HEALTH AND SAFETY

Exposure Monitoring

Model 8530/32/33/34

DUSTTRAK™ II and DRX Aerosol Monitors

Communication Manual

P/N 6002481, Revision G
January 2012

TRUST. SCIENCE. INNOVATION.

Address
TSI Incorporated / 500 Cardigan Road / Shoreview, MN 55126 / USA

Fax No.
(651) 490-3824

Limitation of Warranty and Liability (effective January 2012)

(For country-specific terms and conditions outside of the USA, please visit www.tsi.com.)

Seller warrants the goods sold hereunder, under normal use and service as described in the operator's manual, shall be free from defects in workmanship and material for twenty-four (24) months, or if less, the length of time specified in the operator's manual, from the date of shipment to the customer. This warranty period is inclusive of any statutory warranty. This limited warranty is subject to the following exclusions and exceptions:

a. Hot-wire or hot-film sensors used with research anemometers, and certain other components when indicated in specifications, are warranted for 90 days from the date of shipment;
b. DUSTTRAK internal pump for Models 8530 and 8533 is warranted for two (2) years or 4000 hours, whichever comes first;
c. DUSTTRAK internal pump for Models 8530 and 8533 is warranted for operation within ambient temperatures between 5–45°C. Warranty is void when the internal pump is operating outside of this temperature range;
d. Parts repaired or replaced as a result of repair services are warranted to be free from defects in workmanship and material, under normal use, for 90 days from the date of shipment;
e. Seller does not provide any warranty on finished goods manufactured by others or on any fuses, batteries or other consumable materials. Only the original manufacturer's warranty applies;
f. Unless specifically authorized in a separate writing by Seller, Seller makes no warranty with respect to, and shall have no liability in connection with, goods which are incorporated into other products or equipment, or which are modified by any person other than Seller.

The foregoing is **IN LIEU OF** all other warranties and is subject to the **LIMITATIONS** stated herein. **NO OTHER EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR PARTICULAR PURPOSE OR MERCHANTABILITY IS MADE**. **WITH RESPECT TO SELLER'S BREACH OF THE IMPLIED WARRANTY AGAINST INFRINGEMENT, SAID WARRANTY IS LIMITED TO CLAIMS OF DIRECT INFRINGEMENT AND EXCLUDES CLAIMS OF CONTRIBUTORY OR INDUCED INFRINGEMENTS. BUYER'S EXCLUSIVE REMEDY SHALL BE THE RETURN OF THE PURCHASE PRICE DISCOUNTED FOR REASONABLE WEAR AND TEAR OR AT SELLER'S OPTION REPLACEMENT OF THE GOODS WITH NON-INFRINGING GOODS.**

TO THE EXTENT PERMITTED BY LAW, THE EXCLUSIVE REMEDY OF THE USER OR BUYER, AND THE LIMIT OF SELLER'S LIABILITY FOR ANY AND ALL LOSSES, INJURIES, OR DAMAGES CONCERNING THE GOODS (INCLUDING CLAIMS BASED ON CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR OTHERWISE) SHALL BE THE RETURN OF GOODS TO SELLER AND THE REFUND OF THE PURCHASE PRICE, OR, AT THE OPTION OF SELLER, THE REPAIR OR REPLACEMENT OF THE GOODS. IN THE CASE OF SOFTWARE, SELLER WILL REPAIR OR REPLACE DEFECTIVE SOFTWARE OR IF UNABLE TO DO SO, WILL REFUND THE PURCHASE PRICE OF THE SOFTWARE. IN NO EVENT SHALL SELLER BE LIABLE FOR LOST PROFITS OR ANY SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES. SELLER SHALL NOT BE RESPONSIBLE FOR INSTALLATION, DISMANTLING OR REINSTALLATION COSTS OR CHARGES. No Action, regardless of form, may be brought against Seller more than 12 months after a cause of action has accrued. The

goods returned under warranty to Seller's factory shall be at Buyer's risk of loss, and will be returned, if at all, at Seller's risk of loss.

Buyer and all users are deemed to have accepted this LIMITATION OF WARRANTY AND LIABILITY, which contains the complete and exclusive limited warranty of Seller. This LIMITATION OF WARRANTY AND LIABILITY may not be amended, modified or its terms waived, except by writing signed by an Officer of Seller.

Service Policy

Knowing that inoperative or defective instruments are as detrimental to TSI as they are to our customers, our service policy is designed to give prompt attention to any problems. If any malfunction is discovered, please contact your nearest sales office or representative, or call TSI at (800) 874-2811 (USA) or (001 651) 490-2811 (International).

CONTENTS

(page intentionally left blank)

Chapter 1

Instrument Communication Methods

Desktop Basic Model	**8530**
Handheld Basic Model	**8532**
Desktop DRX Model	**8533**
Handheld DRX Model	**8534**

Connecting to the Instrument

USB and Ethernet instrument connections use the NDIS communication method, port 3602.

Remote radio communications use the RS-232 serial communication method.

NIDS Communication Method

This method communicates with the DUSTTRAK™ II and DUSTTRAK™ DRX instruments via USB and Ethernet connections. This method uses windows sockets.

RS-232 Serial Communication Method

This method communicates with the DUSTTRAK™ II and DUSTTRAK™ DRX instruments via host to remote radio communication.

