INSTITUTE OF OCEANOGRAPHIC SCIENCES
DEACON LABORATORY

INTERNAL DOCUMENT No. 342

SWALES Sonic Buoy - meteorological data report

C H Clayson

1994

Wormley
Godalming
Surrey GU8 5UB UK
Tel +44-(0)428 684141
Telex 858833 OCEANS G
Telefax +44-(0)428 683066

DOCUMENT DATA SHEET

AUTHOR CLAYSON, C H	*PUBLICATION DATE* 1994

TITLE

SWALES Sonic Buoy - meteorological data report.

REFERENCE

Institute of Oceanographic Sciences Deacon Laboratory, Internal Document, No. 342, 42pp.
(Unpublished manuscript)

ABSTRACT

During the SWALES experiment in the autumn of 1993, the Sonic Buoy was deployed twice as part of an array of moored instrumentation. On the Sonic Buoy, meteorological data were acquired by the Formatter Processor from the Sonic and Multimet Processors' output data streams. These data were combined and logged as 10 minute means on the Formatter Flashcard memory; a selection of the data was also telemetered in near real time via the polar orbiting ARGOS and geostationary METEOSAT satellite data collection systems.

This data report briefly describes the processes employed in acquisition of the data. It then describes the processes for the recovery of the data from the various source media, the quality control procedures applied and, finally, the resulting output data files.

Appendices include comprehensive details of the software developed for the above processes and of the formats used for the input and output data.

KEYWORDS

ISSUING ORGANISATION

Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB. UK.

Director: Colin Summerhayes DSc

Telephone Wormley (0428) 684141
Telex 858833 OCEANS G.
Facsimile (0428) 683066

Copies of this report are available from: ***The Library,*** *PRICE* £0.00

Index

SWALES SONIC BUOY - METEOROLOGICAL DATA REPORT 7

Equipment 7

Data Sources and Processing 7
ARGOS Data 8
Meteosat Data 9
PCMCIA Flash Card Data 9

Data Quality Checking 11
Figures 2, 3 13
Figures 4,5 14
Figures 6,7 15
Figures 8,9 16
Figures 10,11 17
Figures 12,13 18
Figure 14 19
Figures 15,16 20
Figures 17,18 21

Summary of Data Produced 22
Raw Data Files 22
CricketGraph Data 22

Data Time Stamping 22

Acknowledgements 23

References 23

APPENDIX A SOFTWARE LISTINGS 24

A.1 ARGSONFILE 24

A.2 SORT RECS 28

A.3 METEO SORT 30

A.4 4MTO1M.C 33

A.5 FORMDECODE 35

A.6 SONDIRN 37

APPENDIX B DATA FORMATS 39

Appendix B.1 Raw Data Files 39

Appendix B.2 ARGSONFILE and SORT RECS Output Files 43

Appendix B.3 CricketGraph Data 43

SWALES Sonic Buoy - Meteorological Data Report

Equipment

The Formatter operation is fully described in ref.1. The Formatter consists of a single board PC-compatible processing system, with 4 additional serial ports, a 4 Mbyte PCMCIA series 1 Flash EEPROM card and software embedded in EPROM.

Briefly, the Formatter asynchronously accepts quarter-hourly processed data messages from the Sonic Processor and one-minute-mean data messages from the Multimet Processor. Upon receipt of a Sonic message, or on the quarter-hour if no Sonic message is received, the Formatter averages the Multimet data lying within a 10 minute window corresponding to the Sonic data acquisition period (having corrected the received message time stamps for clock drift relative to its own real time clock).

The Formatter then converts Sonic data to a concise binary format, with parity checks, for transmission via the ARGOS polar-orbiting satellite system; the ARGOS data is sent cyclically as 4 x 32 byte messages, comprising 5 hours of Sonic data.

The Formatter also converts Sonic and averaged Multimet data, converted to engineering units, to a 288 byte ASCII format message for transmission via the Meteosat geostationary satellite at hourly intervals.

The 128 bytes of ARGOS data and the 288 bytes of Meteosat data were written to a Flash EEPROM data card at quarter-hourly intervals as a back up. The Formatter data were originally envisaged mainly as a real time source of quick-look data, with the secondary function of providing back up of an abbreviated data set by telemetry and a separate storage medium. Due to the failure of the Multimet EPROM logger on both deployments, this back up became invaluable as a source of Met data, but this necessitated the expenditure of considerable additional effort to produce the required form of data products.

Data Sources and Processing

For an overall view of the data sources and processing, see Figure 1. The Sonic Buoy was deployed for two separate periods:

Day 293.59	(1st deployed)	to Day 315	(recovered inverted)
Day 326.60	(re-deployed)	to Day 355	(recovered from rocks)

During the 1st deployment, the buoy overturned at day 313.58. During the 2nd deployment, the buoy systems progressively failed due to battery exhaustion from approximately day 338.53.

