***

```

PTP_RETURN ptpConnectModuleData(PTP_32 sourceModule,
                               PTP_32 destinationModule,
                               PTP_32 portIndex,
                               PTP_32 reserved)

PTP_RETURN ptpConnectModuleDataEx(PTP_16 Id,
                                 PTP_32 sourceModule,
                                 PTP_32 destinationModule,
                                 PTP_32 portIndex,
                                 PTP_32 reserved)

```

Purpose

Connects the data output from **sourceModule** to the input of **destinationModule**. The **portIndex** specifies which source module output port for modules that provide multiple output ports.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

The source module will post a buffer of data to the destination module. A single source module can be connected to multiple destination modules.

PTPCONNECTMODULEEVENT**PTPCONNECTMODULEEVENTEX*****Syntax***

```
PTP_RETURN ptpNetModuleEvent( PTP_32 sourceModule,
                             PTP_32 destinationModule,
                             PTP_32 portIndex,
                             PTP_32 reserved)

PTP_RETURN ptpNetModuleEventEx( PTP_16 Id,
                             PTP_32 sourceModule,
                             PTP_32 destinationModule,
                             PTP_32 portIndex,
                             PTP_32 reserved)
```

Purpose

Connects the event output from **sourceModule** to the input of **destinationModule**. The **portIndex** specifies which source module output port for modules that provide multiple output event ports.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

The source module will post events to the destination module. A single source module can be connected to multiple destination modules. The definition of the event is module specific. For a serial output module, the event indicates that a buffer of data has been transmitted.

PTPCONNECTTOMONITOR***Syntax***

```
PTP_RETURN ptpNetToMonitor(const char *ptpAddress)
```

Purpose

Typically monitor calls connect to and disconnect from the Avtec PTP for Windows Monitor for every command. This call will hold the monitor connection open.

Return Value

Returns socket file descriptor if successful, negative otherwise.

Discussion

The user must call this function to open a TCP/IP connection to an Avtec PTP for Windows Monitor. The PTP_RETURN value should be cast to a PTP_16 and used for the Id parameter for AV_PTP **ptpDisconnectFromMonitorEx()**.

PTPCONNECTTOSERVER

Syntax

```
PTP_RETURN ptpConnectToServer(const char *serverIPAddress, PTP_32 serverIPPort)
```

Purpose

Connect to server via socket interface.

Return Value

Returns socket file descriptor if successful, negative otherwise.

Discussion

The user must call this function to open a TCP/IP connection to an Avtec PTP for Windows Server. **ptpConnectToServer** must be called before sending any other commands to the PTP. The PTP_RETURN value should be cast to a PTP_16 and used for the Id parameter for AV_PTP multi-server functions (Ex functions).

PTPCOPYMULTICASTBUFFER

Syntax

```
PTP_RETURN ptpCopyMulticastBuffer(char *destBuffer, PTP_32 destBufferSize)
```

Purpose

Copy raw multicast status buffer. This function is used with **ptpRegisterCallBack**.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

This function is only supported for Win32.

PTPCOPYRESPONSEBUFFER**PTPCOPYRESPONSEBUFFEREX*****Syntax***

```
PTP_RETURN ptuCpyResponseBuffer(char *destBuffer, PTP_32 destBufferSize)
PTP_RETURN ptuCpyResponseBufferEx(PTP_16 Id, char *destBuffer, PTP_32 destBufferSize)
```

Purpose

Copy raw response message from last command sent to Avtec PTP for Windows Server into local variable.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

Allows user to access data fields within the returned Avtec PTP for Windows socket header. This function is only supported for Win32.

PTPCREATEDEFAULTPATHEX***Syntax***

```
PTP_RETURN ptuCrtDefaultPathEx(PTP_16 Id, char *NewPath)
```

Purpose

Create a file system ‘default working path’ for the specified server.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

Every server may have a different working path.

The working path is used by newly loaded modules to preset file names.

The working path is also used by the server while loading and saving desktops.

The default Avtec PTP for Windows directory structure will be added to the user provided NewPath.

\|Host\c\MyPTPStuff results in \|Host\c\MyPTPStuff\Ptp_User\Desktops,
\|Host\c\MyPTPStuff\Ptp_User\Logs etc...

See **ptpSetDefaultPathEx** and **ptpGetDefaultPathEx**.

PTPCREATENEWCONFIGURATION**PTPCREATENEWCONFIGURATIONEx****PTPCREATENEWCONFIGURATIONALLSOURCES****Syntax**

```
PTP_RETURN ptpCreateNewConfiguration(void)
PTP_RETURN ptpCreateNewConfigurationEx(PTP_16 Id)
PTP_RETURN ptpCreateNewConfigurationAllServers(void)
```

Purpose

Clears the current Avtec PTP for Windows configuration.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

ptpCreateNewConfiguration must be called before building an Avtec PTP for Windows configuration. This function is the same as a Console New Desktop.

PTPDISABLEALLMODULES**PTPDISABLEALLMODULESEX****PTPDISABLEALLMODULESALLSOURCES****Syntax**

```
PTP_RETURN ptpDisableAllModules(void)
PTP_RETURN ptpDisableAllModulesEx(PTP_16 Id)
PTP_RETURN ptpDisableAllModulesAllServers(void)
```

Purpose

Disables all loaded modules. Stops the flow of data through all modules in the system. Recommended prior to calling **ptpCreateNewConfiguration** if the value returned from **ptpGetEnableStatus** is positive.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPDISABLEMODULE**PTPDISABLEMODULEEx*****Syntax***

```
PTP_RETURN ptpNetModule(PTP_32 module)
PTP_RETURN ptpNetModuleEx(PTP_16 Id, PTP_32 module)
```

Purpose

Disables the instance of a module referred to by the parameter **module**.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPDISABLEPINGEx***Syntax***

```
PTP_RETURN ptpNetPingEx(PTP_16 Id)
```

Purpose

Disables the network assurance ping that is sent by any open TCP receiver socket within the Avtec PTP for Windows.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

The network assurance ping is used to make sure that the remote side of a TCP connection is still accessible on the network. If the ping fails for a given socket, then Avtec PTP for Windows will try to re-establish the connection. The network assurance ping will not work on networks where ICMP messages are blocked by a firewall so it should be disabled. The network assurance ping is enabled by default.

PTPDISCONNECTFROMMONITOREx***Syntax***

```
void ptpNetDisconnectFromMonitorEx(PTP_16 Id)
```

Purpose

Close control socket, created with **ptpNetConnectToMonitor**, of Avtec PTP for Windows Monitor.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPDISCONNECTFROMSERVER**PTPDISCONNECTFROMSERVEREx****PTPDISCONNECTFROMSERVERALLSOURCES*****Syntax***

```
void ptpDisconnectFromServer(void)
void ptpDisconnectFromServerEx(PTP_16 Id)
void ptpDisconnectFromServerAllServers(void)
```

Purpose

Close control socket, created with **ptpConnectToServer**, of Avtec PTP for Windows Server.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPENABLEALLMODULES**PTPENABLEALLMODULESEX****PTPENABLEALLMODULESALLSOURCES*****Syntax***

```
PTP_RETURN ptpEnableAllModules(void)
PTP_RETURN ptpEnableAllModulesEx(PTP_16 Id)
PTP_RETURN ptpEnableAllModulesAllServers(void)
```

Purpose

Enables all the loaded modules. Starts the flow of data through the modules.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPENABLEKEEPALIVEEx***Syntax***

```
PTP_RETURN ptpEnableKeepAliveEx(PTP_16 Id, bool Flag)
```

Purpose

Enables TCP keep alive for the Remote Interface Library (RIL) messages if **Flag** is true. Disables TCP keep alive for RIL messages if **Flag** is false. Keep alive attributes are controlled by the Windows registry. TCP keep alive must be available for both PTPServer and the RIL for this function to succeed.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPENABLEMODULE**PTPENABLEMODULEEx****Syntax**

```
PTP_RETURN ptpNetableModule(PTP_32 module)
PTP_RETURN ptpNetableModuleEx(PTP_16 Id, PTP_32 module)
```

Purpose

Enables the instance of a module referred to by the parameter **module**. Starts the flow of data through the module.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPENABLENAMESERVER**Syntax**

```
void ptpNetableNameServer(PTP_8 EnableFlag)
```

Purpose

Enable name resolution for received multicast statuses.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPENABLEPINGEX**Syntax**

```
PTP_RETURN ptpNetablePingEx(PTP_16 Id)
```

Purpose

Enables the network assurance ping that is sent by any open TCP receiver socket within the Avtec PTP for Windows.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

The network assurance ping is used to make sure that the remote side of a TCP connection is still accessible on the network. If the ping fails for a given socket, then Avtec PTP for Windows will try to re-establish the connection. The network assurance ping will not work on networks where ICMP messages are blocked by a firewall and it should be disabled. The network assurance ping is enabled by default.

PTPENABLESECUREBUFFERMODE**PTPENABLESECUREBUFFERMODEEx****Syntax**

```
PTP_RETURN ptpNetableSecureBufferMode(unsigned char Flag)
PTP_RETURN ptpNetableSecureBufferModeEx(PTP_16 Id, unsigned char Flag)
```

Purpose

Enables secure buffer mode operation in Avtec PTP for Windows.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

When Avtec PTP for Windows passes data buffers from one module to another, it actually sends a “pointer” to the buffer. If the same data port is connected to several modules, all modules will receive a “pointer” to the same buffer. This can be a problem if any one of these modules is going to alter the data in any way (for example, by adding or removing a header). With secure buffer mode enabled, Avtec PTP for Windows makes a copy of the data buffer before sending it to a module. Each module receives a “pointer” to a unique buffer that contains the same data as the original.

Buffer copying has been automated in Avtec PTP for Windows Version 1.49 and higher. This function no longer needs to be called but is still valid.

PTPGETCONFIGURATION**PTPGETCONFIGURATIONEX****Syntax**

```
PTP_RETURN ptpNetConfiguration(PTP_CONFIG *config, PTP_32 configSize)
PTP_RETURN ptpNetConfigurationEx(PTP_16 Id, PTP_CONFIG *config, PTP_32 configSize)
```

Purpose

Retrieves the current Avtec PTP for Windows configuration.

Parameters

```
#define PTP_MAX_MODULES 32
#define PTP_MAX_MODULE_EVENTCONNECTIONS 32
#define PTP_MAX_MODULE_OUTCONNECTIONS PTP_MAX_MODULE_EVENTCONNECTIONS
#define PTP_MAX_MODULE_INCONNECTIONS 32

typedef struct PTPConfig
{
    PTP_8 moduleHandle[PTP_MAX_MODULES]; /* array of module handles */
    PTP_32 moduleType[PTP_MAX_MODULES]; /*moduleType for each moduleHandle */
    PTP_8 resources[PTP_MAX_MODULES];
    PTP_32 dataConnections[PTP_MAX_MODULES][PTP_MAX_MODULE_OUTCONNECTIONS];
    PTP_32 eventConnections[PTP_MAX_MODULES][PTP_MAX_MODULE_OUTCONNECTIONS];
} PTP_CONFIG;
```

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

This function can be used to determine which modules are loaded in a desktop and how they are interconnected. It is often used in monitor and control applications where the Avtec PTP for Windows configuration is loaded from previously saved desktop files.