RS-232 Method Supports TSI Radio Accessory

801820	922 MHz modem with antenna mount kit for enclosure
801821	922 MHz modem
801825	2.4 GHz modem with antenna mount kit for enclosure
801826	2.4 GHz modem

The serial settings are as follows:

Baud Rate:	9600
Flow Control:	NONE
Data Bits:	8
Parity:	NONE
Stop Bits:	1

(page intentionally left blank)

Chapter 2

Instrument Command Set

Read the Model of the Instrument

Command: **RDMN**

Return value: a string containing the model of the instrument.

Code Example:

```
// "RDMN" is the DustTrak command to read the model of the instrument.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RDMN\r", 5);

// Reply
// buff = a string containing the model of the instrument (i.e. 8530).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read the Serial Number of the Instrument

Command: **RDSN**

Return value: a string containing the serial number of the instrument.

Code Example:

```
// "RDSN" is the DustTrak command to read the serial number of the instrument.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RDSN\r", 5);

// The returned value is a string containing the serial number of the
//   instrument (i.e. 8530083001).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read the Firmware Version of the Instrument

Command: **RDBS**

Return value: a string containing the firmware version of the instrument.

Code Example:

```
// "RDBS" is the DustTrak command to read the firmware version of the
//    instrument.
// The command must be followed by a CR.
m_pSocket->Send( "RDBS\r", 5);


// The returned value is a string containing the firmware version of
//   the instrument (i.e. 1.0).
m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read the Current User Polling Status of the Instrument

Command: **MSTATUS**

Return value: a string containing the user polling status of the instrument.

Code Example:

```
// "MSTATUS" is the DustTrak command to read the current user polling
//   status of the instrument.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "MSTATUS\r", 8);

// The returned value is a string containing the user polling status of
//   the instrument (i.e. Idle, Waiting, Running, etc.).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Start the Instrument Measurement

Command: **MSTART**

Return value: a response OK indicating that the instrument started, or FAIL indicating an error starting the measurement.

Code Example:

```
// "MSTART" is the DustTrak command to start the instrument measurement.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "MSTART\r", 7);

 // The returned value is a string containing status
//   of the command (i.e. OK or FAIL).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Stop the Instrument Measurement

Command: **MSTOP**

Return value: a string response OK indicating that the instrument is stopping, or FAIL indicating an error stopping the measurement.

Code Example:

```
// "MSTOP" is the DustTrak command to stop the instrument measurement.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "MSTOP\r", 6);

 // The returned value is a string containing status
//   of the command (i.e. OK or FAIL).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Update Instrument RAM

Command: **MUPDATE**

Return value: a string response OK indicating that the instrument RAM was updated, or FAIL indicating an error during the RAM update.

Code Example:

```
// "MUPDATE" is the DustTrak command to update the instrument RAM.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "MUPDATE\r", 8);

// The returned value is a string containing status
//   of the command (i.e. OK or FAIL).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Shutdown the Instrument

Command: **MSHUTDOWN**

Return value: a string response OK indicating that the instrument is shutting down, or FAIL indicating an error shutting down the instrument

Code Example:

```
// "MSHUTDOWN" is the DustTrak command to shut down the instrument.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "MSHUTDOWN\r", 10);

 // The returned value is a string containing status
//   of the command (i.e. OK or FAIL).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read the Current Measurements of the Instrument

Command: **RMMEAS**

Return value: a comma separated string containing the following data format based on the instrument model.

All Basic Models:
Current second of test, Aerosol measurement,

All DRX Models:
Current second of test, PM_1 channel measurement, $PM_{2.5}$ channel measurement, PM_4 *(RESP)* channel measurement, PM_{10} channel measurement, Total channel measurement,

Code Example:

```
// "RMMEAS" is the DustTrak command to read the current measurements
//    of the instrument.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMMEAS\r", 7);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
 // DRX Reply
// buff = "10,0.023,0.024,0.123,0.156,0.179,"
//  10 = Current second of the sample time
//  0.023 = PM1 measurement
//  0.024 = PM2.5 measurement
//  0.123 = Resp (PM4) measurement
//  0.156 = PM10 measurement
//  0.179 = Total measurement
// All measurements in mg/m3

// DTII  Reply
// buff = "10,0.024,"
//  10 = Current second of the sample time
//  0.024 = Mass measurement in mg/m3
```

Read the Current Measurements and Statistics of the Instrument

Command: **RMMEASSTATS**

Return value: a comma separated string containing the following data format based on the instrument model.

All Basic Models:
Current second of test, Aerosol measurement, Aerosol minimum, Aerosol maximum, Aerosol average, Aerosol TWA *(time weighted average)*,

All DRX Models:
Current second of test, PM_1 channel measurement, PM_1 channel minimum, PM_1 channel maximum, PM_1 channel average, PM_1 channel TWA *(time weighted average)*, $PM_{2.5}$ channel measurement, $PM_{2.5}$ channel minimum, $PM_{2.5}$ channel maximum, $PM_{2.5}$ channel average, $PM_{2.5}$ channel TWA *(time weighted average)*, PM_4 *(RESP)* channel measurement, PM_4 *(RESP)* channel minimum, PM_4 *(RESP)* channel maximum, PM_4 *(RESP)* channel average, PM_4 *(RESP)* channel TWA *(time weighted average)*,PM_{10} channel measurement, PM_{10} channel minimum, PM_{10} channel maximum, PM_{10} channel average, PM_{10} channel TWA *(time weighted average)*, Total channel measurement, Total channel minimum, Total channel maximum, Total channel average, Total channel TWA *(time weighted average)*,

