ETHERNET PROTOCOL

User's Guide

Without measurement, there is no control.

PARTICLE MEASURING SYSTEMS
www.pmeasuring.com

P/N M10260

Ethernet Protocol

User's Guide

Particle Measuring Systems Headquarters
5475 Airport Blvd., Boulder, CO 80301, USA
(303) 443-7100 1-800-238-1801 FAX: (303) 449-6870
Instrument Service & Support: 1-800-557-6363
Customer Response Center: 1-877-475-3317

Particle Measuring Systems Europe
Tel: +44 (0)1684-581000 FAX: +44 (0)1684-560337
PMSEurope@pmeasuring.com

Particle Measuring Systems Asia Pacific
PMSAsiaPacific@pmeasuring.com

Particle Measuring Systems Singapore
Tel: 65 6496 0330 FAX: 65 6496 0357
PMSSingapore@pmeasuring.com

Particle Measuring Systems Japan
PMSJapan@pmeasuring.com

Particle Measuring Systems China
PMSChina@pmeasuring.com

Particle Measuring Systems Mexico
PMSMexico@pmeasuring.com

Particle Measuring Systems Puerto Rico
PMSPuertoRico@pmeasuring.com

P/N M10260 Rev C

Ethernet Protocol User's Guide
P/N M10260 Rev C

Quality Statement

The Quality Policy of Particle Measuring Systems is to strive to meet or exceed the needs and expectations of our customers and to align the activities of all employees with the common focus of customer satisfaction through continuous improvement in the quality of our products and services.

Environmental Information

This equipment must be properly disposed of at end-of-life by means of an authorized waste management system. Information regarding dismantling of the equipment and location of any hazardous materials can be located on our web site at: http://www.pmeasuring.com/support/papers/disassemblyinstructions.

Manual Conventions

WARNING

A warning in the text is used to notify the user of the potential for bodily injury or death.

CAUTION

A caution in the text is used to highlight an item that if not done, or incorrectly done, could damage the instrument and/or any materials or devices affected by the instrument.

– — NOTICE — –

A notice in the text is an instructional communication regarding requirements or policies issued by Particle Measuring Systems.

NOTE: A note in the text is used to highlight an item that is of operational importance to the user.

It is important that you observe cautions and warnings while performing the procedures described in this manual. Caution and warning labels are located on and inside the instrument to alert you to potentially hazardous conditions. Please familiarize yourself with this information.

This page is intentionally left blank.

Table of Contents

Table of Contents

Chapter 1
Introduction

This document describes the tools available to programmers who want to directly communicate with and control Particle Measuring Systems' (PMS) instruments using the specified Ethernet-based protocol.

The interface procedures made available are primarily concerned with manipulation of the information contained in each ethernet communication packet. These packets are transmitted and received by the host application. This requires that the programmer have the appropriate understanding and expertise when establishing and maintaining an ethernet Transmission Control Protocol/ Internet Protocol (TCP/IP) communication socket. A working knowledge of how ethernet and TCP/IP is processed is also required. This interface provides the means to process PMS ethernet packets, messages, and message fields as specified by this document.

The information presented in this document applies to the instruments listed in Appendix C. There is no direct or implied guarantee that future instruments will fully comply with, be limited to, or be constrained by what is described by the current interface. In fact, this interface has been designed in such a way as to allow for expansion of communication message and data field identifiers while retaining backward compatibility with previously defined instruments. This means that as new instruments are developed, new message and data field identifiers will be added as deemed appropriate.

This page is intentionally left blank.

Chapter 2
Using the Protocol

This section lists the required files and the method by which the user can access the available interface procedures. There are a number of possible ways to integrate the interface protocol routines.

The source code has been developed using the standard "C" programming language. Executables were created using the Microsoft® C/C++ compiler. All source code has been successfully compiled into PMS instruments and PMS monitoring application software. Each module is designed such that the instruments and the host application can use the same exact source code for communication.

This model is designed to be the basis for all PMS ethernet-based communication, both with instruments and between applications on a network. This interface is not specific to Windows or MS-DOS, nor is it specific to a particular communication library (i.e., Winsock).

Introduction to the Protocol

The primary interface with the protocol is a Dynamic Link Library (DLL) and its associated library and header files. The DLL is compatible with Windows 32 Bit. If an Operating System (OS) other than Windows is used, the source code for the DLL is made available.

The Ethernet Protocol Diskette

The following directories and files can be found on the distribution diskette:

Directory DLL

PMS_PAKT.DLL Ethernet Protocol Interface DLL

PMS_PAKT.LIB Ethernet Protocol Interface Library

PMS_PAKT.H Header for interface procedures

PMS_FLDS.H Header for message and data type field identifiers

Directory SOURCE

PMS_PAKT.C Source code for DLL

Directory SAMPLES

PMS_INST.C Source for interpreting data into sample structures

PMS_INST.HH Header for sample source code

Required Header Files

The PMS_PAKT.H file contains all defined values relevant to packet, message, and field processing. It also has an introductory comment section for each procedure prototype.

The PMS_FLDS.H file contains all the defined values relevant to instrument command and control including a list of all possible message types. There is a comment section for each message type where all of that message's associated Field Identifiers are listed. Also, there is a comment section for each Field Identifier where its description, base data type, scaling, and default value are listed.

Sample Code

The sample files have examples of how every message and its associated field identifiers can be processed into related structures. This code can be used as a guide or used in its entirety. No attempt to integrate this code or the associated structures into the DLL interface was made, since any expansion of the message or field identifier sets would require recompilation.

An important feature shown in the sample code is that the basic field data is extracted using "memmove" and "memcpy" type commands. This methodology is used to insure that differences in "byte alignment" compile-time selections between the DLL and the host application remain transparent.

Using the Interface in an Application

In order to use the available interface, the following steps must be taken to insure that the procedures are correctly exported and linked to the application being developed:

A) If using the DLL interface, the compile time directive must be used:

__WANT_PMS_PAKT_DLL

B) If creating your own DLL, the compile time directive must be used:

__PMS_PAKT_DLL

The PMS_PAKT.H and PMS_FLDS.H files must be placed in a directory from which your application project has been instructed to access information. If the source code is being integrated directly, the PMS_PAKT.C module must be included in the project and placed in the appropriate source directory as with the PMS_INST.C and PMS_INST.H files if the sample code is also integrated.

This page is intentionally left blank.

Chapter 3
PMS Ethernet Protocol Format

This section outlines the basic instrument requirements for PMS ethernet-based communication. In addition, the format of each PMS Ethernet communication packet is presented.

Basic Instrument Communication Requirements

The instrument makes available a single TCP/IP socket for host communication and control. Most instruments also make available various other ethernet and/or serial communication ports for setup and debug purposes. See the specific instrument documentation for information concerning the availability of additional hardware interface other than the TCP/IP connection.

The instrument documentation also outlines the steps necessary to set the required Etherenet communication parameters into the device. These parameters must be set to appropriate values before the device can be successfully integrated into a network. These parameters include, but are not limited to, the following:

- Instrument IP Address
- Multicast Address
- Gateway Address
- Net Mask

The single TCP/IP connection is made available to one host application at a time. If an application has established a connection with the instrument, no other connection allowing data/control messages to be processed will be allowed.

All PMS instruments present an available TCP/IP socket at a predefined Port Identifier number. This predefined value is defined as FAC_VIEW_SOCKET_ID in the PMS_PAKT.H header file. Notice that the PMS_PAKT.H file contains a small number of defined values required for the PMS ethernet communication protocol.

If the instrument allows for the use of a Multicast Address, it is possible, and often advantageous, for the host application to Multicast a basic identification request. This type of message is most often used to dynamically identify PMS instruments that are on-line. This feature is used via a User Datagram Protocol (UDP)/IP socket, at the same predefined Port Identifier number outlined above, and will

make available the instrument connection status and IP Address. Most messages are available for UDP/IP Multicast /Unicast transmission, but the actual runtime control of the instrument must be performed via the TCP/IP connection.

Ethernet Communication Packet Format

The PMS ethernet packet format is comprised of the following levels of information identification:

- Packet
- Message
- Field

Each contains identification and length parameters. Given the identification type and its associated length, the data can then be accessed from the packet buffer. See Appendix B for Example Byte Stream information.

PACKET

The packet header information includes the following information:

- PMS Packet Identifier (Referred to as Magic Number)
- PMS Packet Version Identifier
- Source IP Address
- Source Port Identifier
- Packet Length
- Number of Contained Messages
- Packet Data Buffer

The packet level is the base level of communication, information that is transmitted and received by the host application. A packet contains a buffer of data, which typically contains one or more messages, along with verification information and the sender's IP address and Port Identifier. The packet length includes the packet header. The interface contains procedures which allow the programmer direct access to all the packet header and data elements.

MESSAGE

The message header includes the following information:

- Message Length
- Message Type
- Message Data Buffer

The message level encapsulates processing of a single message. A message consists of a message length, type, and a data buffer. The message length includes the message header. Often, a message's data buffer will be composed of one or more fields. The interface contains procedures which allow the programmer direct access to each message header and data element.