PTPGETDEFAULTPATHEX**Syntax**

```
PTP_RETURN ptpNetDefaultPathEx(PTP_16 Id, char *NewPath, PTP_32 NewPathSize)
```

Purpose

Query the file system ‘default working path’ for the specified server.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

See **ptpNetDefaultPathEx** and **ptpNetCreateDefaultPathEx**.

PTPGETDESKTOPIDEx**Syntax**

```
PTP_RETURN ptpNetDesktopIdEx(PTP_16 Id, ULONG *DesktopId, ULONG *ParentDesktopId)
```

Purpose

Get the desktop id and the parent desktop id. When a desktop is first saved it is given a ParentDesktopId. Every time a desktop is saved it is given a new DesktopId. The Id's are statistically unique since they are a UTC time stamp.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETDESKTOPNAME**PTPGETDESKTOPNAMEEx****Syntax**

```
PTP_RETURN ptpNetDesktopName(char *DesktopName, PTP_32 nameSize)
PTP_RETURN ptpNetDesktopNameEx(PTP_16 Id, char *DesktopName, PTP_32 nameSize)
```

Purpose

Retrieve Avtec PTP for Windows Server desktop name.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETDESKTOPTTL**PTPGETDESKTOPTTLEx****Syntax**

```
PTP_RETURN ptpNetDesktopTTL()
PTP_RETURN ptpNetDesktopTTLEx(PTP_16 Id)
```

Purpose

Gets the time-to-live (TTL) setting for the Avtec PTP for Windows multicast status messages for the current desktop. The TTL setting controls how many hops the message will be routed. The default is 1.

Return Value

Returns the nonnegative TTL if successful, negative otherwise.

PTPGETDRIVESPACE**Syntax**

```
PTP_RETURN ptpNetDriveSpace(char *ptpAddress,
                           char *Path,
                           PTP_MONITOR_DISKSPACE *InfoDst,
                           int DstSize)
```

Purpose

Get the drive space information.

The following input parameters are required:

- **ptpAddress** is the address that the Avtec PTP for Windows Monitor will connect to.
- **Path** defines the drive(or directory if quota management is enabled) to query. ex. c:\
- **InfoDst** is a pointer the user PTP_MONITOR_DISKSPACE structure.
- **DstSize** should be sizeof(**InfoDst**).

The following output parameters are returned:

- **InfoDst.DirectoryName** will return Path.
- **InfoDst.lpFreeBytesAvailableToCaller** will return the space available to Avtec PTP for Windows.(quota management)
- **InfoDst.lpTotalNumberOfBytes** will return the total size of the selected drive.
- **InfoDst.lpTotalNumberOfFreeBytes** will return the maximum unallocated disk space for the selected drive.

Parameters

```
typedef struct _PTP_MONITOR_DISKSPACE
{
    char   DirectoryName[256];
    PTP_64  lpFreeBytesAvailableToCaller;
    PTP_64  lpTotalNumberOfBytes;
    PTP_64  lpTotalNumberOfFreeBytes;
    PTP_8   Pad[32];
} PTP_MONITOR_DISKSPACE;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETENABLESTATUS**PTPGETENABLESTATUSEx****Syntax**

```
PTP_LRETURN ptpNetEnableStatus(void)
```

```
PTP_LRETURN ptpGetEnableStatusEx(PTP_16 Id)
```

Purpose

Retrieves the current enable status of all modules. Bit mapped 64 bit return value, a 1 indicates the corresponding module is enabled. Bit 0 corresponds to Module 1, bit 1 to Module 2, etc.

Return Value

Returns **64 bit** nonnegative value if successful, negative otherwise.

PTPGETHEX

Syntax

```
PTP_32 ptpGetHex(char *Text)
```

Purpose

Parses hex format keyboard input and converts input to long int.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETKEEPALIVESTATUSEX

Syntax

```
PTP_RETURN ptpGetKeepAliveStatusEx(PTP_16 Id)
```

Purpose

Determine if TCP keep alive is enabled (1) or disabled (0).

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETLONG

Syntax

```
PTP_32 ptpGetLong(char *Text)
```

Purpose

Parses decimal format keyboard input and converts input to long int.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETMODULECONNECTIONS**PTPGETMODULECONNECTIONSEX****Syntax**

```
PTP_RETURN ptpGetModuleConnections(PTP_32 module,
                                  PTP_32 *moduleType,
                                  PTP_MODULEConfig *moduleConfig,
                                  PTP_32 moduleConfigSize)

PTP_RETURN ptpGetModuleConnectionsEx(PTP_16 Id,
                                    PTP_32 module,
                                    PTP_32 *moduleType,
                                    PTP_MODULEConfig *moduleConfig,
                                    PTP_32 moduleConfigSize)
```

Purpose

Retrieves the data and event connections for the instance of a module referred to by the parameter **module**.

Parameters

```
#define PTP_MAX_MODULE_EVENTCONNECTIONS 32
#define PTP_MAX_MODULE_OUTCONNECTIONS    PTP_MAX_MODULE_EVENTCONNECTIONS

typedef struct _PTP_MODULEConfig
{
    PTP_32 dataConnections[PTP_MAX_MODULE_OUTCONNECTIONS];
    PTP_32 eventConnections[PTP_MAX_MODULE_EVENTCONNECTIONS];
} PTP_MODULEConfig;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETMODULEENABLESTATUS**PTPGETMODULEENABLESTATUSEX****Syntax**

```
PTP_RETURN ptpGetModuleEnableStatus(PTP_32 Module)
PTP_RETURN ptpGetModuleEnableStatusEx(PTP_16 Id, PTP_32 Module)
```

Purpose

Retrieves the current enable status of a specific module.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

Return value of zero indicates module is disabled. Return value greater than zero indicates module enabled.

PTPGETMODULENAME**PTPGETMODULENAMEEX*****Syntax***

```
PTP_RETURN ptpGetModuleName(PTP_32 moduleHandle, char *moduleName, PTP_32 nameSize)
PTP_RETURN ptpGetModuleNameEx(PTP_16 Id,
                           PTP_32 moduleHandle,
                           char *moduleName,
                           PTP_32 nameSize)
```

Purpose

Retrieve specific module name.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETMODULESTATE**PTPGETMODULESTATEEX*****Syntax***

```
PTP_RETURN ptpGetModuleState(PTP_32 moduleHandle, char *params, PTP_32 paramsSize)
PTP_RETURN ptpGetModuleStateEx(PTP_16 Id,
                           PTP_32 moduleHandle,
                           char *params,
                           PTP_32 paramsSize)
```

Purpose

Retrieves the current values of the module state structure for the instance of a module referred to by the handle argument **moduleHandle**. The module handle is returned by the **ptpLoadModule** or **ptpLoadModuleEx** functions. Pass in the entire module structure to the argument **params**. The argument **paramsSize** must be equal to the size of the entire module structure. The current parameters are copied into the structure pointed to by **params**.

Return Value

Returns the module type (nonnegative) if successful, negative otherwise. The module type can be referred to by the #define or the hex value.

PTPGETMODULETLL**PTPGETMODULETTLEX****Syntax**

```
PTP_RETURN ptpNetModuleTTL(PTP_32 moduleHandle, PTP_32 moduleType)
PTP_RETURN ptpNetModuleTTLEx(PTP_16 Id, PTP_32 moduleHandle, PTP_32 moduleType)
```

Purpose

Gets the time-to-live (TTL) setting for the Avtec PTP for Windows multicast status messages for the instance of a module referred to by the handle argument **module**. The TTL setting controls how many hops the message will be routed. The default is 1.

Return Value

Returns the nonnegative TTL if successful, negative otherwise.

PTPGETMODULEVERSION**PTPGETMODULEVERSIONEX****Syntax**

```
PTP_RETURN ptpNetModuleVersion(PTP_32 moduleHandle,
                               PTP_MODULE_ABOUTSTRUCT *Dest,
                               PTP_32 DestSize)

PTP_RETURN ptpNetModuleVersionEx(PTP_16 Id,
                                 PTP_32 moduleHandle,
                                 PTP_MODULE_ABOUTSTRUCT *Dest,
                                 PTP_32 DestSize)
```

Purpose

Get the module version by filling out the PTP_MODULE_ABOUTSTRUCT structure. Note, these functions are not yet implemented in every module.

Parameters

```
typedef struct _PTP_MODULE_ABOUTSTRUCT
{
    PTP_32 Version;
    PTP_64 BuildDate;
    PTP_8  BuildDateText[16];
    PTP_8  Name[256];
} PTP_MODULE_ABOUTSTRUCT;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETPINGSTATEEX

Syntax

```
PTP_RETURN ptpGetPingStateEx(PTP_16 Id)
```

Purpose

Enables the network assurance ping that is sent by any open TCP receiver socket within the Avtec PTP for Windows.

Return Value

Returns nonnegative if successful, negative otherwise. A return value of 1 indicates that the Avtec PTP for Windows network assurance ping is enabled. A return value of 0 indicates that it is disabled.

Discussion

The network assurance ping is used to make sure that the remote side of a TCP connection is still accessible on the network. If the ping fails for a given socket, then Avtec PTP for Windows will try to re-establish the connection. The network assurance ping will not work on networks where ICMP messages are blocked by a firewall and it should be disabled. The network assurance ping is enabled by default.

PTPGETRESOURCES

Syntax

```
PTP_RETURN ptpGetResources(char *ptpAddress, PTP_MONITOR_INFO1 *InfoDst, int DstSize)
```

Purpose

Get the resource information and place it in the PTP_MONITOR_INFO1 structure.

The following input parameters are required:

- **ptpAddress** is the address to connects the Avtec PTP for Windows Monitor to.
- **InfoDst** is a pointer the user PTP_MONITOR_INFO1 structure.
- **DstSize** should be sizeof(**InfoDst**).

The following output parameters are returned:

- **InfoDst.AvailableDrives** is a bit map of available drives.
- **InfoDst.MemoryStatus.dwLength** is the size in bytes of the MEMORYSTATUS structure.
- **InfoDst.MemoryStatus.dwMemoryLoad** is the percent of memory in use.
- **InfoDst.MemoryStatus.dwTotalPhys** is the number of bytes of physical memory.

- **InfoDst.MemoryStatus.dwAvailPhys** is the number of bytes of free physical memory.
- **InfoDst.MemoryStatus.dwTotalPageFile** is the number of bytes of the paging file.
- **InfoDst.MemoryStatus.dwAvailPageFile** is the number of free bytes of the paging file.
- **InfoDst.MemoryStatus.dwTotalVirtual** is the number of user bytes of address space.
- **InfoDst.MemoryStatus.dwAvailVirtual** is the number of free user bytes.

Parameters