Code Example:

```
// "RMMEASSTATS" is the DustTrak command to read the current
//   measurements of the instrument.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMMEASSTATS\r", 12);

iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// DRX Reply
// buff = "10,0.023,0.012,0.028,0.022,0.000,
//          0.024,0.016,0.027,0.025,0.000,
//          0.123,0.120,0.153,0.145,0.000,
//          0.156,0.125,0.187,0.166,0.000,
//          0.179,0.120,0.190,0.180,0.000,"
//  10 = Current second of the sample time
//  0.023 = PM1 current measurement
//  0.012 = PM1 Minimum measurement
//  0.028 = PM1 Maximum measurement
//  0.022 = PM1 Avg calculation
//  0.000 = PM1 TWA calculation
//  0.024 = PM2.5 measurement
//  0.016 = PM2.5 Minimum measurement
//  0.027 = PM2.5 Maximum measurement
//  0.025 = PM2.5 Avg calculation
//  0.000 = PM2.5 TWA calculation
//  0.123 = Resp (PM4) measurement
//  0.120 = Resp (PM4) Minimum measurement
```

```
//  0.153 = Resp (PM4) Maximum measurement
//  0.145 = Resp (PM4) Avg calculation
//  0.000 = Resp (PM4) TWA calculation
//  0.156 = PM10 measurement
//  0.125 = PM10 Minimum measurement
//  0.187 = PM10 Maximum measurement
//  0.166 = PM10 Avg calculation
//  0.000 = PM10 TWA calculation
//  0.179 = Total measurement
//  0.120 = Total Minimum measurement
//  0.190 = Total Maximum measurement
//  0.180 = Total Avg calculation
//  0.000 = Total TWA calculation
// All measurements in mg/m3

// DTII  Reply
// buff = "10,0.179,0.120,0.190,0.180,0.000,"
//  10 = Current second of the sample time
//  0.179 = Mass measurement
//  0.120 = Mass Minimum measurement
//  0.190 = Mass Maximum measurement
//  0.180 = Mass Avg calculation
//  0.000 = Mass TWA calculation
// All measurements in mg/m3
```

Read the Latest Measurement Logged into Memory

Command: **RMLOGGEDMEAS**

Return value: a comma separated string containing the following data format based on the instrument model.

All Basic Models:
Current second of logged data, Aerosol measurement,

All DRX Models:
Current second of logged data, PM_1 channel measurement, $PM_{2.5}$ channel measurement, PM_4 *(RESP)* channel measurement, PM_{10} channel measurement, Total channel measurement,

Code Example:

```
// "RMLOGGEDMEAS" is the DustTrak command to read the last logged data.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMLOGGEDMEAS\r", 13);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// DRX Reply
// buff = "2,1,0.000,1.000,2.000,3.000,4.000"
//  2     = Time elapsed
//  0.000 = PM1
//  1.000 = PM2.5
//  2.000 = PM4
//  3.000 = PM10
//  4.000 = TOTAL

// DTII Reply
// buff = "2,0.000"
//  2     = Time elapsed
//  0.000 = Total
```

Read the Fault Messages of the Instrument

Command: **RMMESSAGES**

Return value: a comma separated string containing the following integer data format based on the instrument model.

Basic Desktop Model:
System Error, Laser Error, Flow Error, Flow Blocked, Max Concentration Total, STEL Alarmed, Filter Concentration Error, Battery Installed , Battery Charging, Battery Percentage available, Battery Low Error, Memory Percentage available, Memory Low Error,

Basic Handheld Model:
System Error, Laser Error, Flow Error, Flow Blocked, Max Concentration Total, Filter Concentration Error, Battery Installed , Battery Charging, Battery Percentage available, Battery Low Error, Memory Percentage available, Memory Low Error,

DRX Desktop Model:
System Error, Laser Error, Flow Error, Flow Blocked, Max Concentration PM_1, Max Concentration $PM_{2.5}$, Max Concentration PM_4 *(RESP)*, Max Concentration PM_{10}, Max Concentration Total, STEL Alarmed, Filter Concentration Error, Battery Installed , Battery Charging, Battery Percentage available, Battery Low Error, Memory Percentage available, Memory Low Error,

DRX Handheld Model:
System Error, Laser Error, Flow Error, Flow Blocked, Max Concentration PM_1, Max Concentration $PM_{2.5}$, Max Concentration PM_4 *(RESP)*, Max Concentration PM_{10}, Max Concentration Total, Filter Concentration Error, Battery Installed , Battery Charging, Battery Percentage available, Battery Low Error, Memory Percentage available, Memory Low Error,