FIELD

The field header includes the following information:

- Field Length
- Field Type
- Field Data Buffer

The field level encompasses field manipulation within a message. A field consists of a field length, identifier, and a data buffer. The field length includes only the data buffer. The fields used by the PMS instrument being integrated are outlined in Appendix C under that instrument type. The interface contains procedures which allow the programmer direct access to each field header and data elements.

This page is intentionally left blank.

Chapter 4
PMS Ethernet Protocol Procedures

This section outlines all the procedures available for each interface level. The PMS_PAKT.H file contains information for each procedure including its function, inputs, and outputs. Specific examples using the interface procedures are referenced later in this document.

Packet Level Procedures

The packet level interface consists of the following "C" language functions (see PMS_PAKT.H):

- Packet manipulation
- Packet header information access
- Packet message access
- Packet byte ordering

Packet Manipulation Procedures

These procedures manipulate packets in their entirety. Packets must be allocated and cleared prior to use. They should be freed when no longer needed. The packet clear procedure loads the packet header with most of the required default information. The user must explicitly set the instrument IP Address and Port Identifier.

DWORD dwPktGetPacketAllocatedSize(void FAR *pvPacket)

void FAR *pvPktAllocatePacket(WORD wMaxSize)

int iPktClearPacket(void FAR *pvPacket)

int iPktFreePacket(void FAR *pvPacket)

int iPktUpdateLength(void FAR *pvPacket)

Packet Header Information Access Procedures

These procedures verify, get, or set packet header elements. When transmitting a packet, the programmer must set the instrument IP address and Port Identifier. When receiving a packet, the header should be verified by received length and PMS identity number, and the contained messages "looped" through.

DWORD dwPktGetPacketLength(void FAR *pvPacket)

short bPktMagicNumberOK(void FAR *pvPacket)

WORD wPktGetPacketHeaderSize(void)

void vPktSetInternetAddress(void FAR *pvPacket,

INTERNET_ADDRESS sAddress, SOCKET_ID sSocket)

void vPktSetInternetAddressVB(void FAR *pvPacket, short Addr4,

short Addr3, short Addr2, short Addr1, SOCKET_ID sSocket)

int iPktGetInternetAddress(void FAR *pvPacket,

INTERNET_ADDRESS FAR *psInternetAddress)

int iPktGetInternetAddressVB(void FAR *pvPacket, short FAR * psAddr4,

short FAR * psAddr3, short FAR * psAddr2, short FAR * psAddr1)

int iPktGetSocketID(void FAR *pvPacket, SOCKET_ID FAR *psSocket)

int iPktGetNumMessages(void FAR *pvPacket)

Packet Message Access Procedures

These procedures allow the programmer to access or set individual messages in the packet.

void FAR *pvPktGetMessagePointer(void FAR *pvPacket, int iMsgNum)

void FAR *pvPktGetNextMessagePointer(void FAR *pvPacket,

void FAR pvPrevMsg)

int iPktAddMessage(void FAR *pvPacket, void FAR *pvMessage)

Packet Byte Ordering Procedures

These procedures can be used when converting from "little-endian" to "big-endian", or visa versa. The dwPktGetPacketLength procedure is the only PMS unpacketing procedure allowed before converting a packet to the native byte order which is "little-endian". This procedure accurately returns the packet length regardless of the byte order of the packet.

The Windows PMS_PAKT.DLL will probably be of little use to the programmer required to integrate a byte reordering process. This is because the Windows operating system running on an Intel platform will be "little-endian" by default. The direct compilation and linking of the source code will then likely be required by non-Windows system application.

DWORD dwPktGetPacketLength (void FAR *pvPacket)

Short bReversePacket(void FAR *pvDestPacket, void FAR *pvSourcePacket)

There are two source code examples:

- Example 3: Packet Reception Processing
- Example 6: Filling a Packet for Transmission, which contain a note specifying where the byte ordering would need to be performed if required.

Message Level Procedures

The message level interface consists of the following "C" language functions (see PMS_PAKT.H):

Message Manipulation Procedures

These procedures manipulate messages in their entirety. Messages must be allocated and cleared before use and cleared before each reuse. They should be freed when no longer needed.

void FAR *pvMsgAllocateMessage(WORD wMaxSize)

int iMsgClearMessage(void FAR *pvMessage)

int iMsgFreeMessage(void FAR *pvMessage)

Message Header Information Access Procedures

These procedures get or set message header elements. When creating a message the programmer must set the message type. When processing a message the programmer must get the message type to correctly interpret the associated Field Identifiers (if there are any).

DWORD dwMsgGetMessageLength(void FAR *pvMessage)

WORD wMsgGetMessageType(void FAR *pvMessage)

Void vMsgSetMessageType(void FAR *pvMessage, WORD wMsgType)

Message Field Access Procedures

This procedure gets a pointer to the message data buffer in its entirety.

void FAR *pvMsgGetData(void FAR *pvMessage)

Field Level Procedures

The field level interface consists of the following "C" language functions (see PMS_PAKT.H):

Field Header Information Access Procedures

These procedures get or set field identifier header elements. The field identifier informs the programmer as to the type of data parameter being processed.

WORD wMsgGetFieldLength(void FAR *pvMessage, WORD wFieldOfs)

WORD wMsgGetFieldID(void FAR *pvMessage, WORD wFieldOfs)

Field Access Procedures

These procedures allow the programmer to access each of the field entries for the supplied message. The "iMsgAddOnChange" procedure is used to minimize the size of the transmitted packets by searching for fields that have a default of NULL or a value of zero, and then deleting these values from the data. This is most useful when structures containing all possible data have been defined, and much of the data is at a default state. See PMS_INST.C for examples).

NOTE: The instruments use this feature extensively. If an instrument-defined field is not included in the message, its default value must be assumed.

---

void FAR *pvMsgGetField(void FAR *pvMessage, WORD wFieldOfs)

WORD wMsgGetFirstField(void FAR *pvMessage)

WORD wMsgGetNextField(void FAR *pvMessage, WORD wFieldOfs)

WORD wMsgGetNthField(void FAR *pvMessage, int iFieldNum)

WORD wMsgGetFieldByID(void FAR *pvMessage, WORD wFieldID,
    WORD wStartOfs)

int iMsgAddField(void FAR *pvMessage, WORD wFieldID,
    WORD wLength, void FAR *pvData)

int iMsgAddOnChange(void FAR *pvMessage, WORD wFieldID,
    WORD wLength, void FAR *pvData)

Chapter 5
PMS Ethernet Protocol Use

This section outlines the method by which each level of the interface can be used. Appendix A contains example "C" language source code in which packets, messages, and individual field data is processed.

The packet is the level of information that is passed directly between the host application and the PMS instrument. Those functions are the responsibility of the host application. Constructing a packet for transmission and interpreting a packet after reception are the functions supplied by the interface procedures.

Receiving a Packet

Once a packet is received, the application generally performs the following functions:

- Allocate Packet — Example #1
- Allocate Message — Example #1
- Read Packet Header — Example #3
- Read Packet Body — Example #3
- Process Messages in Packet — Example #4
- Process Fields in Message — Example #5
- Free Packet — Example #2
- Free Message — Example #2

Creating a Packet for Transmission

Use the following steps to create a packet for transmission to a PMS instrument:

- Allocate Packet — Example #1
- Allocate Message — Example #1
- Clearing the Packet — Example #6
- Setting Instrument IP and Port ID — Example #6
- Clearing the Message — Example #6
- Setting the Message Type — Example #6
- Adding the Message — Example #6

- Adding Field Data to a Message Example #7
- Free Packet Example #2
- Free Message Example #2

Chapter 6
PMS Instrument Communications and Control

This section describes instrument communication and control in general terms. The programmer must determine the actual instrument processing required based on the specific device being integrated and the specific requirements of the host application.

The PMS_FLDS.H file defines which messages are available..

NOTE: Most fields have a default value which must be assumed by the receiving entity. Therefore, even though the field was not explicitly present in the message, its default value is assumed when processing the message. Some fields have no default and must be present in the message.

The PMS_FLDS.H file also contains all the following field information:

- Message descriptions, and the allowed communication direction of these messages
- Field identifiers
- Field descriptions
- Field defaults
- Field message association.

General Instrument Configuration Processing

Normally, the first procedure to be performed is the gathering and setting of instrument configuration parameters. The configuration request process can be represented by the following message sequence:

Application		Instrument
PMS_PAKT_MSG_GET_CONFIG_INFO	→	
	←	PMS_PAKT_MSG_GET_CONFIG_INFO

The PMS_PAKT_MSG_GET_CONFIG_INFO message is sent to the instrument to request a PMS_PAKT_MSG_CONFIG_INFO message detailing its configuration capabilities. This message contains a flag to allow instruments to answer the request, even if it is already connected to an application. Normally, an instrument will not respond to this request if it is already connected to an application.

In the above exchange, the host application requests the instrument configuration data. The configuration request message from the host includes a field which informs the instrument as to the need to respond if the instrument is already in use (TCP/IP connection established). This is useful if only instruments available for connection are desired. Implied by the above situation, this message may be issued via the assigned multicast address over a UDP/IP socket. If this message is issued after the TCP/IP connection is established, the instrument must be requested to answer even if it is already in use.