```

typedef struct _PTP_SYSTEM_INFO
{
    union
    {
        DWORD dwOemId;           // Obsolete field...do not use
        struct
        {
            WORD wProcessorArchitecture;
            WORD wReserved;
        };
    };
    DWORD dwPageSize;
    LPVOID lpMinimumApplicationAddress;
    LPVOID lpMaximumApplicationAddress;
    DWORD dwActiveProcessorMask;
    DWORD dwNumberOfProcessors;
    DWORD dwProcessorType;
    DWORD dwAllocationGranularity;
    WORD wProcessorLevel;
    WORD wProcessorRevision;
} PTP_SYSTEM_INFO, *LPPTP_SYSTEM_INFO;

typedef struct _PTP_MEMORYSTATUS
{
    DWORD dwLength;
    DWORD dwMemoryLoad;
    DWORD dwTotalPhys;
    DWORD dwAvailPhys;
    DWORD dwTotalPageFile;
    DWORD dwAvailPageFile;
    DWORD dwTotalVirtual;
    DWORD dwAvailVirtual;
} PTP_MEMORYSTATUS, *LPPTP_MEMORYSTATUS;

typedef struct _PTP_MONITOR_INFO1
{
    DWORD AvailableDrives;
    PTP_SYSTEM_INFO SystemInfo;
    PTP_MEMORYSTATUS MemoryStatus;
    UCHAR Pad[32];
} PTP_MONITOR_INFO1;

```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETRILVERSION

Syntax

```
PTP_RETURN ptpGetRILVersion()
```

Purpose

Get the current version of the Remote Interface Library (RIL) as a long integer. For example, 149 is returned to indicate version 1.49.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETSERVERVERSIONEX**Syntax**

```
PTP_RETURN ptpNetServerVersionEx(PTP_16 Id)
```

Purpose

Get the current version of the Avtec PTP for Windows Server as a long integer. For example, 149 is returned to indicate version 1.49.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETSTATIONID**PTPGETSTATIONIDEX****Syntax**

```
PTP_RETURN ptpNetStationId(PTP_32 *StationId, PTP_32 paramsSize)
PTP_RETURN ptpNetStationIdEx(PTP_16 Id, PTP_32 *StationId, PTP_32 paramsSize)
```

Purpose

Retrieves station Id of Avtec PTP for Windows Server.

Return Value

Returns nonnegative number if successful, negative otherwise.

PTPGETSTATIONTIME**PTPGETSTATIONTIMEEX****Syntax**

```
PTP_RETURN ptpNetStationTime(unsigned char *Dest, PTP_32 paramsSize)
PTP_RETURN ptpNetStationTimeEx(PTP_16 Id, unsigned char *Dest, PTP_32 paramsSize);
```

Purpose

Retrieve current Avtec PTP for Windows Server time in PB-4 format and place it in the parameter Dest. The parameter paramsSize must be greater than or equal to 6 bytes. PB-4 format is defined for big endian platforms. The byte order will be swapped for little endian platforms. PB-4 format is defined in the table below.

Table 2-2: PB-4 Format

Item	Field Name	Format & Size	Value
1	Parity(MSB for BIG ENDIAN)	Unsigned integer (2 bits)	Not applicable
2	Days of Year	Unsigned integer (9 bits)	Range 1 to 365
3	Milliseconds of Day	Unsigned integer (27 bits)	Range = 0 to 86399999
4	Microseconds of a Millisecond (LSB for BIG ENDIAN)	Unsigned integer (10 bits)	Range = 0 to 999
	Total Length	6 bytes	

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

The Avtec PTP for Windows Server returns the time from the currently selected time source in PB-4 format. The default source is the PC system time. If an Apogee ISA-STG2 Synchronized Time Generator is installed in the system and enabled, then the Avtec PTP for Windows will return the current PB-4 time from the ISA-STG2. The Apogee ISA-STG2 is enabled using the Avtec PTP for Windows Server command LOADAPOGEEFPGA or SELECTAPOGEE.

PTPGETTEXT**Syntax**

```
char * ptpGetText(char *Text)
```

Purpose

Receives keyboard input.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETTIMESTATEEx**Syntax**

```
PTP_RETURN ptpGetTimeStateEx(PTP_16 Id, PTP_TIME_CONTROL *T, unsigned short TSize);
```

Purpose

Gets the current configuration of the Apogee time board on PTPServer.

Parameters

```
typedef struct _PTP_TIME_CONTROL
{
    PTP_8 FpgaFileName[256];      /* Full Path to xxxxx.EXO file */
    PTP_8 Time_Input_Select;     /* FSTS_TM_SEL_(TIME, 1PPS, FREEZE, EXT_REF,
                                  AUX_A, AUX_B, AUX_C, AUX_D) */
#define PTP_TM_SEL_TIME          0      /* Time inut BNC connector */
#define PTP_TM_SEL_1PPS          1      /* 1PPS input BNC connector */
#define PTP_TM_SEL_FREEZE        2      /* FREEZE input BNC connector */
#define PTP_TM_SEL_EXT_REF       3      /* EXT REF Input BNC connector */
#define PTP_TM_SEL_AUX_A          4      /* Aux. Time Input - A */
#define PTP_TM_SEL_AUX_B          5      /* Aux. Time Input - B */
#define PTP_TM_SEL_AUX_C          6      /* Aux. Time Input - C */
#define PTP_TM_SEL_AUX_D          7      /* Aux. Time Input - D */
    PTP_BOOL Synchronized_Time; /* TRUE for synchronized time generation,
                                 FALSE for free running time */
    PTP_BOOL External_Reference; /* set to TRUE to use an external 10MHz
                                 reference clock */
    PTP_BOOL Sync_To_1PPS;      /* set to TRUE to synchronize setting time to
                                 1PPS signal */
    PTP_BOOL Leap_Year;         /* set to TRUE if this is a leap year */
} PTP_TIME_CONTROL;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETTIMESTATUSEx**Syntax**

```
PTP_RETURN ptpGetTimeStatusEx(PTP_16 Id, PTP_TIME_STATUS *T, unsigned short TSize)
```

Purpose

Gets the time status by filling in the in PTP_TIMESTATUS structure. The argument **TSize** should be set to sizeof(**T**).

Parameters

```
typedef struct _PTP_TIME_STATUS
{
    PTP_8 Time_Source;           /* FSTS_TM_SRC_(SYSTEM, ISA_STG2) */
    PTP_32 Tick_Frequency;      /* resolution of tick counter(in Hz) */
    PTP_8 Input_Code_Valid;     /* TRUE if input code incrementing as expected */
    PTP_8 Input_Carrier_Present; /* TRUE if input valid carrier signal detected */
    PTP_8 FpgaFileName[256];    /* last loaded xxxxx.EXO file */
    PTP_32 Spare[8];
} PTP_TIME_STATUS;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPGETUNKNOWNMODULESTATEEX***Syntax***

```
PTP_RETURN ptpNetUnknownModuleStateEx(PTP_16 Id,
                                      PTP_32 module,
                                      char *status,
                                      PTP_32 statusSize)
```

Purpose

Used to retrieve the module state structure for a module type that is not recognized by the application using AV_PTP.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPHASDESKTOPCHANGEDEX***Syntax***

```
PTP_RETURN ptphasDesktopChangedEx(PTP_16 Id, PTP_64 *Status)
```

Purpose

Check to see if server desktop has been modified from a different input.

Parameters

*Status is a pointer to a returned value containing DESKTOP_CHANGE flags.
The possible flags are listed in av_ptp.h.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

There are 4 methods of modifying a desktop:

- ScriptFile
- Typing in the local server
- 2 control ports.

Any change by either ScriptFile or local server sets change flags for both control ports.

A change caused by control port x will set a change flag for control port y.

Flags are ‘sticky’ and are only cleared by this call.

The lower 32 bits of these flags indicate which modules were affected by the changes.

See **ptpRegisterUserAlarmCallBackEx** and **ptpCancelUserAlarmCallBackEx**.

PTPHIDE SERVEREX

Syntax

```
PTP_RETURN ptphideServerEx(PTP_16 Id, PTP_8 Hide)
```

Purpose

Hides or shows a server from/to the local user.

Parameters

Hide is actually a Boolean. 1 = hide. 0 = show.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

When a server is ‘hidden’ it is removed from the TaskManager applications list.

It will still show as a process.

The only way to ‘show’ a server is through either of the remote control ports.

Hiding a server will also shut off all local Avtec PTP for Windows displays, increasing program performance.

PTPKILLPROCESS

Syntax

```
PTP_RETURN ptckillProcess(char *ptpAddress, char *ProgramName)
```

Purpose

Closes the program specified by **ProgramName** that is running on the Avtec PTP for Windows Server machine.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

This function can be used to remotely control the execution of programs running on the Avtec PTP for Windows Server machine. This function requires that the Avtec PTP for Windows Monitor program is running on the Avtec PTP for Windows Server machine.

PTPKILLSCHEDULETHREAD

Syntax

```
PTP_RETURN ptkillScheduleThread(char *ptpAddress, PTP_16 ThreadId)
```

Purpose

Kill the thread started from either the function **ptpScheduleFileCleanup** or **ptpBeginSchedule** (under development).

ptpAddress is text, i.e. "127.0.0.1".

ThreadId Id of thread to kill returned as **InfoInOut.TimerVars.hTimer** from either **ptpScheduleFileCleanup** or **ptpBeginSchedule** (under development).. The function **ptpScheduleFileCleanup** also returns the thread id and can be passed into this function as **ThreadId**.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPLISTSERVERS**Syntax**

```
PTP_RETURN ptpListServer(PTP_SERVERENTRY *list, PTP_32 listSize)
```

Purpose

Retrieve a listing of Avtec PTP for Windows Servers that are broadcasting to the address and port number specified with a call to **ptpBeginMultiCastService**. Function returns a positive value that indicates number of items in the list array. The value **listSize** indicates size of list array user is passing. **ptpListServers** is only supported for Win32.

Parameters

```
typedef struct _PTP_SERVERENTRY
{
    PTP_8      serverName[48];
    PTP_32     serverTCPAddress;
    PTP_32     serverTCPControlPort;
    PTP_32     serverStationId;
    PTP_32     clientTCPAddress;
    PTP_32     serverStatusCount;
    PTP_8      serverActive;
    PTP_8      serverTimeOut;
    PTP_8      newEntry;
    PTP_8      unused;
    PTP_32     futureUse[6];
} PTP_SERVERENTRY;
```

Return Value

Returns number of items in list if successful, negative otherwise.

PTPLOADMODULE**PTPLOADMODULEEX****Syntax**

```
PTP_RETURN ptpLoadModule(PTP_32 moduleType,
                         const char *moduleName,
                         PTP_32 Future1,
                         void *Future2,
                         PTP_32 *Future3)