Code Example:

```
// "RMMESSAGES" is the DustTrak command to read the fault messages
//   of the instrument.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMMESSAGES\r", 11);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// DRX Desktop Reply
// buff = "0,1,1,0,1,0,1,0,1,0,0,1,0,80,0,90,0,"
//  0  = No System Error
//  1  = Laser Error
//  1  = Flow Error
//  0  = No Flow Blocked error
//  1  = Maximum Concentration PM1
//  0  = No Maximum Concentration PM2.5
//  1  = Maximum Concentration Resp
//  0  = No Maximum Concentration PM10
//  1  = Maximum Concentration Total
//  0  = No STEL Alarmed
//  0  = Filter Concentration error
//  1  =  Battery Installed
//  0  = Battery Not Charging
//  80 = Battery Percentage Available
//  0  = No Battery Low error
//  90 = Memory Percentage Available
//  0  = No Memory Low error

// DRX Handheld Reply
// buff = "0,1,1,0,1,0,1,0,1,0,80,0,90,0,"
//  0  = No System Error
//  1  = Laser Error
//  1  = Flow Error
//  0  = No Flow Blocked error
//  1  = Maximum Concentration PM1
//  0  = No Maximum Concentration PM2.5
//  1  = Maximum Concentration Resp
//  0  = No Maximum Concentration PM10
//  1  = Maximum Concentration Total
//  0  = Filter Concentration error
//  1  = Battery Installed
//  0  = Battery Not Charging
//  80 = Battery Percentage Available
//  0  = No Battery Low error
//  90 = Memory Percentage Available
//  0  = No Memory Low error
```

```
// DTII Desktop Reply
// buff = "0,1,1,0,1,0,0,1,0,80,0,90,0,"
//  0  = No System Error
//  1  = Laser Error
//  1  = Flow Error
//  0  = No Flow Blocked error
//  1  = Maximum Concentration
//  0  = No STEL Alarmed
//  0  = Filter Concentration error
//  1  = Battery Installed
//  0  = Battery Not Charging
//  80 = Battery Percentage Available
//  0  = No Battery Low error
//  90 = Memory Percentage Available
//  0  = No Memory Low error

// DTII Handheld Reply
// buff = "0,1,1,0,1,0,0,1,0,80,0,90,0,"
//  0  = No System Error
//  1  = Laser Error
//  1  = Flow Error
//  0  = No Flow Blocked error
//  1  = Maximum Concentration
//  0  = Filter Concentration error
//  1  = Battery Installed
//  0  = Battery Not Charging
//  80 = Battery Percentage Available
//  0  = No Battery Low error
//  90 = Memory Percentage Available
//  0  = No Memory Low error
```

Read the Alarm Messages of the Instrument

Command: **RMALARM**

Return value: a comma separated string containing the following integer data format based on the instrument model.

All Basic Models:
Mass Alarm1
Mass Alarm2

All DRX Models:
PM1 Alarm1,PM2.5 Alarm1,Resp Alarm1,PM10 Alarm1,Total Alarm1
PM1 Alarm2,PM2.5 Alarm2,Resp Alarm2,PM10 Alarm2,Total Alarm2

Code Example:

```
// "RMALARM" is the DustTrak command to read the alarm messages
//   of the instrument.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMALARM\r", 8);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// DRX Reply
// buff = "0,1,1,0,0,1,0,1,0,1"
//  0  = No Alarm1 in PM1
//  1  = Alarm1 in PM2.5
//  1  = Alarm1 in Resp
//  0  = No Alarm1 in PM10
//  0  = No Alarm1 in Total
//  1  = Alarm2 in PM1
//  0  = No Alarm2 in PM2.5
//  1  = Alarm2 in Resp
//  0  = No Alarm2 in PM10
//  1  = Alarm2 in Total

// Basic Reply
// buff = "0,1"
//  0  = No Alarm1
//  1  = Alarm2
```

Read the State of Data Logging

Command: **RMLOGINFO**

Return value: a comma separated string containing the following data format.

Log Name, Log Error, Total Time, Time Elapsed, Time Remaining, Current Test, Total Tests,

Value formats:

Log Name:	String; Logging test name
Log Error:	Integer; indicates logging error
Total Time:	Integer; in seconds
Time Elapsed:	Integer; in seconds
Time Remaining:	Integer; in seconds
Current Test:	Integer; the number of the test that is running
Total Tests:	Integer; the total number of tests being executed.

The following logging error corresponds to the following integers:

OK:	0
Number of tests is greater than max allowed:	1
Start Time for test has elapsed:	2
Number of data points is greater than max allowed:	3
Logging Interval is to short:	4

Code Example:

```
// "RMLOGINFO" is the DustTrak command to read the state of data logging.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMLOGINFO\r", 10);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// Reply
// buff = "LOG MODE 1_001,0,60,50,10,2,3"
//  LOG MODE 1_001 = Log Mode Name and current run number
//  0  = Log Error
//  60 = Total Time in seconds
//  50 = Time Elapsed
//  10 = Time Remaining
//  2  = Current Test
//  3  = Total Tests
```

Read Instrument Logging Mode

Command: **RMODELOG<<#>>**
is a value 0-6 corresponding to the logging program number.
(0 = Survey, 1 = Manual, 2-6 = Program 1-5)

Return value: a comma separated string containing the following data format.