The instrument configuration message delivered by the instrument includes general information about the instrument and specific information concerning its associated sample point(s). At least one sample point should be present. Sample points are defined to be identifiable data inputs monitored by the instrument.

For example, the many channels of contiguous particle sizing data is one sample point while individual temperature and humidity inputs would be two more separate sample points. The number of sample points and their associated configurations can be unique to each instrument type.

The general information section includes the instrument name, version, Medium Access Control (MAC) address, the number of sample points, whether or not it is used, and the owners IP address if used. Each set of sample point field data that follows is separated by the unique sample point field identifier.

Since each sample point may contain the same type of information, thus the same field identifier values defining this information, the sample point field identifier will separate each set of input configuration data. The sample point

configuration information delivered by the instrument not only contains the current settings but also parameters which define the possible settings that can be made and parameters that define specific characteristics of the device.

NOTE: Messages containing more than one Sample Point will have repeated field identifier values. These values must be delineated by searching for the Sample Point field identifier which separates each Sample Point data set.

NOTE: Careful attention to the meaning of each and every field identifier must be made. Many parameters not only define the current state of the feature in the device but also the possible values that can be applied to it. It is often good practice to re-request the instrument configuration after setting parameters to insure that all the host applications selections were implemented.

Once the information has been processed and a TCP/IP connection made to the instrument (if not already in place), the application can configure the instrument's sample point(s) via the following message sequence.

Application		**Instrument**
PMS_PAKT_MSG_SET_CONFIG	→	

The PMS_PAKT_MSG_SET_CONFIG message is sent to an instrument to configure it.

In the above message, the host application is setting a sample point’s configuration. The parameters that are allowed to be set for that sample point were defined in the original configuration information delivered from the instrument. If there is more than one sample point, each sample point can be configured.

General Instrument Setup and Control Processing

Normally the second procedure to be performed is to setup the instrument to begin sampling. The message sequences could include the following:

Application		Instrument
PMS_PAKT_MSG_DATE_TIME	→	
PMS_PAKT_MSG_GET_STATUS	→	
	←	PMS_PAKT_MSG_STATUS
PMS_PAKT_MSG_CONTROL	→	

The PMS_PAKT_MSG_DATE_TIME message is sent to an instrument to set the current date/time. The data is a time_t value containing the date/time as a 32-bit unsigned integer containing the number of seconds since January 1, 1970. The PMS_PAKT_MSG_GET_STATUS message is sent to the instrument to request a PMS_PAKT_MSG_STATUS message detailing its current status. This message contains no data. The PMS_PAKT_MSG_STATUS message is sent from the instrument to inform the requestor about its current state. The PMS_PAKT_MSG_CONTROL message is sent to an instrument to control it in a specified manner. The available control commands are defined in PMS_FLDS.H.

In the above message set, the application sets date and time into the instrument. This is required if the date and time in the instrument data packets is used. In addition, the application requests instrument status and (possibly based on the status received) starts the device sampling.

The specific requirements concerning instrument control can vary between instrument types. Some instruments may default to a sampling state and therefore not require a start command at all. That is why checking the instrument status before issuing a command may be handy. Some instruments like samplers may require start commands at discrete intervals.

Since the instrument being integrated may not contain a true real-time clock, it may be advantageous to update the instrument date and time on a periodic basis (once an hour for example). This insures that drift in the instrument timer is kept to a minimum. In the same way, requesting instrument status periodically may be advantageous when attempting to determine instrument connectivity. It has been found that the time-out used by the TCP/IP socket is not "timely" enough for certain applications. Using the status request is one way of establishing a local handshake/time-out depending on the implementation in the application.

NOTE: The command message can contain one of a number of predefined commands. These commands are listed in the PMS_FLDS.H file. The usage of these commands can be instrument specific. In addition, a sub-command can be specified which is defined to be instrument specific. Generally, every instrument will process the basic Start and Stop commands.

General Instrument Data and Runtime Processing

Once the instrument is setup and sampling it will send the host application data via the following message:

Application		Instrument
PMS_PAKT_MSG_DATA	↔	PMS_PAKT_MSG_DATA

The PMS_PAKT_MSG_DATA message is sent from an instrument in order to inform the connected application of new data values. The message may contain multiple sample points.

The data message is shown as being bi-directional. The primary usage of this message is for the instrument to send sample data to the host application at the times when it has been configured to do so. There are some special circumstances where a data message can be sent to an instrument. An example of this is to set the state of a special status Light Emitting Diode (LED) on a sensor. The use of this feature in general, and specifically the status LED function, is sensor specific.

There is one particular feature common to most on-line PMS ethernet instruments worth explaining in detail. There are two field identifiers, which together define a feature referred to as Variable Sample Interval. These fields are identified in the PMS_FLDS.H file as PMS_FID_SAMP_INT and PMS_FID_MAX_SAMP_INT. These are set by the host application during instrument configuration in the PMS_PAKT_MSG_SET_CONFIG message.

The Variable Sample Interval feature consists of the following processing. The sensor uses the base sample interval to process actual particle counts. IF there are no particle counts during the base sample interval, then these "zero" counts are put into a special "zero count" sample. If another base sample interval is processed where more "zero" counts are detected then this data is "accumulated" into the same "zero count" sample. This "accumulation" continues until the maximum sample interval is reached or actual particle counts are finally detected.

If the maximum sample interval is reached, a data message is sent with the "zero count" sample where the sample time is set to the configured maximum sample interval. In the case where actual particle counts are detected, a data message with the "zero count" sample where the sample time representing the time of accumulation thus far is sent and then another data message containing the actual detected particle counts is sent at the base sample interval.

It is worth noting that during runtime processing there can be unsolicited status messages from the instrument. This is instrument specific and is most often associated with more complex devices such as samplers.

Other Associated Instrument Communication

There are a number of other available message sequences used for special circumstances. The following message sequences cover these utility functions:

Application		Instrument
PMS_PAKT_MSG_RESET	→	

This message contains no data and is sent to the instrument to request that the instrument reset to its power-on state.

Use care when issuing the Reset message, as the instrument will basically "come-up" from a powered-off condition. This means that it no longer has a connection established to your application. One way of performing a "graceful" reset is to close the TCP/IP connection to the instrument and then issue the Reset message via the UDP/IP socket. Note that in this case even though the UDP/IP socket is used, the instrument IP address would be sent (not the assigned multicast address).

Application		Instrument
PMS_PAKT_MSG_SET_MULTI_ADDR	→	

This message is sent from the host application to the instrument (sensor) to set its multicast address. The new multicast address (INTERNET_ADDRESS) is the only data.

Application		Instrument
PMS_PAKT_MSG_GET_MULTI_ADDR	→	
	←	PMS_PAKT_MSG_MULTI_ ADDR

This message from the host application contains no data and is sent to request the instrument's current multicast address. The response is a PMS_PAKT_MSG_MULTI_ADDR message.

Application		Instrument
PMS_PAKT_MSG_STRING	↔	PMS_PAKT_MSG_STRING

The PMS_PAKT_MSG_STRING message is transmitted between an instrument and the host system bidirectionally. This message type was specifically designed for the "Virtual Port" type instrument but can be expanded to other instruments as required. The message contains the transmitted string.

This message allows the host application and the instrument to transmit strings. The use of this message is instrument specific.

Appendix A
Example Code

NOTE: These source code examples have been taken from their original context and are not intended to be complete or used as written. The examples are intended to show use of the interface procedures only. Any compiler specific, purchased tool, or other unexplained code is included simply for readability and continuity.

Example 1: Packet and message creation

```
static void *pTCPGeneralUsePkt; // General use pkt
static void *pTCPGeneralUseMsg; // General use msg
/*****************************************************/
/* iPMSTCPCreate Procedure                           */
/*****************************************************/
int iPMSTCPCreate(void)
{
  // Allocate general use packet and message (for sending out)
  pTCPGeneralUsePkt  = pvPktAllocatePacket(MAX_INST_UDP_MSG_LEN);
  if (pTCPGeneralUsePkt == NULL) {
    return (FAIL);
   }
   pTCPGeneralUseMsg = pvMsgAllocateMessage(MAX_INST_UDP_MSG_LEN);
  if (pTCPGeneralUseMsg == NULL) {
    return (FAIL);
  }
  return (SUCCESS);
}
```

Example 2: Packet and message destruction

```
/*****************************************************/
/* vPMSTCPDestroy Procedure                  */ /
*****************************************************/
void vPMSTCPDestroy(void)
{
   iPktFreePacket(pTCPGeneralUsePkt);
   iMsgFreeMessage(pTCPGeneralUseMsg);
}
```