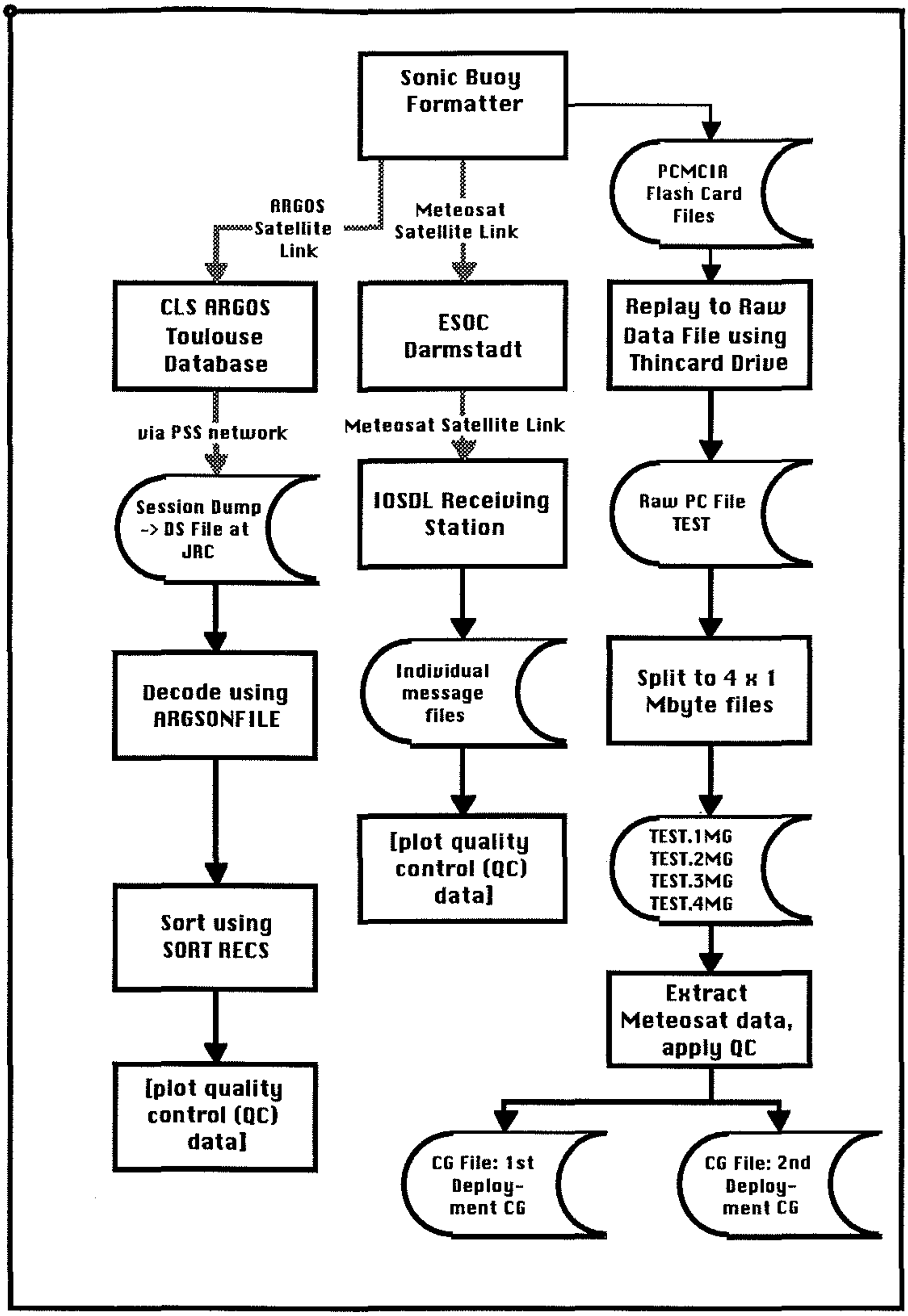

Figure 1 Data Sources and processing

ARGOS Data

During the deployments, data were received from the Sonic Buoy via the ARGOS system, which was regularly interrogated during the experiment to allow checks on both buoy position and data quality. The ARGOS messages, downloaded from the CLS ARGOS computer at Toulouse via the Public Switched System were decoded and sorted by the QuickBasic applications ARGSONFILE and SORT RECS (Appendices A.1, A.2). The former decoded all Sonic Buoy ARGOS messages within an ARGOS dump into wind data; the latter sorted the data into chronological order, selecting the best choice from duplicated data, and produced chronologically ordered tabular files of the parameters PSD, MWS, Fit-A, Vertical MWS and N2. File formats are given in Appendix B.

The ARGOS data naturally terminated with the capsize of the buoy in the first deployment and with exhaustion of the batteries in the second deployment.

Meteosat Data

Due to a battery charger failure, Meteosat telemetry was not possible during the first deployment. During the second deployment, the buoy messages were transmitted via a transponder on Meteosat to the European Space Operations Centre at Darmstadt. They were then retransmitted via Meteosat (interleaved with the WEFAX transmissions) and received by a local Meteosat DCS message recovery unit (MRU) at IOSDL, Wormley. The MRU decoded the DCS transmissions and filtered out the Sonic Buoy messages; these were then passed via RS232 to a Macintosh Classic running the compiled application LOGSW1 APL

The Macintosh further decoded the messages and stored them in daily numbered folders, which were transferred at intervals to Macintosh IIvx for examination and further processing. A QuickBasic program METEO SORT (Appendix A.3) was written to combine the individual messages into day files and to produce a report file, flagging errors, since the messages were not error free. Manual editing was used to correct the message files for the flagged errors (or to substitute 999 default data, if appropriate); the program METEO SORT was then re-run to produce an error free tabular file suitable for importation into CricketGraph.

PCMCIA Flash Card Data

The complete data set was recovered from the 4 Mbyte Flash Memory Card after each deployment. The card contents were dumped to a PC disk file, using a Databook ThinCard drive and associated software. The resulting 4 Mbyte file was then split into 4 x 1 Mbyte files by the C program 4MTO1M.C (Appendix A.4) to allow easier handling and transfer to other machines; the resulting PC files were named FORMSWAL.1MG, FORMSWAL.2MG, FORMSWAL.3MG, FORMSWAL.4MG

The directory information in the first 256 kbytes of the card memory gave the Formatter date/time and memory location for the start of each 128 + 288 byte "record". Since the 128 byte ARGOS data was duplicated (with the exception of the N2 values) over a number of the 288 byte Meteosat data sets in the records, attention was focussed primarily on processing the Meteosat data. However, the QuickBasic program FORMDECODE (Appendix A.5) was written to decode the Flash file ARGOS data into a similar format to that produced by ARGSONFILE, but incorporating the Meteosat ASCII data. This gave additional useful diagnostic information when it was necessary to correct for timing errors of the Sonic Processor during the second deployment.

The required data products were tabular ASCII files of all the quarter-hourly data, for the two deployments. The Meteosat header data includes a Julian day number, JJJ. It also contains the most recent four quarter-hourly sets of the Sonic and Multimet data

i.e. QQ,+PSD,MWS,+NWS,+EWS,+VWS,+F_A,+AT1,+AT2,+ST1,+ST2,YWS,YDR<CR>

plus a housekeeping line of data containing:

BAT,HDG,HSD,+TMET,+TSON<CR>

where QQ = Quarter-hours since midnight (range 00 - 96)

+PSD = 100 * log10(Power Spectral Density * f^5/3

MWS = 10 * (Mean Wind Speed in m/s)

NWS = 10 * (North Mean component of Wind Speed in m/s)

EWS = 10 * (East Mean component of Wind Speed in m/s)

VWS = 10 * (Vertical Mean component of Wind Speed in m/s)

F_A = 100 * Coefficient 'a', for linear regression fit of PSD vs log10(frequency)

(over the frequency range 2 - 4 Hz, PSD = a + b.log10(frequency))

AT1 = 10 * (Mean Air Temperature from sensor 1 in ^{o}C)

AT2 = 10 * (Mean Air Temperature from sensor 2 in ^{o}C)

ST1 = 10 * (Mean Sea Temperature from sensor 1 in ^{o}C)

ST1 = 10 * (Mean Sea Temperature from sensor 2 in ^{o}C)

YWS = 10 * (Mean Young AQ1 Wind Speed in m/s)

YDR = Mean Young AQ1 Wind Direction in degrees.