Start Time, Start Date, Interval, Test Length, Number of Test, Time Between Tests, Time Constant, Use Start Time, Use Start Date, Auto Zero Interval, Auto Zero Enable, Program Name

Return value formats:

Start Time:	Hour:Minute:Second
Start Date:	Month/Day/Year
Interval:	Minute:Second
Test Length:	Day:Hour:Minute
Number of Test:	Integer
Time Between Tests:	Day:Hour:Minute
Time Constant	Integer
Use Start Time	Integer; 1 to use the start time, 0 to not use the start time
Use Start Date	Integer; 1 to use the start date, 0 to not use the start date
Auto Zero Interval	Hour:Minute
Auto Zero Enabled	Integer; 1 enabled, 0 disabled
Program Name	char [12]

Code Example:

```
// "RMODELOG#" is the DustTrak command to read an instrument logging
//   mode.
// The command must be followed by a CR.
// Command
// buff = "RMODELOG2"
//  2   = Log Mode index
iRet = m_pSocket->Send( "RMODELOG2\r", 10);
```

```
 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// Reply
// buff = "12:15:0,09/30/2008,0:1,0:2:0,3,0:0:1,5,0,0,0:15,0,LOG MODE 1"
//  12:15:0 = Start Time
//  09/30/2008 = Start Date
//  0:1   = Interval
//  0:2:0 = Test Length
//  3     = Number of Tests
//  0:0:1 = Time Between Tests
//  5     = Time Constant
//  0     = Use Start Time
//  0     = Use Start Date
//  0:15  = Auto Zero Interval (unused on a Handheld model)
//  0     = Auto Zero Enable (unused on a handheld model)
//  LOG MODE 1 = Log Mode Name
```

Write Instrument Logging Mode

Command: **WMODELOG**<<**#**>> <<**Start Time, Start Date, Interval, Test Length, Number of test, Time Between tests, Time Constant, Use Start Time, Use Start Date, Auto Zero Interval, Auto Zero Enable, Test Name**>>

Where # is a value 0–6 corresponding to the logging program number.
(0 = Survey, 1 = Manual, 2-6 = Program 1–5)

Value formats:

Start Time:	Hour:Minute:Second
Start Date:	Month/Day/Year
Interval:	Minute:Second
Test Length:	Day:Hour:Minute
Number of Test:	Integer
Time Between Tests:	Day:Hour:Minute
Time Constant	Integer
Use Start Time	Integer; 1 to use the start time, 0 to not use the start time
Use Start Date	Integer; 1 to use the start date, 0 to not use the start date
Auto Zero Interval	Hour:Minute
Auto Zero Enabled	Integer; 1 enabled, 0 disabled
Program Name	char[12]

Return value: a string response OK indicating that the logging program was written to the instrument, or FAIL indicating an error during the write of the logging program.

***Note**: This command requires the MUPDATE command to be sent to update the instrument RAM.*

Survey Mode: Only Time Constant will change

Manual Mode: Only Interval, Test Length, and Time Constant will change

Code Example:

```
    // "WMODELOG#" is the DustTrak command to write the instrument logging
    //   mode.
    // The command must be followed by a CR.
    // Command
    // buff = "WMODELOG2 12:15:0,09/30/2008,0:1,0:2:0,3,0:0:1,5,0,0,0:15,0,LOG
MODE 1"
    //  2     = Log Mode index (2 is the first log mode)
    //  12:15:0 = Start Time
    //  09/30/2008 = Start Date
    //  0:1   = Interval
    //  0:2:0 = Test Length
    //  3     = Number of Tests
    //  0:0:1 = Time Between Tests
    //  5     = Time Constant
    //  0     = Use Start Time
    //  0     = Use Start Date
    //  0:15  = Auto Zero Interval (unused on a Handheld model)
    //  0     = Auto Zero Enable (unused on a handheld model)
    //  LOG MODE 1 = Log Mode Name
    iRet = m_pSocket->Send( "WMODELOG2
12:15:0,09/30/2008,0:1,0:2:0,3,0:0:1,5,0,0,0:15,0,LOG MODE 1\r", 71);

     // The returned value is a string containing status
    //   of the command (i.e. OK or FAIL).
    iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read the Current Instrument Logging Mode

Command: **RMODECURLOG**

Return value: an integer that corresponds to a logging program.

The following logging program corresponds to the following integers:

SURVEY:	0
MANUAL:	1
PROGRAM_1:	2
PROGRAM_2:	3
PROGRAM_3:	4
PROGRAM_4:	5
PROGRAM_5:	6

Code Example:

```
// "RMODECURLOG" is the DustTrak command to read the current instrument
//   logging mode.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMODECURLOG\r", 12);

//Reply
//buff = “4”
//Instrument currently in Program log #3
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Set the Current Instrument Logging Mode

Command: **WMODECURLOG**<<#>>

Where: # is a value 0–6 corresponding to the logging program number.

The following logging program corresponds to the following integers:

SURVEY:	0
MANUAL:	1
PROGRAM_1:	2
PROGRAM_2:	3
PROGRAM_3:	4
PROGRAM_4:	5
PROGRAM_5:	6

Return value: a string response OK indicating that the logging program was set, or FAIL indicating an error during the setting of the logging program.