Example 3: Packet reception processing

```
/************************************************/
/* vPMSTCPReceivePacket Procedure       */
/************************************************/
void vPMSTCPReceivePacket(void)
{
  int   iBytesReceived, iInCount;
  byte  *inBuffSensor;
  WORD  wHeaderLen;
  DWORD dwPacketLen;

  // Allocate sensor packet and buffer
  pTCPSensorPacketIn = pvPktAllocatePacket(MAX_INST_UDP_MSG_LEN);
  if (pTCPSensorPacketIn == NULL) return;
  inBuffSensor = (byte *)malloc(MAX_INST_UDP_MSG_LEN);
  if (inBuffSensor == NULL) return;

  // Assume sensor connected - device specific code
  // Lots of missing code which ensures that the TCP/IP
  // socket for this device is still alive and healthy

  while (TRUE) {
     // Get the expected packet header length
     wHeaderLen = wPktGetPacketHeaderSize();

    // Read in packet header
    iBytesReceived = 0;
    while ( ((WORD)iBytesReceived < wHeaderLen) && iTCPSensorConnected){

      // TCP/IP tool specific receive function
      if ( (iInCount = recv(iTCPSensorSocket, (char FAR *)inBuffSensor,
           wHeaderLen - iBytesReceived, 0)) < 1) {

        // Device specific receive error processing
        continue;
      } else {

        // More of header received
        memcpy((char FAR *)pTCPSensorPacketIn + iBytesReceived, inBuffSensor,
        iInCount);
         iBytesReceived += iInCount;
       }
     }
    // Get the header specified packet length
    dwPacketLen = dwPktGetPacketLength(pTCPSensorPacketIn);

    // Read in packet body
    while ( ((DWORD)iBytesReceived < dwPacketLen) && iTCPSensorConnected) {

      // TCP/IP tool specific receive function
      if ( (iInCount = recv(iTCPSensorSocket, (char FAR *)inBuffSensor,
           (int)(dwPacketLen - iBytesReceived), 0)) < 1) {

        // Device specific receive error processing
        continue;
      } else {
```

```
            // More of header received
             memcpy((char FAR *)pTCPSensorPacketIn + iBytesReceived, inBuffSensor,
iInCount);
               iBytesReceived += iInCount;
          }
      }
//At this point, if it is required to byte reverse the packet, call the "bReversePacket"
//procedure now. One possible implementation is shown below. Note: A real code
//segment would include the declaration and allocation of the appropriate variables.

      if ( bReversePacket(pTCPSensorPacketReversed, pTCPSensorPacketIn ) < 0 ) {
         // Device specific receive error processing …
      }

      // Process the packet
       vTCPProcessPacket(pTCPSensorPacketIn, iTCPSensorSocket);
   }
// Free local packet & buffer
iPktFreePacket(pTCPSensorPacketIn);
 free(inBuffSensor);

}
```

Example 4: Getting messages from received packet

```
/************************************************/
/* vPMSTCPProcessPacket Procedure      */
/************************************************/
void vPMSTCPProcessPacket(void *pInPacket, int iSocketnt)
{
   int        iMsgIdx;              // Message index
   void       *pMessageIn;       // Pointer to message
   WORD  wMessageType;     // Type of message pointed to by pMessageIn

  // Note: inPacket is a pointer to a pms_pakt packet (UDP or TCP).
  // iSocket is the socket number to which a reply will be sent.

  // Note: This function evaluates the data contained in a packet and
  // takes the appropriate action

  // Process all messages in packet
  for (iMsgIdx = 0; iMsgIdx < iPktGetNumMessages(pInPacket); iMsgIdx++) {

    // Get pointer to message & message type
    pMessageIn   = pvPktGetMessagePointer(pInPacket, iMsgIdx);
    wMessageType = wMsgGetMessageType(pMessageIn);

    // Process each defined message type
    switch (wMessageType) {

      // Messages from sensor
      case PMS_PAKT_MSG_CONFIG_INFO: /* Response of config info from inst */
        vReceiveConfigInfo(iSocket, pMessageIn);
        break;
      case PMS_PAKT_MSG_STATUS:          /* Instrument status from sensor */
        vReceiveStatus(iSocket, pMessageIn);
        break;
      case PMS_PAKT_MSG_DATA:             /* Data transmission */
        vReceiveData(iSocket, pMessageIn);
        break;

        .
        .. Messages you want to process .. Etc

        default:
         break;
    }
}
```

Example 5: Getting field data from received message (from PMS_INST.C)

```
/*************************************************************************/
/* The following structure contains the sample point-specific   */
/* info for a Data message                                               */
/*************************************************************************/
typedef struct {
   short        iSamplePoint;
   char FAR  *szSampleID;
   BYTE     yManifoldPosition;
   DATA_TYPE   eDataType;
   BYTE       yNumChannels;
   long        lVolume
   short      iVolExp;
   long        lDCLight;
   time_t     tSampleTime;
   BYTE      yHunSecs;
   long        lSampInt;
   BYTE       yLaserStatus;
   short       iExp;
   long        alValues[MAX_NUM_CHANS];
} SAMPLE_POINT_DATA;

/*************************************************************************/
/* Calling Sequence: int iDataGetSamplePointData                         */
/*                        (void FAR *pvMessage,                          */
/*                         short iSamplePoint,                           */
/*                         short bIsIndex,                               */
/*                         SAMPLE_POINT_DATA FAR *psData)                */
/*                                                                       */
/* Purpose: Fill a SAMPLE_POINT_DATA structure for a specific            */
/*      sample point in a PMS_PAKT_MSG_DATA message.                     */
/*                                                                       */
/* Inputs :    pvMessage      - Pointer to message                       */
/*             iSamplePoint   - Sample point number                     */
/*             bIsIndex       - Indicates that iSamplePoint is the      */
/*                              0-based index into the sample point      */
/*                              in the data message, rather than the     */
/*                              actual sample point number.              */
/*             psData       - Pointer to structure to fill.             */
/*                                                                       */
/* Outputs:  Return values defined in pms_err.h                          */
/*************************************************************************/
int iDataGetSamplePointData(void FAR *pvMessage,  short iSamplePoint,
                    short bIsIndex, SAMPLE_POINT_DATA FAR *psData)
{
   short iVal, bDone;
   BYTE  FAR *pyVal;
   WORD  wFieldOfs;
   WORD  wFieldID;
   int   iIndex;

   if (psData == NULL) return FAIL_INVALID_ARGS;

   /* Find the requested sample point in the message */
   wFieldOfs = wMsgGetFirstField(pvMessage);
   iIndex = 0;
```

```
do {
  wFieldOfs = wMsgGetFieldByID(pvMessage, PMS_FID_SAMPLE_POINT,
   wFieldOfs);
  pyVal = pvMsgGetField(pvMessage, wFieldOfs);
  if (pyVal == NULL)
    return(FAIL_INVALID_ARGS);  /* Can't find this sample point in msg */
  memmove(&iVal, pyVal, sizeof(iVal));
  wFieldOfs = wMsgGetNextField(pvMessage, wFieldOfs);
  bDone =  (short)(((iVal == iSamplePoint) && !bIsIndex) ||
             ((iIndex == iSamplePoint) && bIsIndex));
   iIndex++;
}    while(!bDone); /* Keep going until given sample point found */
 /* Clear passed structure */    memset(psData, 0, sizeof(SAMPLE_POINT_DATA));

/* Fill in non-zero defaults for psData */
psData->iSamplePoint  = iVal;
psData->eDataType     = DATA_TYPE_SAMPLE;

/* Fill in the fields for this sample point in the message */
wFieldID  = wMsgGetFieldID(pvMessage, wFieldOfs);
while ((wFieldOfs != (WORD)0) && (wFieldID != PMS_FID_SAMPLE_POINT)) {
  pyVal = pvMsgGetField(pvMessage, wFieldOfs);
  switch(wFieldID)  {
    case PMS_FID_SAMPLE_ID:
      psData->szSampleID = (char FAR *)pyVal;
      break;
    case PMS_FID_MANIFOLD_POS:
      if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
      psData->yManifoldPosition = *pyVal;
      break;
    case PMS_FID_NUM_CHANS:
      if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
      psData->yNumChannels = *pyVal;
      break;
    case PMS_FID_SAMP_INT:
      if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
      memmove(&psData->lSampInt, pyVal, sizeof(psData->lSampInt));
      break;
    case PMS_FID_DATA_TYPE:
      if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
      memmove(&iVal, pyVal, sizeof(iVal));
      psData->eDataType = (DATA_TYPE)iVal;
      break;
    case PMS_FID_VOLUME:
      if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
      memmove(&psData->lVolume, pyVal, sizeof(psData->lVolume));
      break;
    case PMS_FID_VOL_EXP:
      if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
      memmove(&psData->iVolExp, pyVal, sizeof(psData->iVolExp));
      break;
    case PMS_FID_DC_LIGHT:
      if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
      memmove(&psData->lDCLight, pyVal, sizeof(psData->lDCLight));
      break;
    case PMS_FID_SAMPLE_TIME:
      if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
      memmove(&psData->tSampleTime, pyVal, sizeof(psData->tSampleTime));
      memmove(&psData->yHunSecs, pyVal + sizeof(psData->tSampleTime),
```

```
                sizeof(psData->yHunSecs));
            break;
          case PMS_FID_LASER_STATUS:
            if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
             psData->yLaserStatus = *pyVal;
             break;
          case PMS_FID_DATA_VALUES:
            if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
            memmove(&psData->alValues, pyVal, wMsgGetFieldLength(pvMessage,
                wFieldOfs));
             break;
          case PMS_FID_DATA_EXP:
            if (pyVal == NULL)   return(FAIL_INVALID_ARGS);
            memmove(&psData->iExp, pyVal, sizeof(psData->iExp));
            break;
        }

        wFieldOfs = wMsgGetNextField(pvMessage, wFieldOfs);
        wFieldID  = wMsgGetFieldID(pvMessage, wFieldOfs);
      }

      return(SUCCESS);
    }
```