BAT = 10 * Mean Battery Voltageon the 24V bus

HDG = Mean Buoy Heading in degrees magnetic

HSD = Standard Deviation of Heading in degrees

+TMET = Time difference between Multimet and Formatter Real Time Clocks

(+ve for Multimet clock ahead of Formatter clock)

+TSON = Time difference between Sonic and Formatter Real Time Clocks

(+ve for Sonic clock ahead of Formatter clock)

<CR> = Carriage Return

The Julian day number and Quarter normally originate from the Sonic Processor but, in the absence of a Sonic message, originates from the Formatter clock. Latch up of the COM3 port interrupt occasionally resulted in loss of Sonic messages, resulting in a temporary reversion to Formatter date/time; fortuitously, this assisted in correction for timing errors of the Sonic Processor during the second deployment.

The production of a tabular data set for the first buoy deployment was relatively straightforward. The ARGOS (binary) data were stripped out of the Flash files, together with the redundant part of the header. For each remaining Meteosat "record" the housekeeping data were then appended to the most recent set of the four quarter-hourly sets of Sonic and Multimet data. The other three sets were stripped out, leaving two lines per record of the format:

JJJ<CR>

QQ,+PSD,MWS,+NWS,+EWS,+VWS,+F_A,+AT1,+AT2,+ST1,+ST2,YWS,YDR,BAT,HD G,HSD,+TMET,+TSON<CR>

A simple program then converted these data to a tabular file with lines of the format:

JJJ.JJJJ<tab>+P.SD<tab>MW.S<tab>+NW.S<tab>+EWS<tab>+VW.S<tab>+F._A<tab> +AT.1<tab>+AT.2<tab>+ST.1<tab>+ST.2<tab>YW.S<tab>YDR<tab>BA.T<tab>HDG <tab>HSD<tab>+TMET<tab>+TSON<CR>

as described in Appendix B.3

The production of a tabular data set for the second buoy deployment was made more difficult by jumps in the Sonic Processor clock; these occurred at a Sonic Processor clock time of just before midnight due to incorrect functioning of the RTCN.EXE application in the Sonic software. This application was intended to reset the system clock (computed time) from the Real Time Clock just before midnight each day. However, on a number of occasions, the operation of the application caused an incorrect time to be set in, resulting in a time slip as indicated by +TSON.

The time slip +TSON (in minutes) was used in conjunction with the (Formatter clock) quarter-hours from the ARGOS data to correct the Sonic date and quarter-hours in the Meteosat data. Otherwise, the method used to extract the data into tabular form was similar to that used for the first deployment data.

Due to latch up of the Sonic UART interrupt on a few occasions, some Sonic data were missing from the FlashCard data; after the Sonic Processor EPROM logger data had been processed to tabular parameter files by SONPARAMS.BAS (see ref. 2), it was possible to paste the missing data from these files into the CricketGraph data. At the same time, it was possible to correct some minor timing errors. This resulted in the files 1st Deplyment CG final and 2nd Deployment CG final.

Data Quality Checking

The tabular data sets were separately imported into CricketGraph; they were then edited to remove duplicated sonic data arising from the COM3 latch-up problem mentioned above. Values of F_A of -9.99 were left unaltered; this value occurs when the least squares fit of PSD against frequency gives a negative intercept.

The value of Young Direction is normally zero in the first deployment, due to an incorrect channel allocation to the Young 2 wind direction channel in the Multimet message (Young 2 was connected to the Buoy Motion Package and <u>not</u> to Multimet). However, it is interesting to note that Young Direction shows non-zero values after the buoy overturned. The channel allocation fault was corrected for the second deployment.

In order to give a wind direction for the first deployment, the (relative wind direction + 180°), referred to as Sonic Heading, was calculated from the Sonic +NWS and +EWS values, using the QuickBasic application SONDIRN (Appendix A.6). The 180° was added to make the direction comparable to the Young Direction for the second deployment, the Young sensors being aligned at 180° to the Sonic North, to prevent the 360°/0° discontinuity problem. Quadrant direction averaging was not incorporated in the Multimet Wind Direction channels, although it was used for Buoy Heading. Examination of the calculated relative wind direction, figure 2, showed the buoy North to be heading into the wind for the majority of the deployment; exceptions were during spells of low wind and were probably due to combinations of wind and tidal aligning moments. It may be considered desirable to omit Sonic data for such instances. In contrast, during the second deployment (figure 3), the buoy did not maintain such good alignment with the wind; this was probably due in part to the stronger tidal currents, but may also have been due to the modified mooring. Again, it may be considered desirable to omit Sonic data when the relative wind direction was more than about $\pm 90°$.

The wind direction relative to magnetic North, "Wind Direction (to)", was calculated from Sonic Heading + Buoy (magnetic) Heading; figures 4 and 5 show Sonic MWS and Wind Direction (to) for the two deployments. Examination of the data shows quite large cyclical variations in Wind Direction (to), especially during the second deployment; these are correlated to periods of high values of Heading Standard Deviation (figures 17, 18). Due to a misinterpretation, quadrant heading averaging in the Formatter was carried out on the basis of the Multimet data being in degrees (0 - 359) and not in digital units (0 - 255). This results in incorrect averaging when the 360°/0° discontinuity occurs. However, it would appear from inspection of the data that this is flagged by very large values of Heading Standard Deviation (100 degrees or more). The above-mentioned cases of cyclical variations in Wind Direction (to) occurred with maximum Heading Standard Deviation values of about 10 degrees and with Buoy Heading nowhere near the 360°/0° discontinuity. Examination of the data shows that the Buoy Heading swung through a greater angle than Sonic Direction. This could be due to magnetic materials within the buoy canister/hull. Sonic Direction is generally close to Young Heading and is considered to be correct. To resolve this anomaly, it would be necessary to do a compass calibration with the buoy in its full working configuration.