***Note**: This command requires the MUPDATE command to be sent to update the instrument RAM.*

Code Example:

```
// "WMODECURLOG#" is the DustTrak command to write the current
//   instrument logging mode.
// The command must be followed by a CR.

// Command
// buff = "WMODECURLOG0"
//  0   = Log Mode index (0 is the SURVEY mode)
iRet = m_pSocket->Send( "WMODECURLOG0\r", 13);

 // The returned value is a string containing status
//   of the command (i.e. OK or FAIL).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read Instrument Alarm Settings

Command: **RMODEALARMTWO**<<#>>
is a value 0–4 corresponding to the channel of the instrument.
The channel number is only necessary with a DUSTTRAK™ DRX model.

Return value: a comma separated string containing the following data format.

Alarm1 State, Alarm1 Value, STEL Alarm1 State, Alarm2 State, Alarm2 Value

Return value formats:

Alarm1 State:	Integer; see table below
Alarm1 Value:	Float; value of measurement when it is to alarm
STEL Alarm1 State:	Integer; 1 enabled, 0 disabled
Alarm2 State:	Integer; 1 enabled, 0 disabled
Alarm2 Value:	Float; value of measurement when it is to alarm

The following alarm settings correspond to the following integers:

Off	0
Alarm 1 & 2 Audible	1
Alarm 1 & 2 Visible	2
Alarm 1 & 2 Audible, Alarm 1 & 2 Visible	3
Alarm 1 Relay	4
Alarm 1 & 2 Audible, Alarm 1 Relay	5
Alarm 1 & 2 Visible, Alarm 1 Relay	6
Alarm 1 & 2 Audible, Alarm 1 & 2 Visible, Alarm 1 Relay	7

All DRX Models:
The following channels corresponds to the following integer

PM_1	0
$PM_{2.5}$	1
PM_4 (RESP)	2
PM_{10}	3
TOTAL	4

Code Example:

```
// "RMODEALARMTWO#" is the DustTrak command to read the instruments
//    alarm settings.
// The command must be followed by a CR.
// Command
// buff = "RMODEALARMTWO0"
//  0   = Measurement (DRX only)
iRet = m_pSocket->Send( "RMODEALARMTWO0\r", 15);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// Reply
// buff = "3,10.0,0,1,5.0"
//  3    = Alarm1 State
//  10.0 = Alarm1 Value
//  0    = STEL Alarm1 State (Desktop models only use this value)
//  1    = Alarm2 State
//  5.0 = Alarm2 Value
```

Set Instrument Alarm Settings

Command: **WMODEALARMTWO**<<#>> <<**Alarm1 State, Alarm1 Value, STEL Alarm1 state, Alarm2 State, Alarm2 Value**>>

is a value 0-4 corresponding to the channel of the instrument.

The channel number is only necessary with a DUSTTRAK™ DRX model.

Value formats:

Alarm1 State:	Integer; see table below
Alarm1 Value:	Float; value of measurement when it is to alarm
STEL Alarm1 State:	Integer; 1 enabled, 0 disabled
Alarm2 State:	Integer; 1 enabled, 0 disabled
Alarm2 Value:	Float; value of measurement when it is to alarm

The following alarm settings correspond to the following integers:

Off	0
Alarm 1 & 2 Audible	1
Alarm 1 & 2 Visible	2
Alarm 1 & 2 Audible, Alarm 1 & 2 Visible	3
Alarm 1 Relay	4
Alarm 1 & 2 Audible, Alarm 1 Relay	5
Alarm 1 & 2 Visible, Alarm 1 Relay	6
Alarm 1 & 2 Audible, Alarm 1 & 2 Visible, Alarm 1 Relay	7

All DRX Models:
The following channels corresponds to the following integer

PM_1	0
$PM_{2.5}$	1
PM_4 (RESP)	2
PM_{10}	3
TOTAL	4

Return value: a string response OK indicating that the alarm was set, or FAIL indicating an error during the setting of the alarm.

***Note**: On the DRX models the RELAY, AUDIBLE, & STEL are only assessable in one channel. The first channel with the selected item will be the one it is linked to. Alarm2 value must be lower than Alarm1 when both alarms are enabled.*

Also, this command requires the MUPDATE command to be sent to update the instrument RAM.

Code Example:

```
    // "WMODEALARMTWO#" is the DustTrak command to write the instruments alarm
    //   settings.
    // The command must be followed by a CR.
    // Command
    // buff = "WMODEALARMTWO0 3,10.0,0,1,5.0"
    //  0    = Measurement (DRX only)
    //  3    = Alarm1 State
    //  10.0 = Alarm1 Value
    //  0    = STEL Alarm1 State (Desktop models only use this value)
    //  1    = Alarm2 State
    //  5.0 = Alarm2 Value
    iRet = m_pSocket->Send( "WMODEALARMTWO0 3,10.0,0,1,5.0\r", 30);

     // The returned value is a string containing status
    //   of the command (i.e. OK or FAIL).
    iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read Instrument Analog Output Settings

Command: **RMODEANALOG**

Return value: comma separated string containing the following data format.

Desktop Basic Models:
Analog output state, Minimum output range value, Maximum output range value,

Desktop DRX Models:
Analog output state, Analog output measurement, Minimum output range value, Maximum output range value,

The following analog output states correspond to the following values:

OFF	0
Voltage	1
Current	2

The following channels correspond to the following analog output measurement:

PM_1	0
$PM_{2.5}$	1
PM_4 (RESP)	2
PM_{10}	3
TOTAL	4

Code Example:

```
// "RMODEANALOG" is the DustTrak command to read the instruments analog
//    out settings.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMODEANALOG\r", 12);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// DRX Reply
// buff = "2,1,0.000,150.0"
//  2     = Analog Out State
//  1     = Analog Out Measurement
//  0.000 = Lower Limit
//  150.0 = Upper Limit

// DTII Reply
// buff = "2,0.000,150.0"
//  2     = Analog Out State
//  0.000 = Lower Limit
//  150.0 = Upper Limit
```

Set Instrument Analog Output Settings

Command: **WMODEANALOG << Analog output state, Analog output measurement** *(Desktop ONLY)***, Minimum output range value, Maximum output range value** >>

Value formats:

Analog output state:	integer
Analog output measurement:	integer
Minimum output range value:	float
Maximum output range value:	float

The following analog output states correspond to the following values:

OFF	0
Voltage	1
Current	2

The following channels correspond to the following analog output measurement:

PM_1	0
$PM_{2.5}$	1
PM_4 (RESP)	2
PM_{10}	3
TOTAL	4

Return value: a string response OK indicating that the instrument analog output was set, or FAIL indicating an error during the setting of the analog output.