PTP_RETURN ptpLoadModuleEx(PTP_16 Id,
                          PTP_32 moduleType,
                          const char *moduleName,
                          PTP_32 Future1,
                          void *Future2,
                          PTP_32 *Future3)
```

Purpose

Loads an instance of a **moduleType** or **moduleName** and returns a handle to the instance in **module**. Module can be loaded by type or by name. **moduleType** takes precedence over **moduleName**.

Parameters

<i>PTP_32 moduleType</i>	defines the type of module to load.
<i>const char *moduleName</i>	pointer to a string containing the module name
<i>PTP_32 resource</i> to	defines the serial I/O board or auxiliary I/O board to bind
<i>void *params</i>	a pointer to the module specific parameters structure
<i>PTP_32 *module</i>	returns a handle to the module instance

Return Value

Returns module handle ID if successful, negative otherwise. The module handle ID is 0 for the first successfully loaded module, 1 for the second successfully load module to (n-1) for the nth successfully loaded module. The module handle ID is the MNN number which is displayed in the upper left portion of the title bar in the Console GUI. Where NN is replaced by 00, 01, 02, etc.

Discussion

The **ptpLoadModule** function loads an instance of the specified module. The module can be specified by type or by name. If specifying by type, set **moduleName** to NULL. If specifying by name, set **moduleType** to zero. For serial I/O or auxiliary I/O modules, the parameter **resource** specifies to which I/O board the module should bind. The **resource** parameter is not used for other modules at this time. The module specific parameters structures are defined in the next chapter. If **params** is NULL, then the module loads with the default parameters. The handle returned in **module** is used in other function calls to control and monitor the particular instance of the module.

PTPLOADPROFILE**PTPLOADPROFILEEX****PTPLOADPROFILEALLSOURCES*****Syntax***

```
PTP_RETURN ptpLoadProfile(const char *name)
PTP_RETURN ptpLoadProfileEx(PTP_16 Id, const char *name)
PTP_RETURN ptpLoadProfileAllServers(const char *name)
```

Purpose

Loads the previously stored desktop configuration in file **name** on the Avtec PTP for Windows' disk.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

ptpGetConfiguration may be called after **ptpLoadProfile** to get the module handles, connection information, and parameters for the loaded modules.

PTPNOOP**PTPNOOPEx****PTPNOOPALLSOURCES*****Syntax***

```
PTP_RETURN ptpNoOp(void)
PTP_RETURN ptpNoOpEx(PTP_16 Id)
PTP_RETURN ptpNoOpAllServers(void)
```

Purpose

Verifies that the AV_PTP control socket connection and Avtec PTP for Windows Server are alive.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPPOLLCUSTOMSTATUSEx***Syntax***

```
PTP_RETURN ptpPollCustomStatusEx(PTP_16 Id,
```

```
PTP_CUSTOM_STATUS_STRUCT *CustomStatus,
char *dest,
PTP_32 destSize)
```

Purpose

Retrieves the custom status into **dest** if a nonnegative value is returned. The custom status structures reduce the network traffic since only the pertinent status is relayed back to the caller instead of the entire status. Not all of the status was needed for each user. A custom status message is one where the user defines a structure of data points to be retrieved.

To request a custom status from the Avtec PTP for Windows Server follow these steps:

- 1) Declare a PTP_CUSTOM_STATUS_STRUCT.
- 2) Create a local user defined structure whose address will be passed as a character pointer to the argument **dest**. In the below example this structure is named UserStructure.
- 3) Build the list of PTP_CUSTOM_STATUS_ENTRYs. The following describe each member variable of the PTP_CUSTOM_STATUS_ENTRY structure:
 - **ModuleType** - if supplied will test for matching module type loaded into PTSPServer at **ModuleHandle**.
 - **ModuleHandle** - 0 based module number to be queried.
 - **FromStructureOffset** - 0 based offset of data point to be retrieved .
 - **FromStructureByteCount** - size of data point to be retrieved in bytes.
- 4) Set **PTP_CUSTOM_STATUS_STRUCT.NumberOfEntries** to indicate the number of PTP_CUSTOM_STATUS_ENTRYs.
- 5) To get the current status, call the function:
 - `ptpPollCustomStatusEx(Id (from ptpNetConnectToServer), &PTP_CUSTOM_STATUS_STRUCT, &UserStructure, sizeof(UserStructure));`

Parameters

```
typedef struct _PTP_CUSTOM_STATUS_ENTRY
{
    PTP_32 ModuleType;
    PTP_8 ModuleHandle;
    PTP_16 FromStructureOffset;
    PTP_16 FromStructureByteCount;
    PTP_8 Spare[8];
} PTP_CUSTOM_STATUS_ENTRY;

typedef struct _PTP_CUSTOM_STATUS_HEADER
{
    PTP_16 OffsetToData;
    PTP_8 StatusNum;
    PTP_8 NumberOfEntries;
    PTP_8 MulticastTTL;
    PTP_32 MulticastDest;
    PTP_8 Spare[8];
} PTP_CUSTOM_STATUS_HEADER;

typedef struct _PTP_CUSTOM_STATUS_STRUCT
{
    PTP_CUSTOM_STATUS_HEADER Header;
    PTP_CUSTOM_STATUS_ENTRY Entry[1];
```

```
 } PTP_CUSTOM_STATUS_STRUCT;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPPOLLMODULESTATE

PTPPOLLMODULESTATEEx

Syntax

```
PTP_RETURN ptpPollModuleState(PTP_32 moduleHandle, char *params, PTP_32 paramsSize)
PTP_RETURN ptpPollModuleStateEx(PTP_16 Id,
                                PTP_32 moduleHandle,
                                char *params,
                                PTP_32 paramsSize)
```

Purpose

Retrieves the current values of the module state structure for the instance of a module referred to by the handle argument **module**. This function does not increment the Avtec PTP for Windows Server command counter. The calling program should allocate memory for the structure specific to the module type. The current parameters are copied into the structure pointed to by **params**.

Return Value

Returns the module type (nonnegative) if successful, negative otherwise.

PTPPOLLUNKNOWNMODULESTATEEx

Syntax

```
PTP_RETURN ptpPollUnknownModuleStateEx(PTP_16 Id,
                                       PTP_32 module,
                                       char *status,
                                       PTP_32 statusSize);
```

Purpose

Retrieves the current values of the module state structure for the instance of a module referred to by the handle argument **module**. This function does not increment the Avtec PTP for Windows Server command counter. The calling program should allocate at least 16000 bytes of memory for the state structure pointed to by **status**.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

This function can be used to get the state structures for modules which are not recognized by the application using AV_PTP.

PTPPOSTMESSAGE**PTPPOSTMESSAGEEX*****Syntax***

```
PTP_RETURN ptpPostMessage(char *Buffer)
PTP_RETURN ptpPostMessageEx(PTP_16 Id, char *Buffer)
```

Purpose

Copy raw Avtec PTP for Windows command buffer to control socket.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPQUERYAVAILABLEMODULES**PTPQUERYAVAILABLEMODULESEX*****Syntax***

```
PTP_RETURN ptpQueryAvailableModules(PTP_FILELIST *FileList, PTP_32 listSize)
PTP_RETURN ptpQueryAvailableModulesEx(PTP_16 Id,
                                      PTP_FILELIST *FileList,
                                      PTP_32 listSize)
```

Purpose

Retrieves a list of available modules in the \ptp_nt\modules directory on the Avtec PTP for Windows Server machine.

Parameters

```
typedef struct _PTP_FILELIST
{
    PTP_8 fileName[PTP_MAX_FILES][PTP_MAX_PATH];
}PTP_FILELIST;
```

Return Value

Returns number of items in list if successful, negative otherwise.

PTPQUERYDIRECTORY**PTPQUERYDIRECTORYEX*****Syntax***

```
PTP_RETURN ptpQueryDirectory(const char *path,
                            PTP_FILELIST *FileList,
                            PTP_32 listSize)

PTP_RETURN ptpQueryDirectoryEx(PTP_16 Id,
```

```
const char *path,
PTP_FILELIST *FileList,
PTP_32 listSize)
```

Purpose

Retrieves a directory listing of file names stored in the directory pointed to by path on the Avtec PTP for Windows' disk.

Parameters

```
typedef struct _PTP_FILELIST
{
    PTP_8 fileName[PTP_MAX_FILES][PTP_MAX_PATH];
} PTP_FILELIST;
```

Return Value

Returns number of items in list if successful, negative otherwise.

PTPQUERYEXISTINGLOGS**PTPQUERYEXISTINGLOGSEX****Syntax**

```
PTP_RETURN ptQueryExistingLogs(PTP_FILELIST *FileList, PTP_32 listSize)
PTP_RETURN ptQueryExistingLogsEx(PTP_16 Id,
                                PTP_FILELIST *FileList,
                                PTP_32 listSize)
```

Purpose

Retrieves a list of log file names stored in the \ptp_user\logs directory on the Avtec PTP for Windows' disk.

Parameters

```
typedef struct _PTP_FILELIST
{
    PTP_8 fileName[PTP_MAX_FILES][PTP_MAX_PATH];
} PTP_FILELIST;
```

Return Value

Returns number of items in list if successful, negative otherwise.

PTPQUERYOPENWINDOWS**Syntax**

```
PTP_RETURN ptQueryOpenWindows(char * ptAddress, PTP_FILELIST *List)
```

Purpose

Retrieves a list of programs that are running on the Avtec PTP for Windows Server machine. The Avtec PTP for Windows Monitor program must be running on the Server machine to use this function.

Return Value

Returns number of items in list if successful, negative otherwise.

PTPQUERYSAVEDPROFILES**PTPQUERYSAVEDPROFILESEx****Syntax**

```
PTP_RETURN ptQuerySavedProfiles(PTP_FILELIST *FileList, PTP_32 listSize)
PTP_RETURN ptQuerySavedProfilesEx(PTP_16 Id,
                                  PTP_FILELIST *FileList,
                                  PTP_32 listSize)
```

Purpose

Retrieves a list of desktop configurations previously stored in the \ptp_user\desktops directory on the Avtec PTP for Windows' disk.

Parameters

```
typedef struct _PTP_FILELIST
{
    PTP_8 fileName[PTP_MAX_FILES][PTP_MAX_PATH];
} PTP_FILELIST;
```

Return Value

Returns number of items in list if successful, negative otherwise.

PTPREBOOT**Syntax**

```
PTP_RETURN ptpReboot(char * ptpAddress, PTP_32 TimeOut)
```

Purpose

This function can be used to reboot the Avtec PTP for Windows Server machine remotely. After the function is called, the Monitor program on the Server machine will wait **TimeOut** seconds before attempting to reboot the machine. This functions requires that the Monitor program is running on the Server machine.

Return Value

Returns number of items in list if successful, negative otherwise.