Air and Sea Temperatures were plotted; the plot for the first deployment (figure 6) clearly shows the capsize, after which the air temperatures were underwater and may give a good measure of sea temperature at a depth of about 1.5 metres. The sea temperature sensors could not be expected to measure air temperature correctly after the capsize. The plot for the second deployment (figure 7) shows an interesting transient oscillatory sea temperature change starting at day 329.5

Plots of the differences between the sensor pairs (figures 8 - 11) show that AT1 was reading on average between 0.2 and 0.3 °C higher than AT2 whilst ST1 was reading on average approximately 0.2 °C lower than ST2 during the first deployment and 0.1 °C higher than ST2 during the second deployment. Any possiblecorrelation between the temperature differences and meteorological conditions has yet to be demonstrated.

Scatter plots (figures 12 and 13) of Young wind speed (YWS) against Sonic wind speed (MWS) showed low scatter, with slopes of 1.037 and 1.025, for the two deployments.

Figure 2. Sonic Direction vs. Day for 1st Deployment

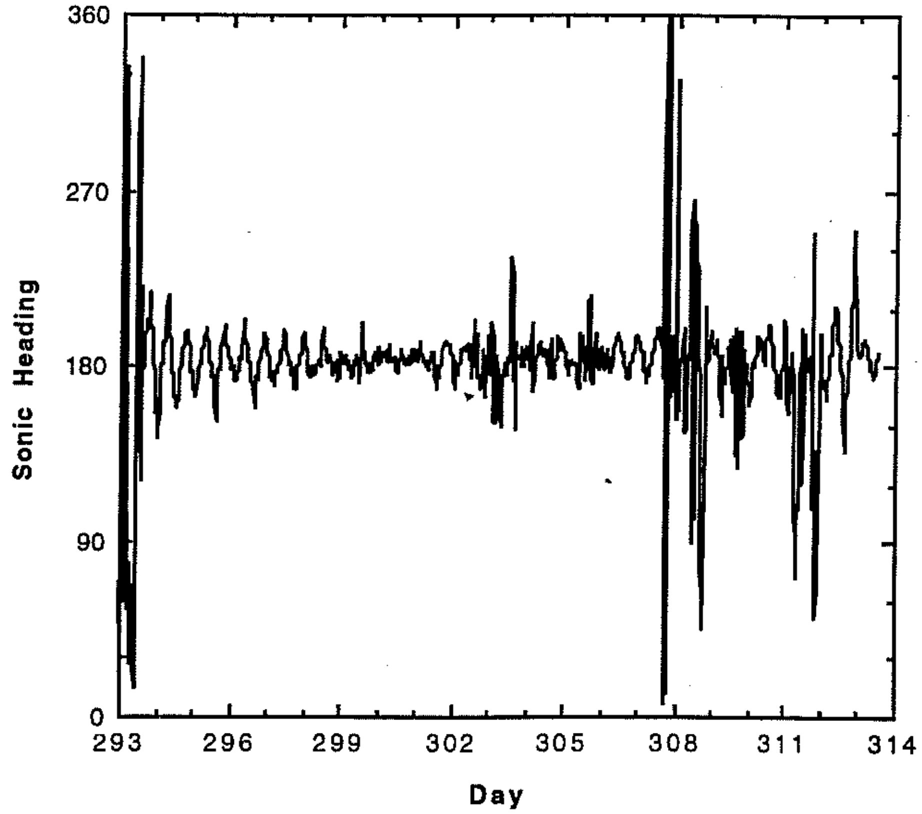

Figure 3. Sonic Direction vs. Day for 2nd Deployment

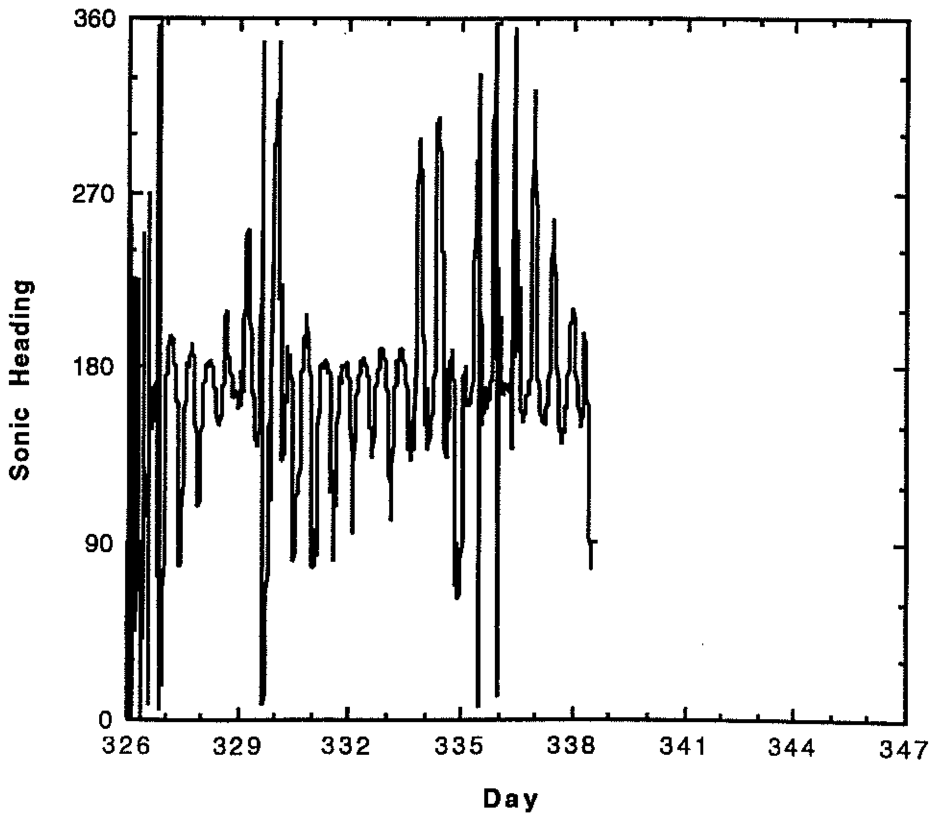

Figure 4. Sonic MWS and Wind Direction (to) vs. Day for 1st Deployment

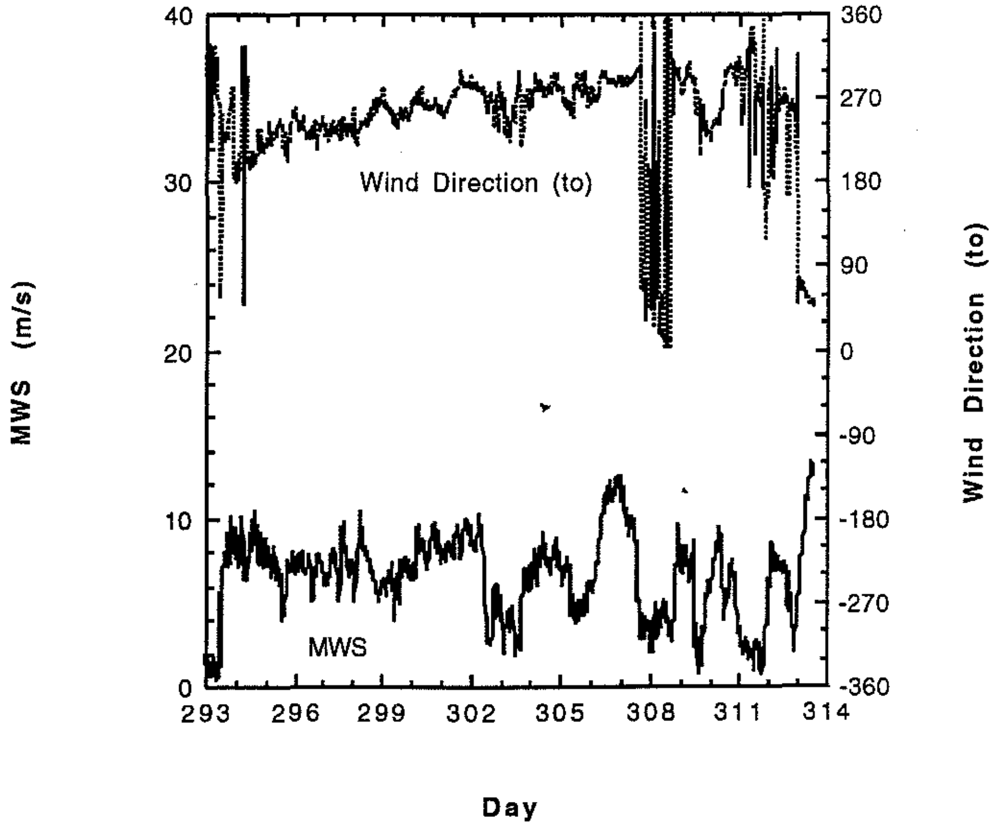

Figure 5. Sonic MWS and Wind Direction (to) vs. Day for 2nd Deployment

