Pike **A**erospace **R**esearch **C**orporation
42 Silver Aspen Crescent
Kitchener, Ontario
Canada, N2N-1J1

Phone: (705) 586-2255

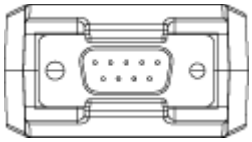
email: sales@pikeaero.com
web: http://www.pikeaero.com

Model PA1200

Serial Temperature Probe Sensor

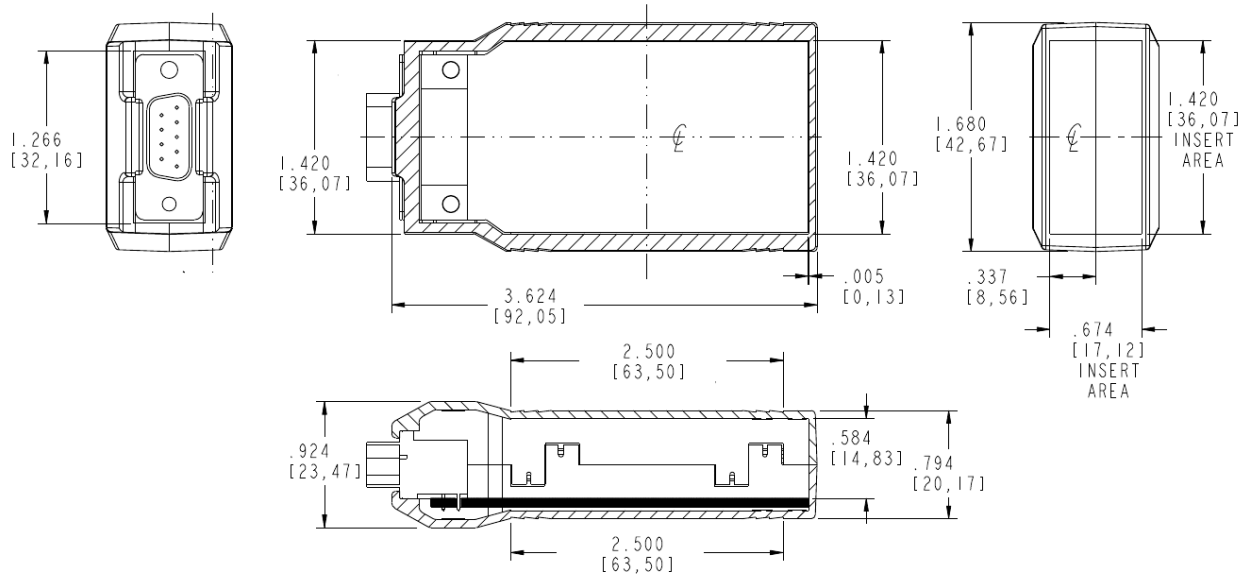
User Manual

Firmware Revision: **3.1**



PA1200 Specifications

Temperature Range: -55 º C to +125 º C (-67 º F to +257 º F)
Dimensions: 9.1 cm (3.6") x 4.2 cm (1.7") x 2 cm (0.8")
Housing Material: injection molded ABS plastic
Housing Colour: Black (Bone White Optional)
Power source: Data Cable Powered (DTR/RTS) or 5VDC to 24VDC supply.
Current Consumption: ~10 mA
Communications Interface: RS232.
Maximum RS232 extension cable length: 152.5 m (950 ft.) @ 2400 baud.
Communications Protocol: 8-bit ASCII with 16-bit CRC or Checksum error detection.
Communication Baud Rates: 1200, (2400), 4800, 9600, 19200, 38400, 57600, 115200



PA1200 DB9F RS232 Pinout

PIN	SIGNAL	IN/OUT *	DESCRIPTION
1	DCD	N/C	Data Carrier Detect
2	RxD	OUT	Receive Data
3	TxD	IN	Transmit Data
4	DTR **	IN	Data Terminal Ready
5	GND	-	Signal Ground
6	DSR ***	OUT	Data Set Ready
7	RTS **	IN	Request To Send
8	CTS ***	OUT	Clear To Send
9	RI	N/C	Ring Indicator

- * IN/OUT Relative to the PA1200 device.
- ** DTR and RTS are required as a power source and must be asserted for data-line powered operation.

POWER

For data-lline powered operation, both DTR (Data Terminal Ready) and RTS (Request To Send) shall be asserted by the host at all times while communications are taking place.

The PA1200 requires approximately 1 millisecond from the time that DTR+RTS are asserted, until the time it is ready for receiving commands.

Additionally, and external positive DC power supply in the range of +5V to +24V may be connected to the DB9F pins 4 and 7 (V+), and DP9F pin 5 (GND).

COMMUNICATION

The communication between the host and the PA1200 is a simple master/slave style of protocol. The host (computer or device server) register queries, and the PA1200 responds. The fields in each query and response packet are separated by a ':' (colon) character, and the packet is terminated with a carriage-return (Hex 0D) character, or a carriage-return/line-feed pair (Hex 0D/0A).