Example 6: Filling a packet for transmission

```
/*****************************************************************************/
/* The following structure contains the sample point-specific info*/
/* for a Status message*/
/*****************************************************************************/
typedef struct {
   short iSamplePoint;
   DWORD dwStatus;
   char  FAR *szStatus;
   BYTE  yManifoldPosition;
} SAMPLE_POINT_STATUS;


/************************************************/
/* vSendStatus Procedure                        */
/************************************************/
void vSendStatus(int iSocket)
{
   INSTRUMENT_STATUS sInstrumentStatus;

   // Setup the message and packet
   iPktClearPacket(pTCPGeneralUsePkt);
   vPktSetInternetAddress(pTCPGeneralUsePkt,
        (INTERNET_ADDRESS)net2hl(*(unsigned long *)sGlobalTCP.IPDisplay),
        FAC_VIEW_SOCKET_ID);

   // Setup the message
   iMsgClearMessage(pTCPGeneralUseMsg);
   vMsgSetMessageType(pTCPGeneralUseMsg,PMS_PAKT_MSG_STATUS);

   // Connection state & date/time
   sInstrumentStatus.eConnectionState = CONNECTION_LISTEN;
   sInstrumentStatus.tDateTime = time(NULL);



   // Status and string
   switch(sGlobalSystem.CommState & COMM_STATE_ONLY_MASK) {
      case COMM_STATE_ERROR:
         sInstrumentStatus.dwStatus  = PMS_INST_STATUS_ERROR;
         sInstrumentStatus.szStatus  = "Communication Error";
         break;
      case COMM_STATE_RUN:
         sInstrumentStatus.dwStatus  = PMS_INST_STATUS_SAMPLING;
         sInstrumentStatus.szStatus  = "Sampling";
         break;
      case COMM_STATE_IDLE:
      case COMM_STATE_SET_CONFIG:
      case COMM_STATE_GET_CONFIG:
      case COMM_STATE_CONNECT:
      case COMM_STATE_INIT:
      default:
         sInstrumentStatus.dwStatus  = PMS_INST_STATUS_IDLE;
         sInstrumentStatus.szStatus  = "Idle";
         break;
   }
```

```
   // This particular device has one sample point
   sInstrumentStatus.iNumSamplePoints = 1;

   // Put instrument status into message - see next example
   iStatSetInstrumentStatus(pTCPGeneralUseMsg, &sInstrumentStatus);

   // Add message and send packet
   iPktAddMessage(pTCPGeneralUsePkt, pTCPGeneralUseMsg);

   //At this point, if it is required to byte reverse the packet, call the "bReversePacket"
procedure now.  One
   //possible implementation is shown below. Note: A real code segment would include the
declaration and
   //allocation of the appropriate variables.

   if ( bReversePacket(pTCPGeneralUsePktReverse, pTCPGeneralUsePkt) < 0 ) {
      // Device specific receive error processing …
   }

   vSendPacket(pTCPGeneralUsePkt,iSocket);
}
```

Example 7: Filling a message for packet transmission (From PMS_INST.C)

```
/*****************************************************************************/
/* Calling Sequence: int iStatSetInstrumentStatus(void *pvMessage,        */
/*                                    INSTRUMENT_STATUS *psStatus)       */
/*                                                                       */
/* Purpose        : Set the general instrument status into a             */
/*                  PMS_PAKT_MSG_STATUS message.  This function         */
/*                  must be called BEFORE iStatAddSamplePointStatus().   */
/*                                                                       */
/* Inputs         : pvMessage - Pointer to message                       */
/*                    psStatus  - Pointer to instrument status           */
/*                                                                       */
/* Outputs        : Return values defined in pms_err.h                   */
/*****************************************************************************/
int iStatSetInstrumentStatus(void FAR *pvMessage, INSTRUMENT_STATUS FAR
*psStatus)
{
  int iRet;

  if ((pvMessage == NULL) || (psStatus == NULL))
    return(FAIL_GENERAL);

  iRet = iMsgAddField(pvMessage, PMS_FID_CONNECTION_STATE, sizeof(short),
       &psStatus->eConnectionState);
  if (iRet != SUCCESS) return(iRet);

  iRet = iMsgAddField(pvMessage, PMS_FID_DATE_TIME, sizeof(time_t),
       &psStatus->tDateTime);
  if (iRet != SUCCESS) return(iRet);

 iRet = iMsgAddField(pvMessage, PMS_FID_STATUS_STRING, (WORD)strlen(psStatus-
       >szStatus)+1, psStatus->szStatus);
  if (iRet != SUCCESS) return(iRet);

  iRet = iMsgAddField(pvMessage, PMS_FID_STATUS_CODE, sizeof(DWORD),
       &psStatus->dwStatus);
  if (iRet != SUCCESS) return(iRet);

  iRet = iMsgAddField(pvMessage, PMS_FID_NUM_SAMPLE_PTS, sizeof(short),
       &psStatus->iNumSamplePoints);
  if (iRet != SUCCESS) return(iRet);

  return(SUCCESS);
}
```

Appendix B
Example Byte Streams

The following examples show PMS packet data as collected from a PC running Windows and an Airnet-301 sensor (firmware revision current at the time of this document). The PMS Magic number, HEX (DEADBEEF), is sent out as HEX (EFBEADDE).

NOTE: Most parameters have defined defaults, and default data is never transmitted.

Example 1

Host 10 . 0 . 0 . 10 (Port ID 1087) Sends Airnet 10 . 0 . 3 . 46 (Port 1561) a PMS_PAKT_MSG_SET_CONFIG

Byte Stream as follows:

EF	BE	AD	DE	- Magic Number	
00	00			- Version	
0A	00	00	0A	- Senders Address	(10 . 0 . 0 . 10)
3F	04			- Senders Port	(1087)
32	00	00	00	- Packet Length	(50)
01	00			- Message Count	(1)
20	00			- Message Length	(32)
0B	00			- Message Type	(11)
02	00			- Field Length	(2)
72	00			- Field Type	(114) Sample Point
00	00			- Field Data	(0) 1st and only for Airnet
04	00			- Field Length	(4)
6A	00			- Field Type	(106) Min Sample Interval
E8	03	00	00	- Field Data	(1000) 10 Seconds
04	00			- Field Length	(4)
6B	00			- Field Type	(107) Max Sample Interval
E8	03	00	00	- Field Data	(1000) 10 Seconds
02	00			- Field Length	(2)
80	00			- Field Type	(128) Disabled Flag
00	00			- Field Data	(0) False

Example 2

Airnet-301 10 . 0 . 3 . 46 (Port ID 1561) Sends Host 10 . 0 . 0 . 10 (Port 1087) a PMS_PAKT_MSG_DATA

Byte Stream as follows:

EF	BE	AD	DE	- Magic Number	
00	00			- Version	
2E	03	00	0A	- Senders Address	(10.0.3.46)
19	06			- Senders Port	(1561)
5D	00	00	00	- Packet Length	(93)
01	00			- Message Count	(1)
4B	00			- Message Length	(75)
0A	00			- Message Type	(10) Data Message
02	00			- Field Length	(2)
7E	00			- Field Type	(126) Number of Sample Points
01	00			- Field Data	(1) Always 1 for Airnet-301
02	00			- Field Length	(2)
72	00			- Field Type	(114) Sample Point
00	00			- Field Data	(0) 1st and only for Airnet-301
02	00			- Field Length	(2)
73	00			- Field Type	(115) Data Type
01	00			- Field Data	(1) Always 1 (Particle Type) for Airnet-301
01	00			- Field Length	(1)
65	00			- Field Type	(101) Number of channels
02	00			- Field Data	(2) Always 2 for Airnet-301
04	00			- Field Length	(4)
74	00			- Field Type	(116) Volume
85	45	01	00	- Field Data	(83333)
02	00			- Field Length	(2)
90	00			- Field Type	(144) Volume Exponent
04	00			- Field Data	(4)
05	00			- Field Length	(5)
76	00			- Field Type	(118) Sample Time
65	74	04	44 00	- Field Data	(time_t data type and Huns of seconds)
04	00			- Field Length	(4)
6A	00			- Field Type	(106) Sample Interval (Huns of seconds)
F4	01	00	00	- Field Data	(500) 5 Seconds
01	00			- Field Length	(1)
77	00			- Field Type	(119) Laser Status
0D				- Field Data	(13)
08	00			- Field Length	(1)
78	00			- Field Type	(120) Data Array
38	4A	0E	00	- Field Data	(936504) Channel 1 Cumulative Data

B2 E4 0D 00 - Field Data Cont. (910514) Channel 2 Cumulative Data

This page is intentionally left blank.

Appendix C
Sensor-specific Fields

Airnet® Message Fields

`PMS_PAKT_MSG_RESET` Host →Inst: Reset the instrument.

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument.