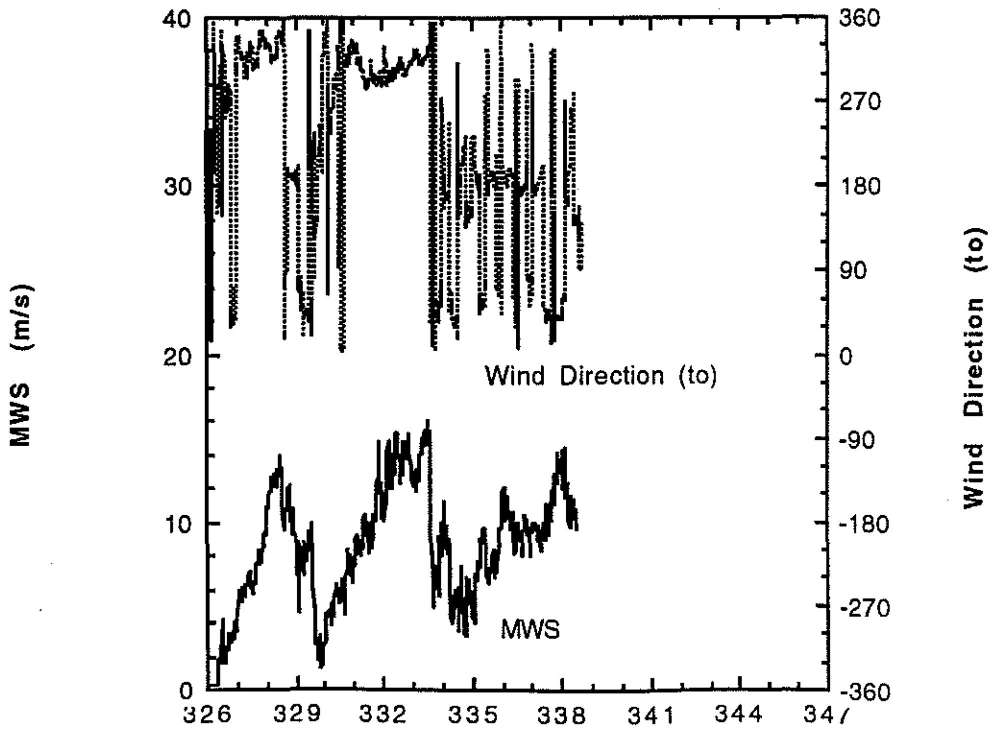

Figure 6. Air and Sea Temperatures vs. Day for 1st Deployment

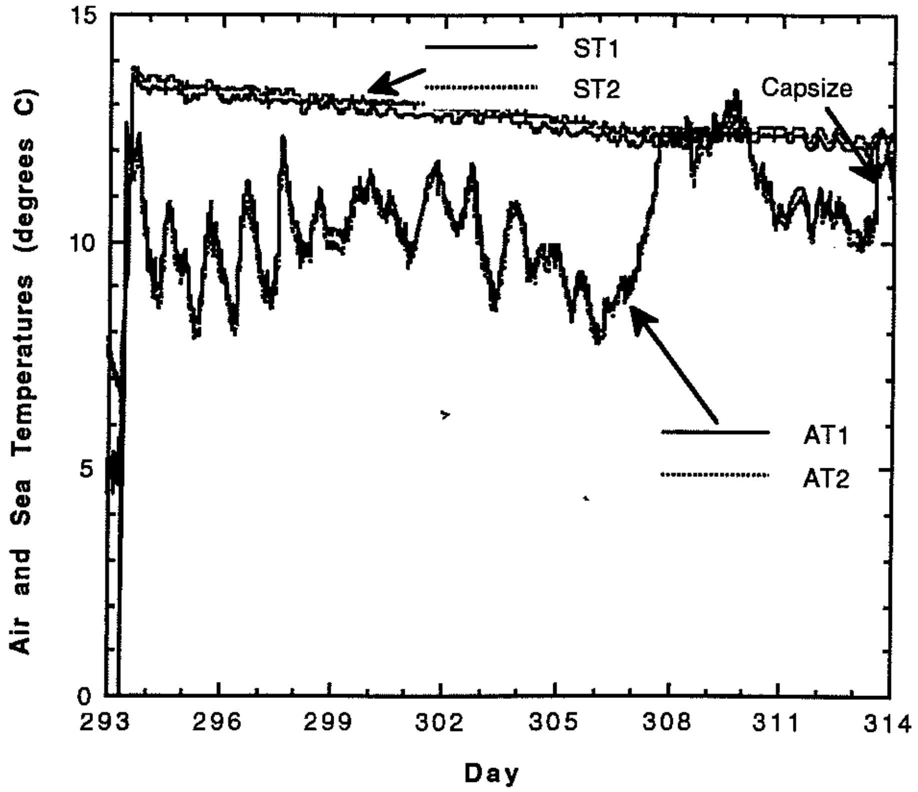

Figure 7. Air and Sea Temperatures vs. Day for 2nd Deployment

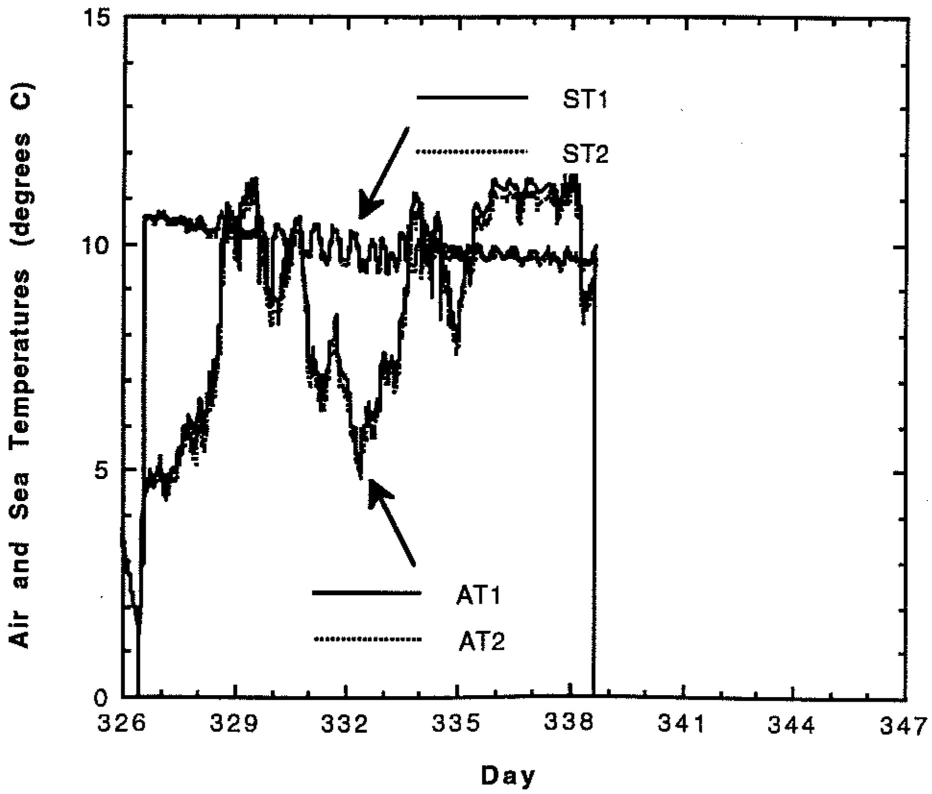

Figure 8. Air Temperature Differences vs. Day for 1st Deployment

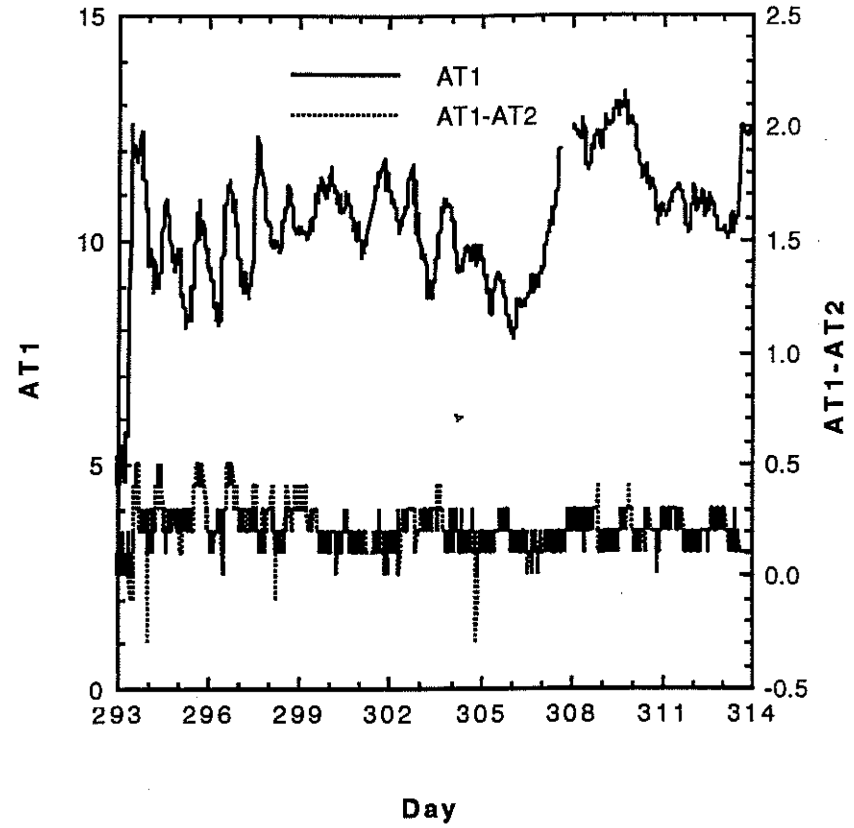

Figure 9. Air Temperature Differences vs. Day for 2nd Deployment

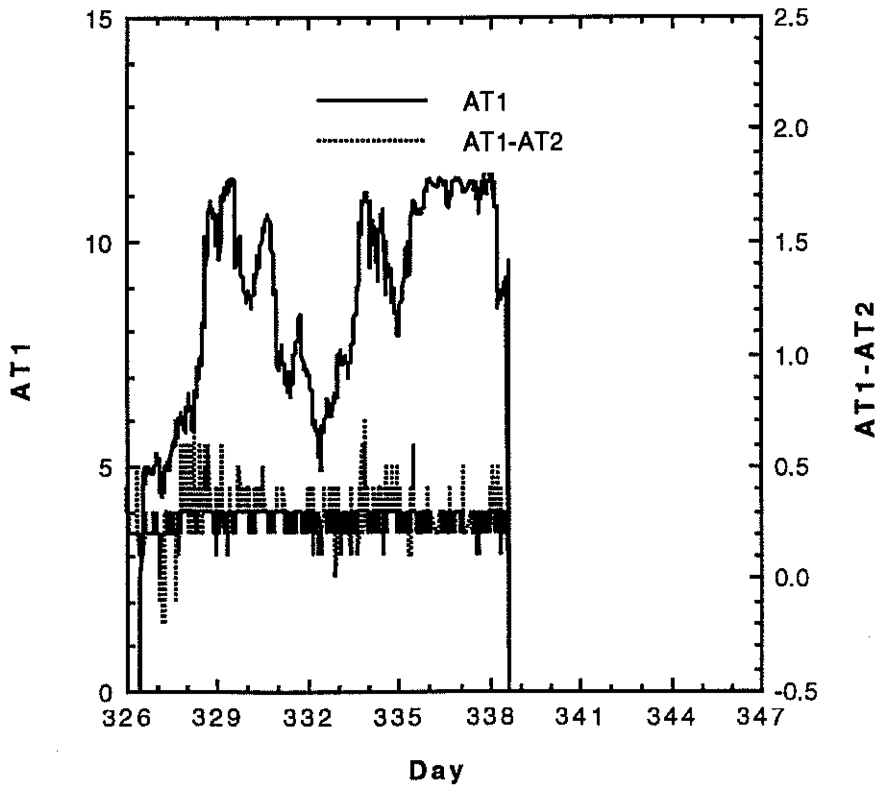