PTPREGISTERCALLBACK

Syntax

```
PTP_RETURN ptpRegisterCallBack(void (*userFunction)(void *structure,
                                                 PTP_32 structureSize,
                                                 PTP_32 serverTCPAddress,
                                                 PTP_32 serverTCPPort,
                                                 PTP_32 serverStationId,
                                                 PTP_32 moduleType,
                                                 PTP_32 moduleHandle),
                                 char *serverTCPAddress,
                                 PTP_32 serverTCPPort,
                                 PTP_32 serverStationId,
                                 PTP_32 moduleType,
                                 PTP_32 moduleHandle)
```

Purpose

Registers local function for call back dependent on particular module event. Each time a module sends a multi-cast status message, the user provided call back function will be executed. Module events are filtered by setting values for **ModuleType** and **ModuleHandle**. Only supported for WIN32. See the Callback example in \ptpapi\callback for an example of how to use this function.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPREGISTERCUSTOMSTATUSEx

Syntax

```
PTP_RETURN ptpRegisterCustomStatusEx(PTP_16 Id, PTP_CUSTOM_STATUS_STRUCT
*CustomStatus)
```

Purpose

Request the Avtec PTP for Windows Server to begin sending custom multicast status messages. The following input parameters are required:

- **StatusNum** - Message number to register. The maximum is PTP_CUSTOM_STATUS_MAX_ENTRIES.
- **MulticastTTL** - Message time to live.
- **MulticastDest** - Class destination address.

See \ptpapi\customstatus for an example program using this function.

Parameters

```
typedef struct _PTP_CUSTOM_STATUS_ENTRY
{
    PTP_32 ModuleType;
    PTP_8 ModuleHandle;
    PTP_16 FromStructureOffset;
```

```

PTP_16 FromStructureByteCount;
PTP_8 Spare[8];
} PTP_CUSTOM_STATUS_ENTRY;

typedef struct _PTP_CUSTOM_STATUS_HEADER
{
    PTP_16 OffsetToData;
    PTP_8 StatusNum;
    PTP_8 NumberOfEntries;
    PTP_8 MulticastTTL;
    PTP_32 MulticastDest;
    PTP_8 Spare[8];
} PTP_CUSTOM_STATUS_HEADER;

typedef struct _PTP_CUSTOM_STATUS_STRUCT
{
    PTP_CUSTOM_STATUS_HEADER Header;
    PTP_CUSTOM_STATUS_ENTRY Entry[1];
} PTP_CUSTOM_STATUS_STRUCT;

```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPREGISTERUSERALARMCALLBACKEx***Syntax***

```

PTP_RETURN ptpNetHasDesktopChangedEx(PTP_16 Id,
                                     void (__cdecl *userFunction)(PTP_16 ServerId,
                                                               PTP_64 AlarmValue),
                                     PTP_64 AlarmMask)

```

Purpose

Check to see if Server desktop has been modified from a different input.

Registers local function for call back dependent on particular change event.

Every message the RIL receives from the server contains the change flags. If the change flags filtered by AlarmMask are set, the user provided call back function will be executed.

Parameters

Parameter 2 is a pointer to a user callback function.

AlarmValue is sent to the user function unfiltered.

AlarmMask is used as a filter to trigger use of the callback function.

The possible flags are listed in av_ptp.h.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

The user callback function is responsible for clearing the change flags by executing **ptpNetHasDesktopChangedEx**.

See **ptpNetHasDesktopChangedEx** and **ptpNetCancelUserAlarmCallBackEx**.

PTPREMOVEMODULEDATA**PTPREMOVEMODULEDATAEx*****Syntax***

```
PTP_RETURN ptpNetModuleData(PTP_32 sourceModule,
                           PTP_32 destinationModule,
                           PTP_32 portIndex,
                           PTP_32 reserved)

PTP_RETURN ptpNetModuleDataEx(PTP_16 Id,
                           PTP_32 sourceModule,
                           PTP_32 destinationModule,
                           PTP_32 portIndex,
                           PTP_32 reserved)
```

Purpose

Removes the data output from **sourceModule** to the input of **destinationModule**. The **portIndex** specifies which source module output port for modules that provide multiple output ports.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

Removes a single data connection from source module to destination module.

PTPREMOVEMODULEEVENT**PTPREMOVEMODULEEVENTEx*****Syntax***

```
PTP_RETURN ptpNetModuleEvent(PTP_32 sourceModule,
                           PTP_32 destinationModule,
                           PTP_32 portIndex,
                           PTP_32 reserved)

PTP_RETURN ptpNetModuleEventEx(PTP_16 Id,
                           PTP_32 sourceModule,
                           PTP_32 destinationModule,
                           PTP_32 portIndex,
                           PTP_32 reserved)
```

Purpose

Removes the event output from **sourceModule** to the input of **destinationModule**. The **portIndex** specifies which source module output port for modules that provide multiple output ports.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

Removes a single event connection from source module to destination module.

PTPSAVEPROFILE

PTPSAVEPROFILEEX

Syntax

```
PTP_RETURN ptpSaveProfile(const char *name)
PTP_RETURN ptpSaveProfileEx(PTP_16 Id, const char *name)
```

Purpose

Saves the current Avtec PTP for Windows configuration desktop in the \ptp_user\desktops directory on the disk to the file **name**.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSCHEDULEFILECLEANUP

Syntax

```
PTP_RETURN ptpScheduleFileCleanup(char *ptpAddress,
                                  PTP_MONITOR_FILECLEANUPVARS *InfoInOut,
                                  int Size)
```

Purpose

Begins a thread (executing on PTSPServer) to continuously call **ptpCallFileCleanup**.

The following input parameters are required:

- **ptpAddress** is text, i.e. "127.0.0.1".
- **InfoInOut.TimerVars.fTime** specifies the start time of the first periodic cycle. Positive values indicate absolute time. Be sure to use a UTC-based absolute time, as the system uses UTC-based time internally. Negative values indicate relative time. A zero indicates to start the thread immediately. The actual timer accuracy depends on the capability of your hardware. For more information about UTC-based time, see Windows NT/2000 description of System Time.
- **InfoInOut.TimerVars.TimeOut** in millisecs defines the periodic rate of the function.
- **InfoInOut.TimerVars.RunOnce** if true function will exit after first delete cycle completes.
- **InfoInOut.FilterPath** defines the path to search, for example C:\ptp_user\mypath\
- **InfoInOut.Filter** defines the files to perform the cleanup to, for example *.Extension or *.*

- **InfoInOut.AgeInSecs** delete files that are older than this time in second. For example, 3600 (60*60) for 1 hour or older, or 604800 (60*60*24*7) for 1 week or older.

Parameters

```
typedef struct _PTP_MONITOR_FILECLEANUPVARS
{
    PTP_MONITOR_TIMERVARS TimerVars;
    char                  FilterPath[256];
    char                  Filter[256];
    long                 AgeInSecs;
    UCHAR                Pad[32];
} PTP_MONITOR_FILECLEANUPVARS;

typedef struct _PTP_MONITOR_TIMERVARS
{
    char      TimerName[256];
    HANDLE   hTimer;
    PTP_64    fTime;
    ULONG    TimeOut;
    BOOL     ThreadRunning;
    BOOL     ThreadEnable;
    BOOL     RunOnce;
    UCHAR    Pad[32];
} PTP_MONITOR_TIMERVARS;
```

Return Value

A nonnegative value indicates success and is the thread handle that is needed in the call to **ptpKillScheduleThread**. A negative value indicates an error.

PTPSENDDESKTOPCOMMAND

PTPSENDDESKTOPCOMMANDEx

Syntax

```
PTP_RETURN ptpSendDesktopCommand(PTP_32 command,
                                 void *commandParams,
                                 PTP_32 commandListSize);

PTP_RETURN ptpSendDesktopCommandEx(PTP_16 Id,
                                   PTP_32 command,
                                   void *commandParams,
                                   PTP_32 commandListSize);
```

Purpose

Used to send commands to all modules on a desktop. This function is not available to user applications.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSENDMODULECOMMAND**PTPSENDMODULECOMMANDEx*****Syntax***

```
PTP_RETURN ptpSendModuleCommand( PTP_32 module,
                                 PTP_32 moduleType,
                                 PTP_32 command,
                                 void *commandParams,
                                 PTP_32 commandListSize)

PTP_RETURN ptpSendModuleCommandEx( PTP_16 Id,
                                   PTP_32 module,
                                   PTP_32 moduleType,
                                   PTP_32 command,
                                   void *commandParams,
                                   PTP_32 commandListSize)
```

Purpose

Used to send commands to a specific module defined by the module parameter. This function is not available to user applications.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETACTIVITYDELAY**PTPSETACTIVITYDELAYEX*****Syntax***

```
PTP_RETURN ptpSetActivityDelay(PTP_32 NewDelayInMilliSecs)
PTP_RETURN ptpSetActivityDelayEx(PTP_16 Id, PTP_32 NewDelayInMilliSecs)
```

Purpose

Sets the activity timeout period in milliseconds for the Avtec PTP for Windows Server Rx Status and Tx Status activity displays. The activity status displays indicate if the module has received a buffer or transmitted a buffer within the last timeout period.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETCONFIGURATION**PTPSETCONFIGURATIONEx*****Syntax***

```
PTP_RETURN ptpSetConfiguration(PTP_CONFIG *config, PTP_32 configSize)
```

```
PTP_RETURN ptpNetConfigurationEx(PTP_16 Id, PTP_CONFIG *config, PTP_32 configSize)
```

Purpose

Set Avtec PTP for Windows configuration parameters defined in the PTP_CONFIG structure.

Parameters

```
typedef struct PTPConfig
{
    PTP_8 moduleHandle[PTP_MAX_MODULES]; /* array of module handles */
    PTP_32 moduleType[PTP_MAX_MODULES]; /* moduleType for each moduleHandle */
    PTP_8 resources[PTP_MAX_MODULES];
    PTP_32 dataConnections[PTP_MAX_MODULES][PTP_MAX_MODULE_OUTCONNECTIONS];
    PTP_32 eventConnections[PTP_MAX_MODULES][PTP_MAX_MODULE_OUTCONNECTIONS];
} PTP_CONFIG;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETCUSTOMSTATUSEx

Syntax

```
PTP_RETURN ptpNetCustomStatusEx(PTP_16 Id,
                                PTP_CUSTOM_STATUS_STRUCT *CustomStatus,
                                char *UserData,
                                PTP_32 UserDataSize)
```

Purpose

Sets the current controls (module configuration) via a custom status structure. Even though the structure is named custom status it is this same structure that is also used to send custom control settings. The custom status structures reduce the network traffic since only the pertinent control settings are sent to the Avtec PTP for Windows Server instead of the entire control settings. Not all of the control settings were needed to be sent for each module. A custom control message is one where the user defines a structure of data points to be sent.

To send a custom control to the Avtec PTP for Windows Server follow these steps:

- 1) Declare a PTP_CUSTOM_STATUS_STRUCT.
- 2) Create a local user defined structure whose address will be passed as a character pointer to the argument **dest**. In the below example this structure is named UserStructure.
- 3) Build the list of PTP_CUSTOM_STATUS_ENTRYs. The following describe each member variable of the PTP_CUSTOM_STATUS_ENTRY structure:
 - **ModuleType** - if supplied will test for matching module type loaded into PTPServer at **ModuleHandle**.
 - **ModuleHandle** - 0 based module number to be queried.
 - **FromStructureOffset** - 0 based offset of data point to be retrieved .
 - **FromStructureByteCount** - size of data point to be retrieved in bytes.

- 4) Set **PTP_CUSTOM_STATUS_STRUCT.NumberOfEntries** to indicate the number of PTP_CUSTOM_STATUS_ENTRYs.
- 5) To set the current controls(module configuration), call the function:
 - **ptpSetCustomStatusEx**(Id(from ptpNetConnectToServer), &PTP_CUSTOM_STATUS_STRUCT, &UserStructure, sizeof(UserStructure));

Parameters