Field Names: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host ←Inst: Configuration information from the instrument

Field Names:

PMS_FID_INST_FAMILY_NAME
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_MINMAX_NUM_CHANS
PMS_FID_SIZES
PMS_FID_MIN_SIZES
PMS_FID_MAX_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_MINMAX_SAMP_INT
PMS_FID_INST_NAME

PMS_FID_INST_VERSION
PMS_FID_DISABLED
PMS_FID_VOLUME_UNITS
PMS_FID_FLOW_RATE
PMS_FID_SSTS
PMS_FID_VOLUME_TYPE
PMS_FID_CONTROL_DATA
PMS_FID_POLL_INT
PMS_FID_SAMPLE_POINT_TYPE
PMS_FID_TARE_TIME
PMS_FID_REPEAT_COUNT

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Name: None

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

Field Names:

PMS_FID_DATE_TIME
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING
PMS_FID_NUM_SAMPLE_PTS

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Name: PMS_FID_DATE_TIME

`PMS_PAKT_MSG_DATA` Host ↔ Inst: Data transmission

Airnet supports data transmission both to and from the sensor. When data is sent to the Airnet it is interpreted as being the state of the multi-color LED on the sensor. This data consists of a single sample point with a single data value.
If the value = 0, the LED will be green.
If the value = 1, the LED will be orange.
If the value = 2, the LED will be red.

Field Names:

PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SAMP_INT
PMS_FID_DATA_TYPE

PMS_FID_VOLUME
PMS_FID_SAMPLE_TIME
PMS_FID_DATA_VALUES
PMS_FID_LASER_STATUS
PMS_FID_DATA_EXP
PMS_FID_VOL_EXP

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

Field Names:

PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SSTS
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_DISABLED
PMS_FID_TARE_TIME
PMS_FID_REPEAT_COUNT

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

Field Name: PMS_FID_COMMAND

CLS-900 Message Fields

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

CLS-900 accepts the reset command through a TCP packet (if connected via TCP to a Host) or through a UDP packet (must be by Host if connected via TCP).

Field Name: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

In versions prior to 1.5, the volume units are assumed to be mL.

CLS-900 can handle up to five CLS-920/CLS-930 samplers and Nano sensors. Multiple samplers/sensors can cause this message to grow larger than can be handled by a single UDP packet. To prevent a UDP packet problem, this message is broken up into as many packets as there are samplers/sensors. Each message depicts the number of sample points as the entire number of samplers/sensors on the CLS-900 system. This allows the Host to accumulate all of the device configuration data before proceeding.

Field Names:

PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_MINMAX_NUM_CHANS
PMS_FID_SIZES
PMS_FID_MIN_SIZES
PMS_FID_MAX_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_MINMAX_SAMP_INT
PMS_FID_MINMAX_POLL_INT
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_INTER_SAMP_DELAY
PMS_FID_INTER_PORT_DELAY

PMS_FID_TRIGGER_MODE
PMS_FID_REPEAT_COUNT
PMS_FID_COMP_TIME
PMS_FID_TARE_TIME
PMS_FID_TRIGGER_DELAY
PMS_FID_SAMPLE_POINT_TYPE
PMS_FID_VOLUME_UNITS Ver 1.5 or later
PMS_FID_VOLUME_TYPE Ver 1.5 or later

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

The CLS-900 sends all sampler/sensor status values, if status is requested.

Field Name: None

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

The CLS-900 sends asynchronous status updates as sampler/sensor status values change (one sampler/sensor per packet), unless status is specifically requested.

Field Names:

PMS_FID_DATE_TIME
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING

Example Instrument-level status types and strings:

PMS_INST_STATUS_ERROR Various Communication Errors
PMS_INST_STATUS_SAMPLING "Sampling"
PMS_INST_STATUS_IDLE "Idle"

Example Sample Point level status types & strings:

PMS_SP_STATUS_ERROR
":Interlock Failure"
":Counldn't establish vacuum"
":Maintenance Mode"
":No LiQuilaz"
":Fill Timeout"
":Drain Timeout"

":Sample Timeout"
":Bad supply pressure"
PMS_SP_STATUS_DISABLED
":Stopped"
":Disabled"
":No Liq"
PMS_SP_STATUS_SAMPLING
":Sampling"
PMS_SP_STATUS_IDLE
":Idle"

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Name: PMS_FID_DATE_TIME

`PMS_PAKT_MSG_DATA` Host ↔ Inst: Data transmission

The CLS-900 sends each sampler/sensor data in its own packet. The number of sample points in each packet is one (this allows the Hosst to process each data packet separately).

Field Names:

PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SAMP_INT
PMS_FID_DATA_TYPE
PMS_FID_VOLUME
PMS_FID_SAMPLE_TIME
PMS_FID_DC_LIGHT
PMS_FID_DATA_VALUES
PMS_FID_LASER_STATUS
PMS_FID_DATA_EXP
PMS_FID_VOL_EXP
PMS_FID_FILL_TIME Ver 1.5 or later

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

The Host sends each sampler/sensor configuration data set in its own message. It is recommended that each message be sent in its own packet.

Field Names:

PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SIZES
PMS_FID_SAMP_INT
PMS_FID_DISABLED
PMS_FID_INTER_SAMP_DELAY
PMS_FID_INTER_PORT_DELAY
PMS_FID_TRIGGER_MODE
PMS_FID_REPEAT_COUNT
PMS_FID_COMP_TIME
PMS_FID_TARE_TIME
PMS_FID_TRIGGER_DELAY

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

The CLS-900 only processes the Start and Stop commands. The CLS-900 will continue to empty the burette, if told to stop during a sample cycle.

Field Name: PMS_FID_COMMAND

Initialization Messages:

Traffic Between the Host and the CLS-900.
Once the Host has established the TCP/IP connection with the instrument an initialization sequence can take place, including the following message traffic:

Host →Sensor: PMS_PAKT_MSG_GET_CONFIG_INFO This message is sent to request a current configuration from the instrument. This will, at the highest level, determine if the instrument is currently connected to another host.

Sensor →Host: PMS_PAKT_MSG_CONFIG_INFO This is the response to the PMS_PAKT_MSG_GET_CONFIG_INFO message. This message informs the host as to the number of sample points currently being run by the instrument (a sample point can be a CLS-920/CLS-930 pair or a Nano sensor). The possible configuration parameters for each sample point are also transmitted.

The message-pair above could be omitted if the availability of the instrument and the number of sample points being processed were ALWAYS known ahead of time.

Host →Sensor: PMS_PAKT_MSG_DATE_TIME This message is sent in order to get data packets from the instrument which are coordinated with system time. The time reported is the start of the sample.

This message need not be sent if the date/time in the data from the instrument is not used.

Host →Sensor: PMS_PAKT_MSG_SET_CONFIG This message is sent for each sample point being processed by the instrument. This message should NOT be omitted. You could set all the sample points Disabled at this point.

Host →Sensor: PMS_PAKT_MSG_CONTROL This message is sent to Start/Stop the instrument as a whole.

Run-Time Processing:

Host →Sensor: PMS_PAKT_MSG_SET_CONFIG This message is sent for each sample point being processed by the instrument. If a sampler is to run the above configuration then enable it. If it is Not to run the above configuration then disable it (field: PMS_FID_DISABLED).

Host →Sensor: PMS_PAKT_MSG_GET_STATUS This message can be sent by the Host in order to verify the state and connection to the instrument.

Sensor →Host: PMS_PAKT_MSG_STATUS This message is sent to the Host in response to a PMS_PAKT_GET_STATUS message. It is also sent to the Host any time the state of any of the attached sample points changes.

CLS-1000 Message Fields

The CLS-1000 presents the host with a single device interface. The sensor portion of the system is communicated to, and controlled by, the CLS-1000. Some of the following messages list the field IDs that will be present, *in addition to sensor fields*. The only sensor currently integrated with the CLS-1000 is the Liquistat®.

WARNING:

Use extreme care when operating any Corrosive Liquid Sampler (CLS) device. This is especially true when remotely processing the Maintenance operating mode.