***Note**: This command requires the MUPDATE command to be sent to update the instrument RAM.*

Code Example:

```
    // "WMODEANALOG" is the DustTrak command to write the analog out
settings.
    // The command must be followed by a CR.

    // DRX Command
    // buff = "WMODEANALOG 2,1,0.000,150.0"
    //  2     = Analog Out State
    //  1     = Analog Out Measurement
    //  0.000 = Lower Limit
    //  150.0 = Upper Limit

    // DTII Command
    // buff = "2,0.000,150.0"
    //  2     = Analog Out State
    //  0.000 = Lower Limit
    //  150.0 = Upper Limit
```

```
iRet = m_pSocket->Send( "WMODEANALOG 2,0.000,150.0\r", 26);

 // The returned value is a string containing status
//   of the command (i.e. OK or FAIL).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read the Current Instrument User Calibration

Command: **RMODECURUSERCAL**

Return value: an integer 0 – 9 that corresponds to a user calibration, 10 corresponds to factory calibration.

Code Example:

```
// "RMODECURUSERCAL" is the DustTrak command to read the current
//    instrument user calibration.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RMODECURUSERCAL\r", 16);

//Reply
//buff = "4"
//Instrument currently using user calibration 4
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Set the Current Instrument User Calibration

Command: **WMODECURUSERCAL** <<**#**>>

Where: # is a value 0-9 corresponding to a user calibration number of the instrument, 10 corresponds to factory calibration.

Return value: a string response OK indicating that the user calibration was set, or FAIL indicating an error during the setting of the user calibration.

***Note**: This command requires the MUPDATE command to be sent to update the instrument RAM.*

Code Example:

```
// "WMODECURUSERCAL#" is the DustTrak command to write the current
//    instrument user calibration.
// The command must be followed by a CR.
// Command
// buff = "WMODECURUSERCAL0"
//  0   = Current User Cal

iRet = m_pSocket->Send( "WMODECURUSERCAL0\r", 17);

 // The returned value is a string containing status
//   of the command (i.e. OK or FAIL).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read Instrument User Calibration Data

Command: **RMODEUSERCAL**<<#>>
is a value 0-9 corresponding to a user calibration number of the instrument.

Return value: comma separated string containing the following data format.
User calibration number, Size correction factor, Photometric calibration factor, User calibration name,

Code Example:

```
// "RMODEUSERCAL#" is the DustTrak command to read the instruments user
//    calibration settings.
// The command must be followed by a CR.
// Command
// buff = "RMODEUSERCAL0"
//  0   = User Cal index
iRet = m_pSocket->Send( "RMODEUSERCAL0\r", 14);

iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// Reply
// buff = "0,1.5,0.9,USER CAL 0"
//  0   = User Cal index
//  1.5 = Size Correction Factor (DRX only)
//  0.9 = Photometric Calibration Factor
//  USER CAL 0 = User Cal Name
```

Set Instrument User Calibration Data

Command: **WMODEUSERCAL** <<**#**>><< **User calibration number, Size correction factor, Photometric calibration factor, User calibration name** >>
is a value 0-9 corresponding to a user calibration number of the instrument.

Value formats:

User calibration number:	integer 0-9
Size correction factor:	float
Photometric calibration factor:	float
User calibration name:	char [12]

Return value: a string response OK indicating that the instrument user calibration was written to the instrument, or FAIL indicating an error during the setting of the user calibration.

Note: *This command requires the MUPDATE command to be sent to update the instrument RAM. The Size Correction factor is only used in the DRX models.*

Code Example:

```
    // "WMODEUSERCAL#" is the DustTrak command to write the instruments user
cal
    //   settings.
    // The command must be followed by a CR.
    // Command
    // buff = "WMODEUSERCAL0 1.5,0.9,USER CAL 0"
    //  0   = User Cal index
    //  1.5 = Size Correction Factor (DRX only)
    //  0.9 = Photometric Calibration Factor
    //  USER CAL 0 = User Cal Name
    iRet = m_pSocket->Send( "WMODEUSERCAL0 1.5,0.9,USER CAL 0\r", 33);

     // The returned value is a string containing status
    //   of the command (i.e. OK or FAIL).
    iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read the Instrument Date And Time

Command: **RSDATETIME**

Return value: a date time string of the format Day/Month/Year Hour:Minute:Second.