```
typedef struct _PTP_CUSTOM_STATUS_ENTRY
{
    PTP_32 ModuleType;
    PTP_8 ModuleHandle;
    PTP_16 FromStructureOffset;
    PTP_16 FromStructureByteCount;
    PTP_8 Spare[8];
} PTP_CUSTOM_STATUS_ENTRY;

typedef struct _PTP_CUSTOM_STATUS_HEADER
{
    PTP_16 OffsetToDate;
    PTP_8 StatusNum;
    PTP_8 NumberOfEntries;
    PTP_8 MulticastTTL;
    PTP_32 MulticastDest;
    PTP_8 Spare[8];
} PTP_CUSTOM_STATUS_HEADER;

typedef struct _PTP_CUSTOM_STATUS_STRUCT
{
    PTP_CUSTOM_STATUS_HEADER Header;
    PTP_CUSTOM_STATUS_ENTRY Entry[1];
} PTP_CUSTOM_STATUS_STRUCT;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETDEFAULTPATHEx

Syntax

```
PTP_RETURN ptpNetDefaultPathEx (PTP_16 Id, char *NewPath)
```

Purpose

Set the file system ‘default working path’ for the specified server.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

Every server may have a different working path.

The working path is used by newly loaded modules to preset file names.

The working path is also used by the server while loading and saving desktops.

The default Avtec PTP for Windows directory structure will be added to the user provided NewPath.

\|Host\c\MyPTPStuff results in \|Host\c\MyPTPStuff\Ptp_User\Desktops,
 \|Host\c\MyPTPStuff\Ptp_User\Logs etc...

If the resulting working path does not exist **ptpSetDefaultPathEx** will not create it.
 See **ptpGetDefaultPathEx** and **ptpCreateDefaultPathEx**.

PTPSETDESKTOPTTL

PTPSETDESKTOPTTLEX

PTPSETDESKTOPTTLALLSOURCES

Syntax

```
PTP_RETURN ptpSetDesktop(PTP_8 TTL)
PTP_RETURN ptpSetDesktopTTLEX(PTP_16 Id, PTP_8 TTL)
PTP_RETURN ptpSetDesktopTTLAllServers(PTP_8 TTL)
```

Purpose

Set Time To Live (TTL) for multicast status messages from all modules in a desktop. The TTL setting controls how many hops the message will be routed. The default is 1.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETMODULESTATE

PTPSETMODULESTATEEx

Syntax

```
PTP_RETURN ptpSetModuleState(PTP_32 moduleHandle,
                           PTP_32 moduleType,
                           char *params,
                           PTP_32 paramsSize)

PTP_RETURN ptpSetModuleStateEx(PTP_16 Id,
                           PTP_32 moduleHandle,
                           PTP_32 moduleType,
                           char *params,
                           PTP_32 paramsSize)
```

Purpose

Sets the current values of the module structure for the instance of a module referred to by the handle argument **moduleHandle**. The module handle is returned by the **ptpLoadModule** or **ptpLoadModuleEx** functions. The argument **params** must point to the module control structure and the argument **paramSize** must be equal to the size of this control structure. The local control

parameters are copied into the module control structure residing on the Avtec PTP for Windows Server.

Parameters

Pointer to control section of module status and control structure.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETMODULETTL

PTPSETMODULETTLEx

Syntax

```
PTP_RETURN ptpNetModuleTTL(PTP_32 moduleHandle, PTP_32 moduleType, PTP_8 TTL)
PTP_RETURN ptpNetModuleTTLEx(PTP_16 Id,
                           PTP_32 moduleHandle,
                           PTP_32 moduleType,
                           PTP_8 TTL)
```

Purpose

Sets the time-to-live (TTL) setting for the Avtec PTP for Windows multicast status messages for the instance of a module referred to by the handle argument **module**. The TTL setting controls how many hops the message will be routed. The default is 1.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETMULTICASTADDRESS

PTPSETMULTICASTADDRESSEx

PTPSETMULTICASTADDRESSALLSOURCES

Syntax

```
PTP_RETURN ptpNetMulticastAddress(char *AddressText)
PTP_RETURN ptpNetMulticastAddressEx(PTP_16 Id, char *AddressText)
PTP_RETURN ptpNetMulticastAddressAllServers(char *AddressText)
```

Purpose

Set IP address for Avtec PTP for Windows periodic status messages. Set to a class D address for multi-cast status. Can also be set to a class C address for unicast status.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETRILSOCKETTIMEOUT**Syntax**

```
PTP_RETURN ptpNetRILSocketTimeOut( PTP_32 NewTimeOut )
```

Purpose

Sets socket timeout for RIL network messages.

Parameters

For Win32 applications NewTimeOut is in milliseconds.

For Unix/Solaris applications NewTimeOut is in seconds.

Return Value

Returns nonnegative number if successful, negative otherwise.

Discussion

Api calls to set socket timeouts do not work correctly in all O/Ss.

In the RIL – any network error will automatically cause a pause for NewTimeOut and a retry.

PTPSETSTATIONTIMEEX**Syntax**

```
PTP_RETURN ptpNetStationTimeEx(PTP_16 Id, unsigned char *Src, PTP_32 paramsSize)
Src is expected to be a pointer to a 6 byte PB4 time structure.
```

Purpose

Sets the Apogee time board on PTPServer to the indicated time.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSETSTATUSDELAY**PTPSETSTATUSDELAYEX****Syntax**

```
PTP_RETURN ptpNetStatusDelay(PTP_32 NewDelayInMilliSecs)
PTP_RETURN ptpNetStatusDelayEx(PTP_16 Id, PTP_32 NewDelayInMilliSecs)
```

Purpose

Sets the time between Avtec PTP for Windows Server periodic status messages in milliseconds. The Avtec PTP for Windows Server transmits the state structure for a particular module on each update.

Return Value

Returns nonnegative number if successful, negative otherwise.

PTPSETTIMESTATEEX**Syntax**

```
PTP_RETURN ptpSetTimeStateEx(PTP_16 Id, PTP_TIME_CONTROL *T, unsigned short TSize);
```

Purpose

Sets the configuration of the Apogee time board on PTPServer.

Parameters

```
typedef struct _PTP_TIME_CONTROL
{
    PTP_8 FpgaFileName[256];      /* Full Path to xxxxx.EXO file */
    PTP_8 Time_Input_Select;      /* FSTS_TM_SEL_(TIME, 1PPS, FREEZE, EXT_REF,
                                    AUX_A, AUX_B, AUX_C, AUX_D) */
#define PTP_TM_SEL_TIME          0      /* Time inut BNC connector */
#define PTP_TM_SEL_1PPS           1      /* 1PPS input BNC connector */
#define PTP_TM_SEL_FREEZE         2      /* FREEZE input BNC connector */
#define PTP_TM_SEL_EXT_REF        3      /* EXT REF Input BNC connector */
#define PTP_TM_SEL_AUX_A          4      /* Aux. Time Input - A */
#define PTP_TM_SEL_AUX_B          5      /* Aux. Time Input - B */
#define PTP_TM_SEL_AUX_C          6      /* Aux. Time Input - C */
#define PTP_TM_SEL_AUX_D          7      /* Aux. Time Input - D */
    PTP_BOOL Synchronized_Time;   /* TRUE for synchronized time generation,
                                    FALSE for free running time */
    PTP_BOOL External_Reference; /* set to TRUE to use an external 10MHz
                                reference clock */
    PTP_BOOL Sync_To_1PPS;        /* set to TRUE to synchonize setting time to
                                1PPS signal */
    PTP_BOOL Leap_Year;           /* set to TRUE if this is a leap year */
} PTP_TIME_CONTROL;
```

Return Value

Returns nonnegative if successful, negative otherwise.

PTPSPAWN**Syntax**

```
PTP_RETURN ptpSpawn(char *ptpAddress, char *ProgramName, char *Arg)
```

Purpose

Spawns the program **ProgramName** with argument **Arg** on the Avtec PTP for Windows Server machine. Requires that the Monitor program is running on the Server machine.

Return Value

Returns nonnegative number if successful, negative otherwise.

PTPUNLOADMODULE**PTPUNLOADMODULEEX*****Syntax***

```
PTP_RETURN ptpUnloadModule(PTP_32 module)
PTP_RETURN ptpUnloadModuleEx(PTP_16 Id, PTP_32 module)
```

Purpose

Destroys the particular instance of a module. The **module** argument is the handle that was returned by **ptpLoadModule**. Any connections to the module instance are broken and any I/O board resources in use by the module instance are released.

Return Value

Returns nonnegative if successful, negative otherwise.

Discussion

All data and event connections to the module must be removed prior to unloading the module.

PTPWINCLS***Syntax***

```
void ptpWinCls(void)
```

Purpose

Clears window of text application.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPWINGOTOXY***Syntax***

```
void ptpWinGotoxy(int x, int y)
```

Purpose

Position text cursor in window of text application at location x, y.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPZEROALLMODULECOUNTERS**PTPZEROALLMODULECOUNTERSEX****PTPZEROALLMODULECOUNTERSALLSOURCES*****Syntax***

```
PTP_RETURN ptpZeroAllModuleCounters(void)
PTP_RETURN ptpZeroAllModuleCountersEx(PTP_16 Id)
PTP_RETURN ptpZeroAllModuleCountersAllServers(void)
```

Purpose

Zeroes the status counters for all loaded modules.

Return Value

Returns nonnegative if successful, negative otherwise.

PTPZEROMODULECOUNTERS**PTPZEROMODULECOUNTERSEX*****Syntax***

```
PTP_RETURN ptpZeroModuleCounters(PTP_32 module)
PTP_RETURN ptpZeroModuleCountersEx(PTP_16 Id, PTP_32 module)
```

Purpose

Zeroes the status counters for the module instance specified by **module**.

Return Value

Returns nonnegative if successful, negative otherwise.

Chapter 3

Programmer's Examples

EXAMPLE 1

The sample program outlined below configures the Avtec PTP for Windows using the function calls available in the Avtec PTP for Windows programmer's library. In this example, the Avtec PTP for Windows Server is configured to accept a CCSDS serial telemetry stream. It then directs virtual channel 0 to an IPDU formatter and sends this encapsulated data out a socket connection. Virtual channels 1 and 2 are directed to a file local to the Server. This telemetry data is received by an IPDU receiver module and displayed by the Null Transceiver module. The Server also is generating command frames that are also encapsulated in IPDU headers and sent out a socket connection. This command data is received by an IPDU receiver module and sent to the AT-HSIO2 serial output module.