REGISTER	READ/ WRITE	VARIABLE	DESCRIPTION
R0	R	VARS	Number of Variables.
R1	R	MODEL	Product Model NUmber.
R2	R	SERIAL	Unit Serial Number.
R3	R	VENDOR	Vendor Identification Data.
R4	R	REV	Firmware Revision.
R5	R	TEMPC	Celcius Reading.
R6	R	TEMPF	Fahrenheit Reading.
R7	R	STATUS	Device Status Flag
R8	W	OPTION	Option Settings

COMMAND PACKET FORMAT

rr<CR>
wr:xx<CR>

rr	Read a register. Example: **R2**.
wr	Write a register. Example: **W12:0x80**
xx	Decimal,Hexadecimal or Text. Hex value must be preceeded with "0x".
<CR>	The carriage-return character. (0D Hex). <CR><LF> is also acceptable.

RESPONSE PACKET FORMAT

rr:t:a:xxxx:s:yyyy:zzzz<CR><LF>

rr	The register name.
t	The data type (I=integer,R=real,S=string,B=boolean).
a	Access Mode (R=read,W=read/write).
xxxx	Contains the value for the register.

s	Unit of measure.
yyyy	Register name.
zzzz	16-Bit hexadecimal 2's compliment checksum.

RESPONSE PACKET EXAMPLES:

R0:I:R:9:*:VARS:FBE7

R1:S:R:PA1200:*:MODEL:FA8B

R2:S:W:12345678:*:SN:FB06

R3:S:R:www.pikeaero.com:*:VENDOR:F531

R4:S:R:3.1:*:REV:FBCF

R5:R:R:20.7:C:TEMPC:FAF5

R6:R:R:69.2:F:TEMPF:FAE6

R7:I:R:1:*:STATUS:FB40

R8:I:W:0x90:*:OPTION:FA65

VENDOR (VENDOR)

The vendor name may be modified by OEMs to reflect their own branding. The field is limited to 30 characters maximum, and special characters such as ':' must be avoided.

SERIAL NUMBER (SN)

The serial number is factory programmed with a factory unique serial number. This field may be modified by OEMs to reflect their own device tracking or inventory scheme.

RELATIVE HUMIDITY CALIBRATION (RHCAL)

This is a factory calibration value, and should rarely require maintenance. After years of service, the capacitive humidity sensing cell used on the PA1200 sensor may drift out of calibration slightly. This tendency is very slight. At the very most, amounting to about +/- 0.5% per year under the worst conditions. Adjusting RHCAL can be used to reset the RH calibration should it fall too far out of acceptable tolerence.

STATUS

This value indicates that the readings are valid of equal to 1, a value of 0 indicates a faulty reading.

OPTION BYTE

The option byte register contains a number of bits which can be used to modify the behaviour of the sensor.

```
OPTION[0] : (0x01) Error Detection Select.
      1 = Use CRC.
      0 = Use Checksum.

OPTION[3:1] : (0x0E) Unused.

OPTION[6:4] : (0x70) Baud Rate Select.
      000 = 1200
      001 = 2400
      010 = 4800
      011 = 9600
      100 = 19200
      101 = 38400
      110 = 57600
      111 = 115200

*OPTION[7] : (0x80) Write Protection Select.
      1 = Write Protected.
      0 = Write Enabled.



* In order to disable write-protection ( OPTION[7] == 1 ), the write-protection
bit (OPTION[7]) should be first written as '1', and then '0' in succession. When
disabling write protection as described, all of the other bits in the option
word remain unaffected.
```

CRC CALCULATION

The 16-bit CRC calculation is the hexadecimal representation of the CRC of all of the data up to and including the last field separator (:) character.

```
/**
 * Example PA110x sensor response packet validation using CRC.
 */
#include <stdio.h>
#include <string.h>

#define SEPCHAR         ':'     /** The field separation character */
#define MAX_SEPCHARS    6       /** Number of significant SEPCHARs */
#define P_16            0xA001  /** 16 bit polynomial for CRC gen */

unsigned short crc_tab16( unsigned char ch )
{
    unsigned short crc, c;
    unsigned char i=0;
    unsigned char j;
    do {
        crc = 0;
        c   = (unsigned short) i;
        for (j=0; j<8; j++)
        {
            if ( (crc ^ c) & 0x0001 )
                crc = ( crc >> 1 ) ^ P_16;
            else
                crc >>= 1;
            c >>= 1;
        }
    } while ( i++ != ch );
    return crc;
}

void update_crc16( unsigned short* crc, char c )
{
    unsigned short tmp, short_c;
    short_c = 0x00ff & (unsigned short) c;
    tmp = (*crc) ^ short_c;
    *crc = ((*crc) >> 8) ^ crc_tab16( tmp & 0xff );
}

unsigned short xtoi(const char* str)
{
    unsigned short rc=0;
    while ( *str && isxdigit(*str ) )
    {
        char ch = toupper(*str++);
        char nibble=0;
        if ( ch >= 'A' && ch <= 'F' )
            nibble = (ch-'A')+10;
        else
            nibble = (ch-'0');
        rc <<= 4;
        rc |= nibble;
    }
    return rc;
}
```