Code Example:

```
// "RSDATETIME" is the DustTrak command to read the instruments date
//    and time.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RSDATETIME\r", 11);

iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// Reply
// buff = "9/30/2008,13:44:5"
//  09   = Month
//  30   = Day
//  2008 = Year
//  13   = Hour
//  44   = Minute
//  05   = Second
```

Set the Current Instrument User Calibration

Command: **WSDATETIME**<<**Date Time**>>

Where: Date Time is a date time string of the format Day/Month/Year Hour:Minute:Second.

Return value: a string response OK indicating that the date and time was set, or FAIL indicating an error during the setting of the date and time of the instrument.

***Note**: This command requires the MUPDATE command to be sent to update the instrument RAM.*

Code Example:

```
// "WSDATETIME" is the DustTrak command to write the instruments date
//   and time.
// The command must be followed by a CR.
// Leading zeros are optional.
// Command
// buff = "WSDATETIME 09/30/2008,13:44:05"
//  09   = Month
//  30   = Day
//  2008 = Year
//  13   = Hour
//  44   = Minute
//  05   = Second
iRet = m_pSocket->Send( "WSDATETIME 09/30/2008,13:44:05\r", 31);

 // The returned value is a string containing status
//   of the command (i.e. OK or FAIL).
iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
```

Read the Filter Change Date

Command: **RSFILTERCHANGEDATE**

Return value: comma separated string containing the following data format.

Day/Month/Year, Concentration since last filter change

Code Example:

```
// "RSFILTERCHANGEDATE" is the DustTrak command to read the instruments
//    filter change date.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RSFILTERCHANGEDATE\r", 19);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
// Reply
// buff = "8/27/2008,456"
//  09   = Month
//  30   = Day
//  2008 = Year
//  456  = Concentration since last filter change
```

Read the Calibration Date

Command: **RSCALDATE**

Return value: comma separated string containing the following data format.

Day/Month/Year, Run time since last calibration in hours, Concentration since last calibration in mg, Pump run time in hours

Code Example:

```
// "RSCALDATE" is the DustTrak command to receive the instruments
// calibration  date.
// The command must be followed by a CR.
iRet = m_pSocket->Send( "RSCALDATE\r", 10);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
 // Reply
 // buff = "12/15/2008,1,2.2017,163"
 //  12   = Month
 //  15   = Day
 //  2008 = Year
 //  1  = Runtime since last cal (hours)
 //  2.2017 = Concentration since last cal (mg)
 //  163 = Total pump runtime (hours)
```

Update Instrument Zero

Command: **MZERO**

Return value: a string response OK indicating that the instrument zero function was initiated, or FAIL indicating the zero function did not start.

Warning: Only send this command if the unit has an autozero accessory attached. If this command is send without an autozero accessory, then it will perform a zero in non-filtered air.

Code Example:

```
// "MZERO" is the DustTrak command to set the instrument into the zero state.
 // The command must be followed by a CR.

iRet = m_pSocket->Send( "MZERO\r", 6);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);

 // Reply
 // buff = a string containing "OK" or "FAIL"
```

Read the Instrument Zeroing state

Command: **RMZEROING**

Return value: comma separated string containing the following data format.

Current Seconds, Total Seconds. Current seconds is the current amount of time instrument has been performing a zero. Total seconds is the total amount of time it will take to complete the zero operation. Use this command after sending the MZERO command to the instrument to track the zeroing progress.

Code Example:

```
// "RMZEROING" is the DustTrak command to read the status of a zeroing
operation.
 // The command must be followed by a CR.

  iRet = m_pSocket->Send( "RMZEROING\r", 10);

 iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
 // Reply
 // buff = "30,150"
 //  30  = Current second
 //  150   = Total seconds
```

Read the Instrument Memory State

Command: **RMMEMORY**

Return value: comma separated string containing the following data format.

Available Tests, Available Samples, Available STEL

Code Example:

```
    // "RMMEMORY" is the DustTrak command to read the status of the
instruments
    //  memory.
    // The command must be followed by a CR.

     iRet = m_pSocket->Send( "RMMEMORY\r", 9);

    iRet = m_pSocket->Receive( buff, PACKET_SIZE, 0);
    // Reply
    // buff = "Available Tests:   999 of 999
    //            Available Samples: 65000 of 65000
    //            Available STEL:    99 of 99"
    //  999 of 999   = number of tests logged out of max number of tests (999)
    //  65000 of 65000   = number of samples logged out of max number of samples
    //                          (65000)
    //  99 of 99   = number of STEL tests logged out of max number of STEL tests
(99)
```

TSI Incorporated – 500 Cardigan Road, Shoreview, MN 55126 U.S.A

USA	Tel: +1 800 874 2811	E-mail: answers@tsi.com	Website: www.tsi.com
UK	Tel: +44 149 4 459200	E-mail: tsiuk@tsi.com	Website: www.tsiinc.co.uk
France	Tel: +33 491 11 87 64	E-mail: tsifrance@tsi.com	Website: www.tsiinc.fr
Germany	Tel: +49 241 523030	E-mail: tsigmbh@tsi.com	Website: www.tsiinc.de
India	Tel: +91 80 41132470	E-mail: tsi-india@tsi.com	
China	Tel: +86 10 8251 6588	E-mail: tsibeijing@tsi.com	
Singapore	Tel: +65 6595 6388	E-mail: tsi-singapore@tsi.com	

Contact your local TSI Distributor or visit our website **www.tsi.com** for more detailed specifications.

P/N 6002481 Rev. G Printed in U.S.A