The telemetry data flow of the example is shown in **Figure 3-1**.

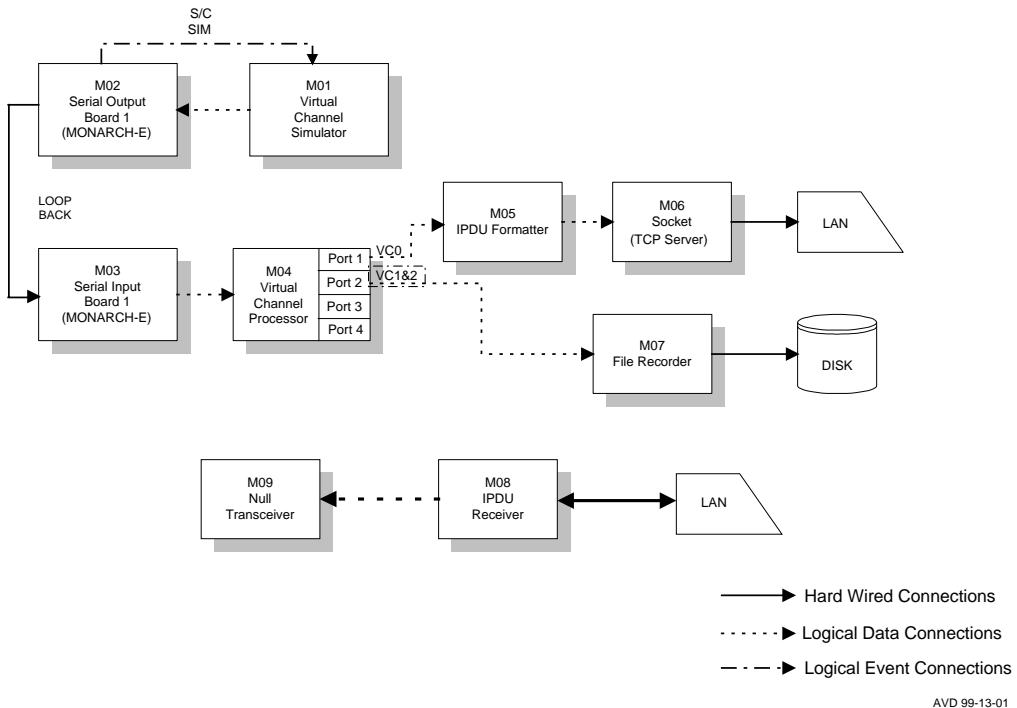


Figure 3-1: Demo Program Telemetry Data Flow Configuration (TestAvptp.cpp)

The modules used for the telemetry data flow are:

- Virtual Channel Simulator (VCS) (Simulate Space Craft (S/C))
- Serial Output (MONARCH-E)
- Serial Input (MONARCH-E)
- Virtual Channel Processor(VCP)
- Internet Protocol Data Unit (IPDU) Formatter
- Network Socket
- File Recorder
- IPDU Receiver
- Null Transceiver

The Avtec PTP for Windows accepts a simulated serial telemetry stream via loop-back within the Monarch board for purpose of quickly demonstrating an operational configuration. The source of the simulated telemetry stream is the Virtual Channel Simulator Module that is connected to a Serial Output Module. The MONARCH-E performs frame synchronization, derandomization, Reed-Solomon decoding, and time tagging. The Serial Input Module controls the input stream operations of the MONARCH-E, while the Serial Output Module controls the output stream

operations of the MONARCH-E. The Serial Input Module's data output port is connected to the CCSDS VCP Module. The Serial Input Module passes telemetry frames with time tag and quality annotation to the CCSDS VCP module. The CCSDS VCP filters frames based on the Virtual Channel Identification (VCID) and data quality. The VCP has multiple data output ports. Each data output port can be configured to output a particular subset of Virtual Channels.

In this example, the VCP port 1 is configured to pass only frames with VCID 0 and is connected to the IPDU Formatter Module. The IPDU Formatter receives Virtual Channel Data Units (VCDUs) from the VCP port 1 and encapsulates this data in an IPDU header. The IPDU Formatter is connected to Socket Module that is configured as a TCP server. The Socket Module sends VCID 0 frames with an IPDU header over the network in real time. These frames are received in the IPDU receiver and are then passed to the Null Transceiver Module for display. This branch of the telemetry data stream ends at the Null Transceiver Module. The other data branch from the VCP Module, is passed through port 2. This port is configured to pass only frames with VCID 1 or 2 and to discard frames with other VCIDs or with sequence errors. The VCP port 2 is connected to the Recorder Module. The frames are then stored to disk by the Recorder Module. This other telemetry data stream branch ends at the Recorder Module.

The Virtual Channel Simulator (VCS) module default parameters are modified as follows:

- Reed-Solomon enable is set to false
- Cyclic Redundancy Check (CRC) enable is set to false.

The MONARCH-E Serial Output module is configured as follows:

- Board Value is set to 2 (*MONARCH-E, Avtec PTP for Windows system dependent*)
- Frame length is set to 1264 bytes (default)
- Clock source is set to internal DDS (default)
- Clock frequency is set to 1.2 MHz
- Idle pattern is set to toggle
- Data is set to randomize
- Data is set to be Reed-Solomon encoded
- The flag parameter is also set to PTP_OUT_CONV BASIS

The MONARCH-E Serial Input module is configured as follows:

- Board Value is set to 2 (*MONARCH-E, Avtec PTP for Windows system dependent*)
- Input source is set to “Loop-Back” (default)
- Timeout value is set to 10000
- Data is set to be de-randomized
- Data is appended with the Reed-Solomon extended status

- The flag parameter is also set to PTP_IN_CORRECTION

The Virtual Channel Processor module is configured as follows:

- Virtual Channel 0 is filtered to port 1
- Virtual Channel 1 is filtered to port 2
- Virtual Channel 2 is filtered to port 2
- Software CRC is disabled

The IPDU Formatter module is configured as follows:

- Source code identifier is set to SFS (2)
- Destination code identifier is set to L7M (1)

The Network Socket module is configured as follows:

- Socket type is set to TCP server
- Avtec PTP for Windows address is set to the IP address where the Server is running
- Avtec PTP for Windows port is set to 1005 (Same as the below IPDU Receiver module)
- The File Recorder module is configured as follows:
 - File name is set to TestAvptp.txt
 - File record mode is set to circular

The IPDU Receiver module is configured as follows:

- Test Bit Steering is Disabled
- Header Strip is Enabled (default)
- Data Path is Enabled (default)
- Echo Path is Disabled
- Echo Generator is Disabled
- Data Filter is Disabled
- Socket Type is set to TCP Client
- Remote address is set to the IP address where the Server is running.
- Avtec PTP for Windows port is set to 1005 (same as the above Socket module)

The Null Transceiver module is configured as follows:

- Display size is set to 1264 bytes

A command data flow of the example is shown in **Figure 3-2**.

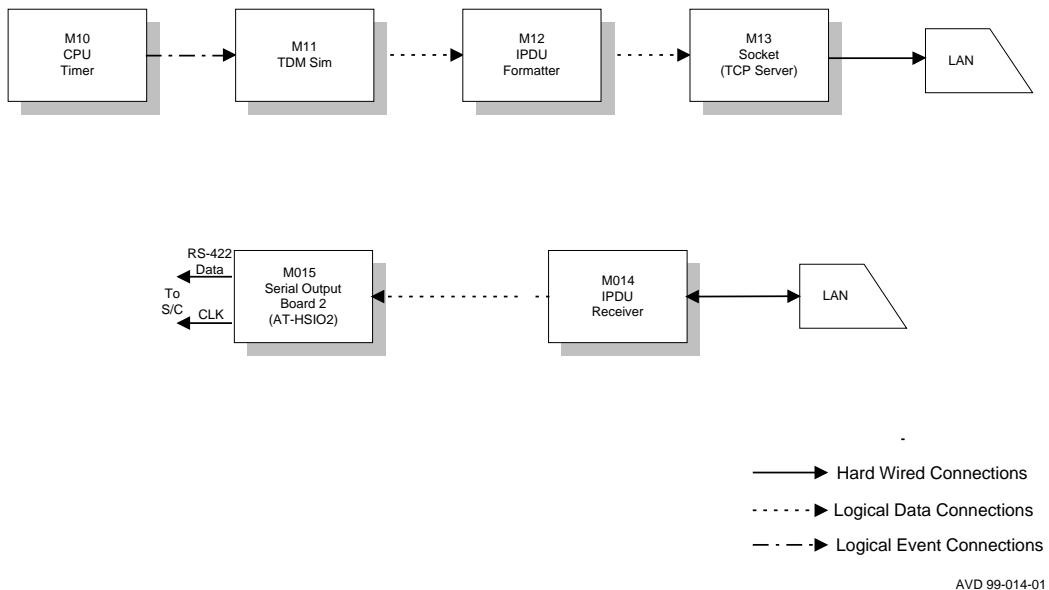


Figure 3-2: Demo Program Command Data Flow Configuration (TestAvptp.cpp)

The modules used for the command data flow are:

- Central Processing Unit (CPU) Timer
- Time Division Multiplex (TDM) Simulator (Sim)
- IPDU Formatter
- Network Socket
- IPDU Receiver Socket
- Serial Output (AT-HSIO2)

The CPU Timer Module sends an event to the TDM Sim Module that transmits a 32 byte command. The IPDU Formatter Module encapsulates this command frame with an IPDU header and then sends it to the Network Socket Module that sends it to the network. The Avtec PTP for Windows accepts these commands from the network using the IPDU Receiver. The IPDU Receiver is a special type of Socket module that supports the reception of IPDU encapsulated data. It is configured as a TCP client and accepts IPDU encapsulated CLTU from a network client. The IPDU Receiver data output is connected to the Serial Output Module. To support variable length commands the Serial Output Module control flag is set to commanding mode. The IPDU receiver strips the IPDU header from accepted network packets and passes the command data to the Serial Output Module. The Serial Output Module serializes the data using the AT-HSIO2.