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

Field Name: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host ←Inst: Configuration information from the instrument

Field Names:

PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Names: None

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

Field Names:

PMS_FID_DATE_TIME
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT

Status Strings and associated Status Codes for the sampler:

Idle: PMS_INST_STATUS_IDLE

Tare: PMS_INST_STATUS_SAMPLING | PMS_INST_STATUS_TARE

Filling:
PMS_INST_STATUS_SAMPLING | PMS_INST_STATUS_FILLING

Compressing:
PMS_INST_STATUS_SAMPLING | PMS_INST_STATUS_COMPRESS

Sampling: PMS_INST_STATUS_SAMPLING

Draining:
PMS_INST_STATUS_SAMPLING | PMS_INST_STATUS_DRAINING

Venting:

PMS_INST_STATUS_SAMPLING | PMS_INST_STATUS_VENTING

Delaying:

PMS_INST_STATUS_SAMPLING | PMS_INST_STATUS_DELAYING);

Purge - Inlet: PMS_INST_STATUS_ERROR

Purge - Overflow: PMS_INST_STATUS_ERROR

Purge - Burette: PMS_INST_STATUS_ERROR

Purge - Sensor: PMS_INST_STATUS_ERROR

System Flush: PMS_INST_STATUS_ERROR

Stopped - Leak: PMS_INST_STATUS_LEAK

Fill Timeout: PMS_INST_STATUS_ERROR

Drain Timeout: PMS_INST_STATUS_ERROR

Stopped - Can't Communicate With Particle Sensor:

PMS_INST_STATUS_ERROR

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Names:

- PMS_PAKT_MSG_DATA
- PMS_FID_FILL_TIME

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

Field Names:

- IN ADDITION TO SENSOR FIELDS
- PMS_FID_TARE_TIME
- PMS_FID_INTER_SAMP_DELAY
- PMS_FID_COMP_TIME
- PMS_FID_TRIGGER_MODE
- PMS_FID_TRIGGER_DELAY
- PMS_FID_OVERFLOW
- PMS_FID_REPEAT_COUNT

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

Field Names:

- PMS_FID_COMMAND
- PMS_CONTROL_START
- PMS_CONTROL_STOP
- PMS_CONTROL_WRITE
- PMS_CONTROL_MAINTENANCE
  - MAINTENANCE_FLUSH 1
  - MAINTENANCE_PURGE_INLET 2
  - MAINTENANCE_PURGE_OVERFLOW 3
  - MAINTENANCE_PURGE_BURETTE 4
  - MAINTENANCE_PURGE_SENSOR 5
  - MAINTENANCE_PURGE_FLUSH_END 6

HSLIS-E Message Fields

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

Field Name: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host ←Inst: Configuration information from the instrument

Field Names:

PMS_FID_INST_FAMILY_NAME
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_MINMAX_NUM_CHANS
PMS_FID_SIZES
PMS_FID_MIN_SIZES
PMS_FID_MAX_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_MINMAX_SAMP_INT
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_DISABLED
PMS_FID_VOLUME_UNITS
PMS_FID_FLOW_RATE
PMS_FID_SSTS
PMS_FID_VOLUME_TYPE
PMS_FID_CONTROL_DATA
PMS_FID_POLL_INT

PMS_FID_SAMPLE_POINT_TYPE
PMS_FID_TARE_TIME
PMS_FID_REPEAT_COUNT
PMS_FID_VOLUME_FACTOR

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Names: None

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

Field Names:

PMS_FID_DATE_TIME
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING
PMS_FID_NUM_SAMPLE_PTS

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Names: PMS_FID_DATE_TIME

`PMS_PAKT_MSG_DATA` Host ↔ Inst: Data transmission

HSLIS-E supports data transmission both to and from the sensor. When data is sent to the HSLIS-E it is interpreted as being the state of the multi-color LED on the sensor. This data consists of a single sample point with a single data value.
If the value = 0, the LED will be green.
If the value = 1, the LED will be orange.
If the value = 2, the LED will be red.

Field Names:

PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SAMP_INT
PMS_FID_DATA_TYPE
PMS_FID_VOLUME
PMS_FID_SAMPLE_TIME
PMS_FID_DATA_VALUES
PMS_FID_LASER_STATUS
PMS_FID_DATA_EXP
PMS_FID_VOL_EXP
PMS_FID_DC_LIGHT

The HSLIS-E transmits a value between 0 and 1 representing the percent of the time during the sample interval that the DC Light was considered bad.

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

Field Names:

PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SSTS
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_DISABLED
PMS_FID_TARE_TIME
PMS_FID_REPEAT_COUNT

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

Field Name: PMS_FID_COMMAND

LASAIR® II Message Fields

The LASAIR II can be set up to run in "Remote Mode". This mode is set from the instrument's front panel and allows for the configuration & data transmission of stored location-tagged data. "Remote Mode" is not compatible with any Manifold system. The normal operating mode is referred to as "Sensor mode" in this document.

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

Field Name: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host ←Inst: Configuration information from the instrument

Field Names:

PMS_FID_INST_FAMILY_NAME
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT

The particle sensor is always Sample Point 0. The six analog sensors are Sample Points 1 through 7. Analog Sample Points do not use all the following fields:

PMS_FID_NUM_CHANS
PMS_FID_MINMAX_NUM_CHANS
PMS_FID_SIZES
PMS_FID_MIN_SIZES
PMS_FID_MAX_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_MINMAX_SAMP_INT
PMS_FID_MINMAX_POLL_INT
PMS_FID_INST_NAME

PMS_FID_INST_VERSION

Sample Point 0 (the particle sensor) reports the instrument serial number for the version field.

PMS_FID_DISABLED

PMS_FID_VOLUME_UNITS

PMS_FID_FLOW_RATE

PMS_FID_SSTS

PMS_FID_VOLUME_TYPE

PMS_FID_CONTROL_DATA

Sensor Mode:

(PMS_NO_LED_CONTROL)

Remote Mode:

(PMS_CONTROL_SAMPLEID)

(PMS_REMOTE_SENSOR)

(PMS_NO_LED_CONTROL)

(PMS_NO_VSI_SUPPORT)

PMS_FID_SAMPLE_POINT_TYPE

One Particle counter & six analog inputs

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Name: PMS_FID_DATE_TIME

This is NOT the current date/time. This is the date/time of the last data packet saved by the host. After a network outage, the instrument will re-send up to 60 packets of data that were time-tagged after this date/time. This date/time field is only used in LASAIR II firmware V2.0 and later.

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

Field Names:

PMS_FID_DATE_TIME

PMS_FID_STATUS_CODE

PMS_FID_STATUS_STRING

PMS_FID_NUM_SAMPLE_PTS

PMS_FID_SAMPLE_POINT

Since all Sample Points on the device have the same status, only the status for Sample Point 0 is included in the message even though the number of Sample Points is registered as 7.

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Name: PMS_FID_DATE_VALUES

`PMS_PAKT_MSG_DATA` Host ←Inst: Data transmission

The LASAIR II size channel data is accumulative.

Field Names:

PMS_FID_NUM_SAMPLE_PTS

PMS_FID_SAMPLE_POINT

PMS_FID_NUM_CHANS

The analog Sample Points do not use all of the following fields. The 6 analog inputs are stored in the first element of the Data Value array of each analog Sample Point.

PMS_FID_SAMP_INT

PMS_FID_DATA_TYPE

PMS_FID_VOLUME

PMS_FID_SAMPLE_TIME

PMS_FID_DATA_VALUES

PMS_FID_LASER_STATUS

There are multiple fields in the status field. The L2-310/510 series of instruments report laser good and flow good. The LASAIR II-110 instrument reports the same data and a coincidence limit flag.

PMS_FID_DATA_EXP

PMS_FID_VOL_EXP

PMS_FID_DC_LIGHT

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

The LASAIR II stops sampling when this message is received.

Field Names:

PMS_FID_SAMPLE_POINT

PMS_FID_SAMP_INT

PMS_FID_MAX_SAMP_INT

PMS_FID_DISABLED

PMS_FID_SSTS

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

LASAIR II accepts the normal start sampling command, if not in Remote mode. Conversely, the LASAIR II will only accept a Data command (either Dump or Delete) if in Remote mode.

Field Names:

- PMS_FID_COMMAND
  - Sensor Mode:
    - PMS_CONTROL_STOP
    - PMS_CONTROL_START
      - PMS_SUBCONTROL_START_DEFAULT
      - PMS_SUBCONTROL_START_PUMP_ONLY
  - Remote Mode:
    - PMS_CONTROL_DATA
    - PMS_SUBCONTROL_DATA_DUMP
    - PMS_SUBCONTROL_DATA_DELETE

`PMS_PAKT_MSG_STRING` Host ↔ Inst: Remote Location

The LASAIR II will process the string command in the Remote processing mode. The command is used to request and deliver the list of available locations stored in the device. The location request is made by transmitting "command: get locations". The response will begin with "response: get locations" and each subsequent "\n" character will delimit a location name in the list. This message is not available in the UDP interface.

Field Names:

- PMS_FID_SAMPLE_POINT
- PMS_FID_MESSAGE_STR_LEN
- PMS_FID_MESSAGE_STRING

IsoAir® PLUS Message Fields

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

Field Name: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host ←Inst: Configuration information from the instrument

Field Names:

PMS_FID_INST_FAMILY_NAME
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT

The particle sensor is always Sample Point 0. The four analog sensors are Sample Points 1 through 4. The analog Sample Points do not use all of the following fields.

PMS_FID_NUM_CHANS
PMS_FID_MINMAX_NUM_CHANS
PMS_FID_SIZES
PMS_FID_MIN_SIZES
PMS_FID_MAX_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_MINMAX_SAMP_INT
PMS_FID_MINMAX_POLL_INT
PMS_FID_INST_NAME
PMS_FID_INST_VERSION

Sample Point 0 (the particle sensor) reports the instrument serial number for the version field.

PMS_FID_DISABLED
PMS_FID_VOLUME_UNITS

PMS_FID_FLOW_RATE
PMS_FID_SSTS
PMS_FID_VOLUME_TYPE
PMS_FID_CONTROL_DATA
PMS_FID_SAMPLE_POINT_TYPE

One Particle counter & four analog inputs

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Name: PMS_FID_DATE_TIME

This is NOT the current date/time This is the date/time of the last data packet saved by the host. After a network outage, the instrument re-sends up to 60 packets of data that were time-tagged after this date/time. This date/time field is only used in IsoAirPLUS firmware V2.0 and later (not released as of this writing).