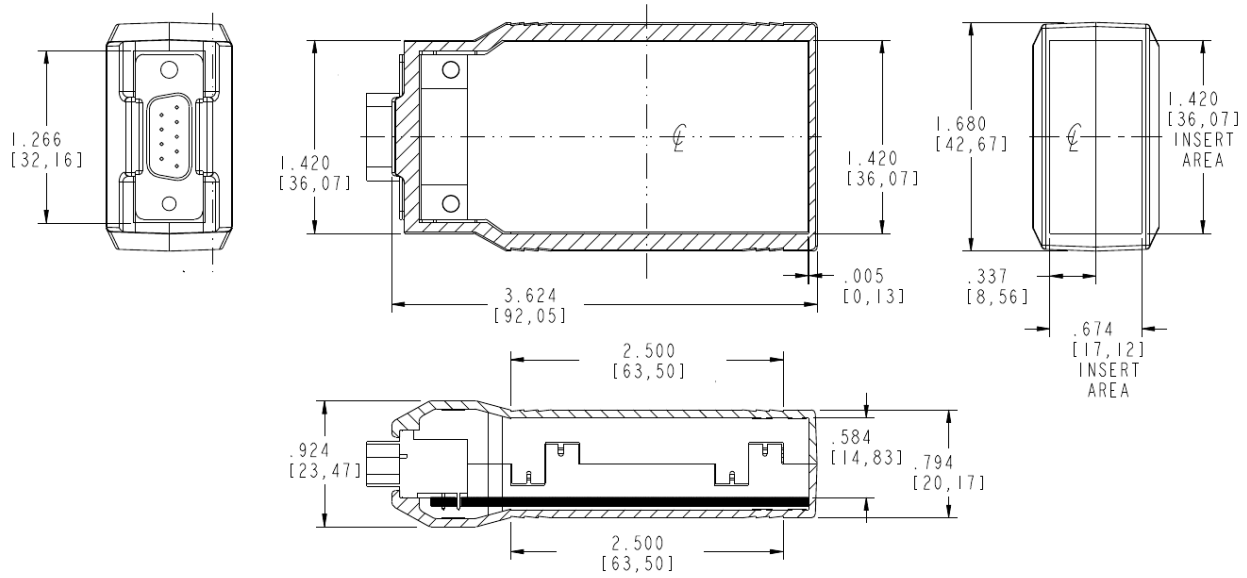
```
int validate(const char* pa110x_response)
{
    unsigned short crc=0;
    int nsepchars=0;
    int n;
    for(n=0; nsepchars < MAX_SEPCHARS && n < strlen(pa110x_response); n++)
    {
        update_crc16(&crc,pa110x_response[n]);
        if ( pa110x_response[n] == SEPCHAR )
            ++nsepchars;
    }
    return crc == xtoi(&pa110x_response[n]);
}

int main(int argc,char* argv[])
{
    if ( argc == 2 )
    {
        printf( "%s\n", validate(argv[1]) ? "valid" : "invalid" );
    }
}
```

CHECKSUM VALIDATION

The checksum is the hexadecimal representation of the the 2's compliment of the sum of all of the data up to and including the last field separator (:).

```
/**
 * Example PA110x sensor response packet validation using checksum.
 */
#include <stdio.h>
#include <string.h>

#define SEPCHAR         ':'
#define MAX_SEPCHARS    6

unsigned short xtoi(const char* str)
{
    unsigned short rc=0;
    while ( *str && isxdigit(*str ) )
    {
        char ch = toupper(*str++);
        char nibble=0;
        if ( ch >= 'A' && ch <= 'F' )
            nibble = (ch-'A')+10;
        else
            nibble = (ch-'0');
        rc <<= 4;
        rc |= nibble;
    }
    return rc;
}

int validate(const char* pa110x_response)
{
    unsigned short checksum=0;
    int nsepchars=0;
    int n;
    for(n=0; nsepchars < MAX_SEPCHARS && n < strlen(pa110x_response); n++)
    {
        checksum += pa110x_response[n];
        if ( pa110x_response[n] == SEPCHAR )
            ++nsepchars;
    }
    checksum = ~checksum;
    return checksum == xtoi(&pa110x_response[n]);
}

int main(int argc,char* argv[])
{
    if ( argc == 2 )
    {
        printf( "%s\n", validate(argv[1]) ? "valid" : "invalid" );
    }
}
```

DIMENSIONS


1.266
[32,16]
1.420
[36,07]
1.420
[36,07]
3.624
[92,05]
.005
[0,13]
1.680
[42,67]
1.420
[36,07]
INSERT
AREA
.337
[8,56]
.674
[17,12]
INSERT
AREA
2.500
[63,50]
.924
[23,47]
.584
[14,83]
.794
[20,17]
2.500
[63,50]