The CPU Timer module is configured as follows:

- Bit Rate is set to 500 Hz (This simulates a command stream where the commands are sent at a rate less the maximum bandwidth)

- Frame Length is set to 32 bytes

The TDM Sim module is configured as follows:

- Frame Sync Patter is set to 0x1ACFFC1D.
- Frame Length is set to 32 bytes
- Counter 0 is set to increment

The IPDU Formatter module is configured as follows:

- Source code identifier is set to SFS (2)
- Destination code identifier is set to L7M (1)

The Network Socket module is configured as follows:

- Socket type is set to TCP server
- Avtec PTP for Windows address is set to the IP address where the Server is running
- Avtec PTP for Windows port is set to 1006 (Same as the below IPDU Receiver module)

The IPDU Receiver module is configured as follows:

- Test Bit Steering is Disabled
- Header Strip is Enabled (default)
- Data Path is Enabled (default)
- Echo Path is Disabled
- Echo Generator is Disabled
- Data Filter is Disabled
- Socket Type is set to TCP Client
- Remote address is set to the IP address where the Server is running.
- Avtec PTP for Windows port is set to 1006 (same as the above Socket module)

The AT-HSIO2 Serial Output module is configured as follows:

- Board Value is set to 1 (*HSIO2, PTP system dependent*)
- Idle Pattern is set to Toggle
- Source is set to Internal DDS (default)
- Clock Phase is set to 180
- Clock Frequency is set to 1000Hz
- Control flag is set to commanding mode to be able to support variable frame lengths

The following guidelines are followed in the example program:

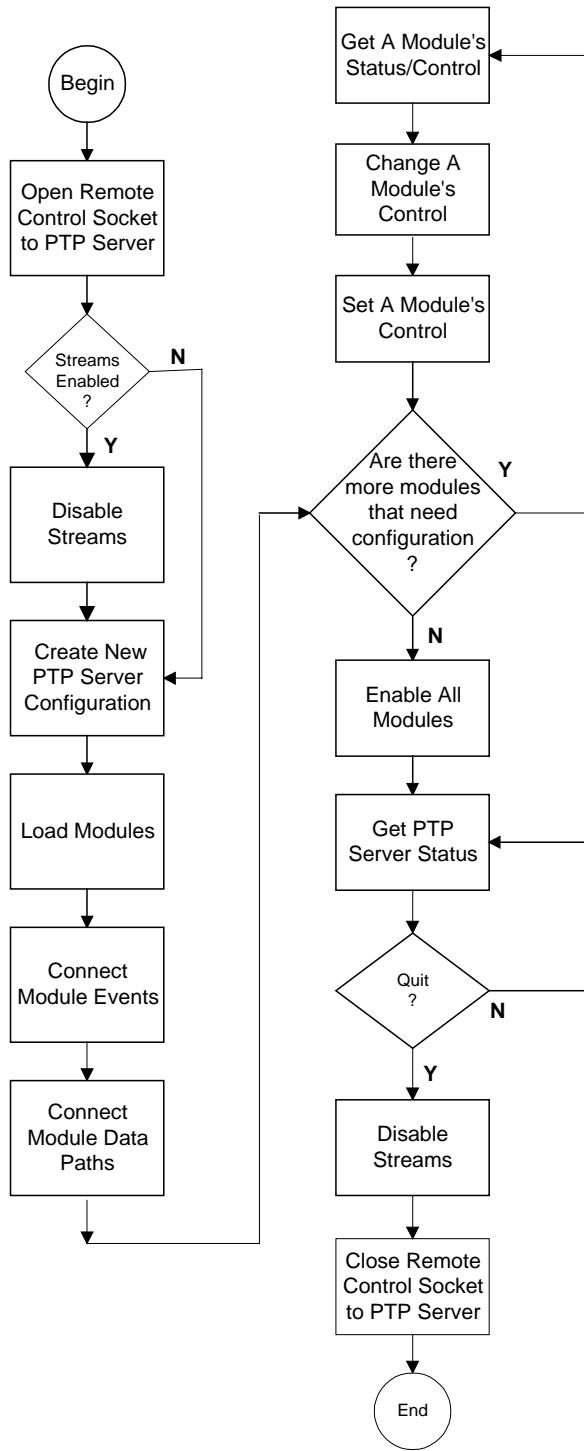
- The function **ptpConnectToServer** is the first command to the Avtec PTP for Windows Server.
- The Avtec PTP for Windows Server is set to a known state by creating a new desktop configuration.
- Modules are loaded.

Module control values are modified by getting the module status and default control settings via the module structure, modifying the control structure variables as needed and completing the sequence by setting only the control structure back to the module.

All control values other than strings or 8 bit values need to be converted for reception and transmission across the network with the one of the **ptp_swap** functions. The **ptp_swap** functions only swap the byte order of the data for operating systems that have the macro **BIG_ENDIAN** defined. This allows programmers to use the same function calls regardless of the target platform. The macro **BIG_ENDIAN** is automatically defined in the Solaris Avtec PTP for Windows Remote Interface Library (av_ptp.a) since this is a **BIG_ENDIAN** operating system. It is not defined in the NT/2000 Avtec PTP for Windows Remote Interface Library (Av_ptp.dll) since this is a little endian machine.

Both the telemetry and command data connect to the same Avtec PTP for Windows Server in this example. Up to 4 Avtec PTP for Windows Servers could have been used in this example by changing the Avtec PTP for Windows address parameters in the Network Sockets modules and the Avtec PTP for Windows remote address parameters in the IPDU Receiver modules. The telemetry data flow could have had modules 1 to 7 (VCS through File Recorder) on one Server, and modules 8 and 9 (IPDU Receiver and Null Transceiver) on another Server. The command data flow could have had modules 10 through 13 (CPU Timer through Network Sockets) on yet another Server with modules 14 and 15 (IPDU Receiver and AT-HSIO2 Serial Output) on a fourth Server.

Figure 3-3 describes the process of configuring the Avtec PTP for Windows Server.



AVD 99-015-01

Figure 3-3: TestAvptp Program Flow Chart

The following discussion pertains to a software example to remotely configure an Avtec PTP for Windows Server using the AV_PTP Remote Interface Library functions. Refer to **Figure 3-1** and **Figure 3-2** and reference the program example code distributed as “TestAvptp.cpp.” In this

example, it is assumed that the Server has installed an AT-HSIO2 board configured as board 1 and a MONARCH board configured as board 2.

Begin the process of remotely configuring the Server by opening a remote control socket connection to the Server by calling the AV_PTP Remote Interface Library function **ptpConnectToServer**. Provide the Server address in dotted Internet format as a string to argument one and set the port assignment, argument two, to 5000. The return code of this function is the control socket descriptor. A nonnegative control socket descriptor indicates that the call was successful and a connection to the Server was made. This socket is later closed with a call to **ptpDisconnectFromServer** prior to exiting the example remote control program.

The command **ptpCreateNewConfiguration** resets and clears the Avtec PTP for Windows desktop. This insures that all data streams are disabled and any previously loaded modules are unloaded from the remote Server, thus preparing the Server for a new configuration.

All modules required for telemetry data flow and command data flow are loaded next via several independent calls to the AV_PTP Remote Interface Library function “**ptpLoadModule**.” For this program example, “TestAvptp.cpp,” these modules are:

- 1) Virtual Channel Simulator
- 2) Serial Output (MONARCH-E)
- 3) Serial Input (MONARCH-E)
- 4) Virtual Channel Processor
- 5) IPDU Formatter
- 6) Network Sockets
- 7) File Recorder
- 8) IPDU Receiver
- 9) Null Transceiver
- 10) CPU Timer
- 11) TDM Sim
- 12) IPDU Formatter
- 13) Network Sockets
- 14) IPDU Receiver
- 15) Serial Output (AT-HSIO2)

Modules 1 through 9 perform the Avtec PTP for Windows Simulated Telemetry acquisition and processing functions. Modules 10 and 15 perform the Avtec PTP for Windows command processing functions. It is necessary to save the module number assigned by the Server to the module since this module number is needed again when establishing the data and event connections. This module number is also used when configuring the modules.

When all modules are loaded, data path connections and/or event path connections are created between the various modules by calling the functions **ptpConnectModuleData** and/or **ptpConnectModuleEvent**.

To generate simulated telemetry frames a Virtual Channel Simulator data path is connected to the Serial Output module for the Monarch board. An event connection is required from the Serial Output module to signal the VCS module once a frame has been sent. In this manner the VCS is instructed to generate and send a new frame.

Next a data path connection is established between the Serial Input module and the Virtual Channel Processor module. From the Virtual Channel Processor module, channel 0 is filtered and connected via data path port 1 to the IPDU Formatter module. Channels 1 and 2 are filtered and connected via data path port 2 to the File Recorder module. Another data path connection is required for the telemetry data stream between the IPDU Formatter module and the Network Sockets module. A data path is also needed between the IPDU Receiver and the Null Transceiver modules.

For the Command data stream, a CPU timer need an event path between it and the TDM simulator. The TDM simulator has a data path connection between it and the IPDU Formatter module. The IPDU Formatter module has a data path connection to the Network Sockets module. There is also a data path connection between the IPDU Formatter and the AT-HSIO2 Serial Output module.

The saved module number obtained when loading the module is used to retrieve module-specific status and control with a call to the function **ptpGetModuleState**. The return value for this function is the module handle. Its value will equal its module #define, i.e. PTP_XXXXX_MODULE if successful and will be negative if unsuccessful. All changes to the module control structure are performed after the default values for the module are returned from the call to **ptpGetModuleState**. Only the control structure is then written to the module residing on the Avtec PTP for Windows Server via a call to **ptpSetModuleState**. This function requires a pointer to control structure, size of control structure and the module number and its associated handle.

At this point, all modules are enabled by calling **ptpEnableAllModules**, thus initiating the telemetry and command data flow.

In this example the status of the modules is checked with a call to **ptpGetEnableStatus**. Prior to exiting the example program, a call is made to **ptpDisableAllModules** to stop all of the data and event flow. Finally, a call to **ptpDisconnectFromServer** is made to close the socket between the application and the Avtec PTP for Windows Server.

Appendix A:

ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
APID	Application ID
ATM	Asynchronous Transfer Mode
Bit Sync	Bit Synchronizer
BPSK	Binary Phase Shift Keying
CCSDS	Consultative Committee for Space Data Systems
CDH	Command Delivery Header
CLTU	Command Link Transition Unit
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DST	Deep Space Terminal
DTR	Data Terminal Ready
EOS	Earth Observing Systems
EDOS	EOS Data Operations Systems
ESH	EDOS Service Header
FM	Frequency Modulation
GPIB	General Purpose Interface Bus
GUI	Graphical User Interface
I/O	Input/Output
IDE	Integrated Drive Electronics
IP	Internet Protocol
IPDU	Internet Packet Data Unit
ISA	Industry Standard Architecture
LAN	Local Area Network
LEO-T	Low Earth Orbit Autonomous Terminal

Mbps	Megabits per Second
NASCOM	NASA Communications
NRZ-L	Non-return to Zero Level
PCI	Peripheral Component Interface
PCM	Pulse Code Modulation
PSK	Phase Shift Key
PTP	Programmable Telemetry Processor
RS	Reed-Solomon
RTP	Real Time Protocol
SCSI	Small Computer Systems Interface
SFDU	Standard Formatted Data Unit
TCP	Transmission Control Protocol
TDM	Time-division Multiplex
TFDH	Telemetry Frame Delivery Header
UDP	User Datagram Protocol
VCDU	Virtual Channel Data Unit
VCID	Virtual Channel ID
VCP	Virtual Channel Processor