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

Field Names:

PMS_FID_DATE_TIME
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT

Since all Sample Points on the device have the same status, only the status for Sample Point 0 is included in the message even though the number of Sample Points is registered as 5.

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Name: PMS_FID_DATE_VALUES

`PMS_PAKT_MSG_DATA` Host ←Inst: Data transmission

The IsoAir PLUS size channel data is accumulative.

Field Names:

PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS

The analog Sample Points do not use all of the following fields. The 4 analog inputs are stored in the first element of the Data Value array of each analog Sample Point.

PMS_FID_SAMP_INT
PMS_FID_DATA_TYPE
PMS_FID_VOLUME
PMS_FID_SAMPLE_TIME
PMS_FID_DATA_VALUES
PMS_FID_LASER_STATUS
PMS_FID_DATA_EXP
PMS_FID_VOL_EXP
PMS_FID_DC_LIGHT

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

The IsoAir PLUS stops sampling when this message is received.

Field Names:

PMS_FID_SAMPLE_POINT
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_DISABLED
PMS_FID_SSTS

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

The IsoAir PLUS will accept the normal start sampling command.

Field Names:

PMS_FID_COMMAND
Sensor Mode:
PMS_CONTROL_STOP
PMS_CONTROL_START
PMS_SUBCONTROL_START_DEFAULT
PMS_SUBCONTROL_START_PUMP_ONLY

`PMS_PAKT_MSG_STRING` Host ↔ Inst: Remote Location

Liquistat® Message Fields

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

Field Names: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host →Inst: Configuration information from the instrument

The Liquistat has an adjustable first channel threshold as indicated by the difference in the Min/Max size array data. The appropriate interpretation is described in the PMS_FLDS.H file.

Field Names:

PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_MINMAX_NUM_CHANS
PMS_FID_SIZES
PMS_FID_MIN_SIZES
PMS_FID_MAX_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_MINMAX_SAMP_INT
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_DISABLED
PMS_FID_VOLUME_UNITS
PMS_FID_FLOW_RATE
PMS_FID_SSTS
PMS_FID_VOLUME_TYPE

PMS_FID_CONTROL_DATA
PMS_FID_POLL_INT

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Names: None

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

In addition to the standard status indications, the Liquistat can send a Leak indication as listed in the PMS_FLDS.H file.

Field Names:

PMS_FID_DATE_TIME
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Name: PMS_FID_DATE_TIME

`PMS_PAKT_MSG_DATA` Host ↔ Inst: Data transmission

The Liquistat size channel data is accumulative.

Field Names:

PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SAMP_INT
PMS_FID_DATA_TYPE
PMS_FID_VOLUME
PMS_FID_SAMPLE_TIME
PMS_FID_DATA_VALUES
PMS_FID_LASER_STATUS
PMS_FID_DATA_EXP
PMS_FID_VOL_EXP
PMS_FID_DC_LIGHT

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

Field Names:

PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SSTS
PMS_FID_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_DISABLED

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

Field Name: PMS_FID_COMMAND

Aerosol Manifold II Message Fields

The Aerosol Manifold II presents the host with a single device interface. The sensor portion of the system is communicated to, and controlled by, the Manifold II. Some of the following messages list the field IDs that will be present *in addition to sensor fields*.

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

This message is processed if the device is not under host control or if it is specified to answer even if used.

Field Name: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host ←Inst: configuration information from the instrument

Sensor Field Names:

PMS_FID_INST_FAMILY_NAME
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS

Other than Sensor Field names:

PMS_FID_NUM_MANIFOLD_POS
PMS_FID_CONTROL_DATA
PMS_NO_LED_CONTROL) (PMS_NO_VSI_SUPPORT)
PMS_FID_MANIFOLD_SEQ
PMS_FID_TARE_TIME

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Names: None

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

Non-sensor Field Names:

PMS_FID_STATUS_CODE

In addition to the normal SAMPLING indication, the following flags can be included.

PMS_INST_STATUS_DELAYING

PMS_INST_STATUS_TARE

PMS_FID_STATUS_STRING

The status string identifies Delaying and Tare as separate sampling states.

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Non-sensor Field Name:

PMS_FID_DATE_TIME

`PMS_PAKT_MSG_DATA` Host ←Inst: Data transmission

Non-sensor Field Name:

PMS_FID_MANIFOLD_POS

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

Non-sensor Field Names:

PMS_FID_MANIFOLD_SEQ

PMS_FID_TARE_TIME

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

Field Names:

PMS_FID_COMMAND

PMS_CONTROL_START

PMS_CONTROL_STOP

MiniNet® Message Fields

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

Field Names: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host ←Inst: Configuration information from the instrument

Field Names:

PMS_FID_INST_FAMILY_NAME
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT

The particle sensor is always Sample Point 0. The three 4-20 mA analog inputs are Sample Points 1 through 3. The analog Sample Points do not use all of the following fields.

PMS_FID_NUM_CHANS
PMS_FID_MINMAX_NUM_CHANS
PMS_FID_SIZES
PMS_FID_MIN_SIZES
PMS_FID_MAX_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_MINMAX_SAMP_INT
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_DISABLED
PMS_FID_VOLUME_UNITS
PMS_FID_FLOW_RATE
PMS_FID_SSTS
PMS_FID_VOLUME_TYPE

PMS_FID_CONTROL_DATA
PMS_FID_POLL_INT
PMS_FID_SAMPLE_POINT_TYPE
One Particle counter & three analog inputs
PMS_FID_TARE_TIME
PMS_FID_REPEAT_COUNT

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Names: None

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

Field Names:
PMS_FID_DATE_TIME
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Name: PMS_FID_DATE_TIME

`PMS_PAKT_MSG_DATA` Host ↔ Inst: Data transmission

MiniNet supports data transmission both to and from the sensor. When data is sent to the MiniNet, it is interpreted as being the state of the multi-color LED on the sensor. This data consists of a single sample point with a single data value.
If the value = 0, then the LED will be green.
If the value = 1, the LED will be orange.
If the value = 2, the LED will be red.

Field Names:
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SAMP_INT
PMS_FID_DATA_TYPE
PMS_FID_VOLUME
PMS_FID_SAMPLE_TIME

PMS_FID_DATA_VALUES
PMS_FID_LASER_STATUS
PMS_FID_DATA_EXP
PMS_FID_VOL_EXP

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

Field Names:

PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_DISABLED
PMS_FID_TARE_TIME
PMS_FID_REPEAT_COUNT

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

Field Name: PMS_FID_COMMAND

Ultra DI® Message Fields

`PMS_PAKT_MSG_RESET` Host →Inst: Reset Instrument

Field Names: None

`PMS_PAKT_MSG_GET_CONFIG_INFO` Host →Inst: Get configuration information from the instrument

Field Name: PMS_FID_IN_USE_FLAG

`PMS_PAKT_MSG_CONFIG_INFO` Host →Inst: Configuration information from the instrument

Field Names:

PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_ADAPT_ADDR
PMS_FID_IN_USE_FLAG
PMS_FID_USER_ADDR
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_MINMAX_NUM_CHANS
PMS_FID_SIZES
PMS_FID_MIN_SIZES
PMS_FID_MAX_SIZES
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_MINMAX_SAMP_INT
PMS_FID_INST_NAME
PMS_FID_INST_VERSION
PMS_FID_DISABLED
PMS_FID_VOLUME_UNITS
PMS_FID_FLOW_RATE
PMS_FID_SSTS
PMS_FID_VOLUME_TYPE
PMS_FID_CONTROL_DATA
PMS_FID_POLL_INT

`PMS_PAKT_MSG_GET_STATUS` Host →Inst: Get instrument status

Field Names: None

`PMS_PAKT_MSG_STATUS` Host ←Inst: Instrument status to host

Field Names:

PMS_FID_DATE_TIME
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING
PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_STATUS_CODE
PMS_FID_STATUS_STRING

`PMS_PAKT_MSG_DATE_TIME` Host →Inst: Set date/time

Field Name: PMS_FID_DATE_TIME

`PMS_PAKT_MSG_DATA` Host ↔ Inst: Data transmission

The Ultra-DI size channel data is accumulative.

Field Names:

PMS_FID_NUM_SAMPLE_PTS
PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SAMP_INT
PMS_FID_DATA_TYPE
PMS_FID_VOLUME
PMS_FID_SAMPLE_TIME
PMS_FID_DATA_VALUES
PMS_FID_LASER_STATUS
PMS_FID_DATA_EXP
PMS_FID_VOL_EXP
PMS_FID_DC_LIGHT

`PMS_PAKT_MSG_SET_CONFIG` Host →Inst: Set instrument configuration

Field Names:

PMS_FID_SAMPLE_POINT
PMS_FID_NUM_CHANS
PMS_FID_SSTS
PMS_FID_SAMP_INT
PMS_FID_MAX_SAMP_INT
PMS_FID_DISABLED

`PMS_PAKT_MSG_CONTROL` Host →Inst: Control (start, stop, etc.)

Field Name: PMS_FID_COMMAND

Index