FORMAT OF THERMALPRO BINARY FILES

TECHNICAL NOTE HWA-001

Thermal anemometry users sometimes want to apply their own corrections to the measured velocity or voltage data, supplied by the IFA300 processor. One way to do this is to export the Bridge Voltage as a text file. This can be done from the Analysis screen. In the “Analyze/Generate” box, check the “Bridge” and “Text” boxes then select the Rxxxx files you wish to export. Click the “Analyze” button to begin. An ASCII text file is created with time in column 1, and then the data in succeeding columns. Channel 1 is in column 2, Channel 2 in column 3, etc.

In some cases this is not practical, due to large number of files and the resulting size of text files. Thus you may want to work with the binary data files instead. The binary formats for Raw data files (*.R*) and Velocity files (*.V*) are the same for the 16 bit version of ThermalPro (up to version 2.xx) as for the 32 bit versions (4.00 and later)

Raw Data Files (*.Rxxxx)

R files use Metrabyte format. Each sample is 16 bits wide. The upper 12 bits are the actual sample data (0 to 4095), and the lower 4 bits designate the channel (0 to15). Data appear in sequential order as a continuous loop (ch 1, ch 2 . . . ch n, ch 1, ch 2 . . . ch n, ch 1, ch 2, etc).

Example:

4 3 2 1
9 9 C 0
1 9 B 1
6 1 9 2
9 A O 0
1 9 1 1
6 1 3 2
9 9 B 0
.
.
.

The corresponding decimal data is:

Channel Number	Value (Scan1)	Value (Scan2)	Value (Scan3)
1	2460	2464	2459
2	411	401	etc.
3	1561	1555	etc.

Velocity Files (*.Vxxxx)

Output Voltage Files (*.Axxxx)

Bridge Voltage Files (*.Exxxx)

Files appear in the same sequential order as Raw Data Files, except for files with multi-channel probes where U, V, W appear in place of ch 1, ch 2, ch 3, as an example.

In C language the union data structure allows multiple data types to occupy the same memory space. The velocity files use an 8 byte union which contains a floating point velocity value and an integer channel number.

C language syntax:

```
union  _utmp
                                {
                                   double  dval;     /* one 8 byte wide double precision */
                                   float  fval[2];   /* two 4 byte wide floating point */
                                   int  ival[4];     /* four 2 byte wide integers */
} utmp;
```

Example:

7	6	5	4	3	2	1	0	Byte Count
C	C	U	U	F	F	F	F	F = Floating Point Data C = Channel (0 to 15) U = Unused space filled with zeros

The following fragment shows how to use the union data structure:

```
int channel;
float velocity;

/* read from the file array of doubles, index to the sample you need, then use the union to extract the
particular component */

velocity = utmp.fval[0];
channel = utmp.ival[3];

....user code here
```