Figure 10. Sea Temperature difference vs. Day for 1st Deployment

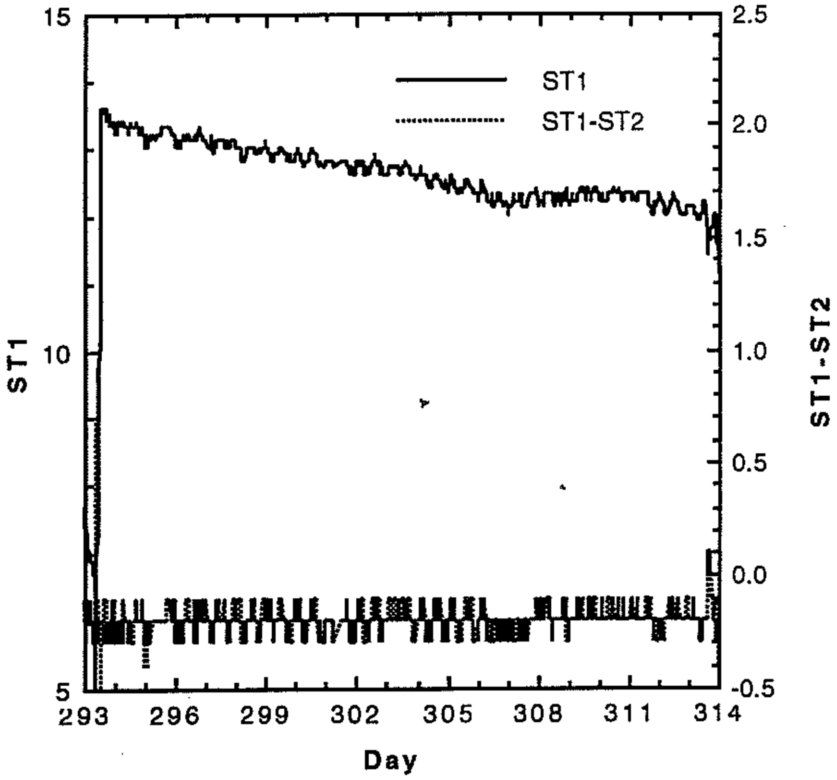

Figure 11. Sea Temperature Difference vs. Day for 2nd Deployment

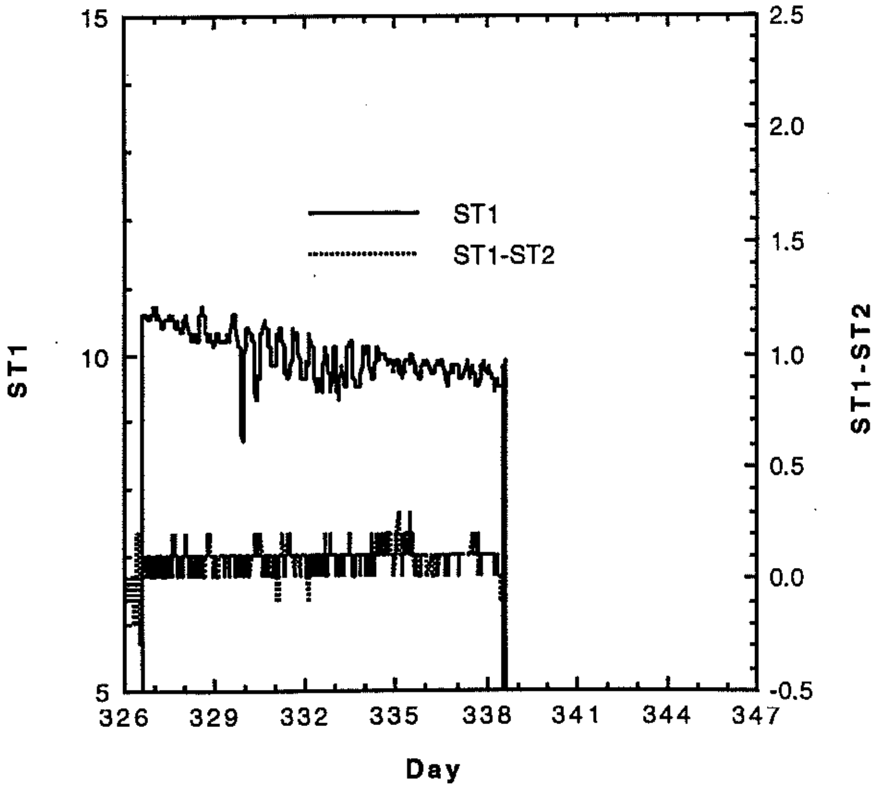

Figure 12. Young WS vs. Sonic MWS for 1st Deployment

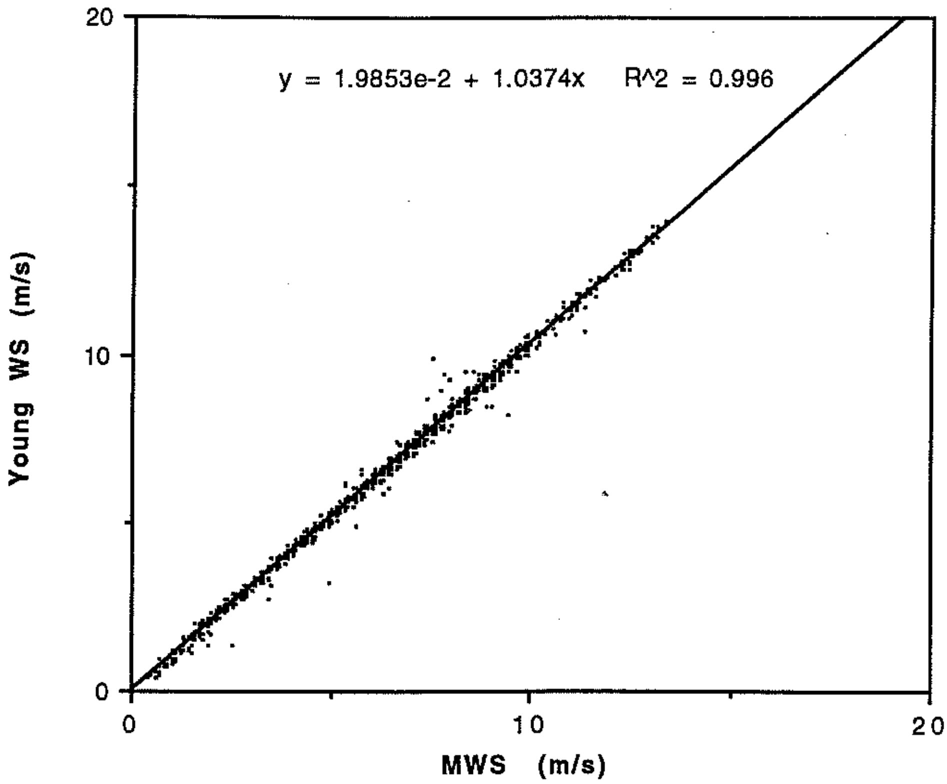

Figure 13. Young WS vs. Sonic WS for 2nd Deployment

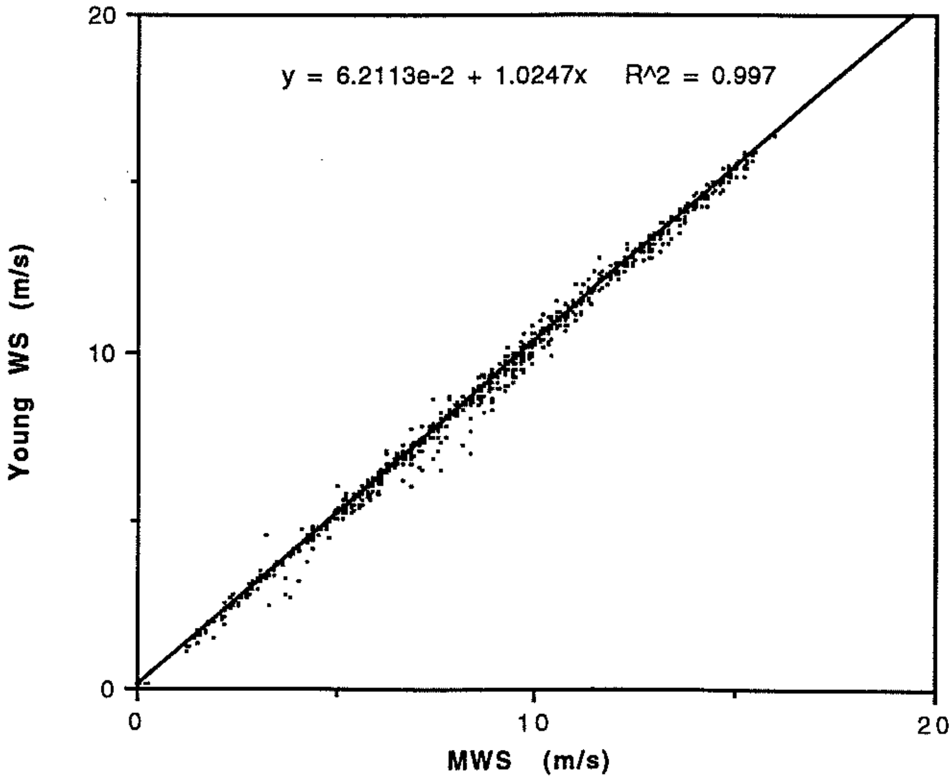

Figure 14. Young and Sonic Wind Direction Differences vs. Day for 2nd Deployment

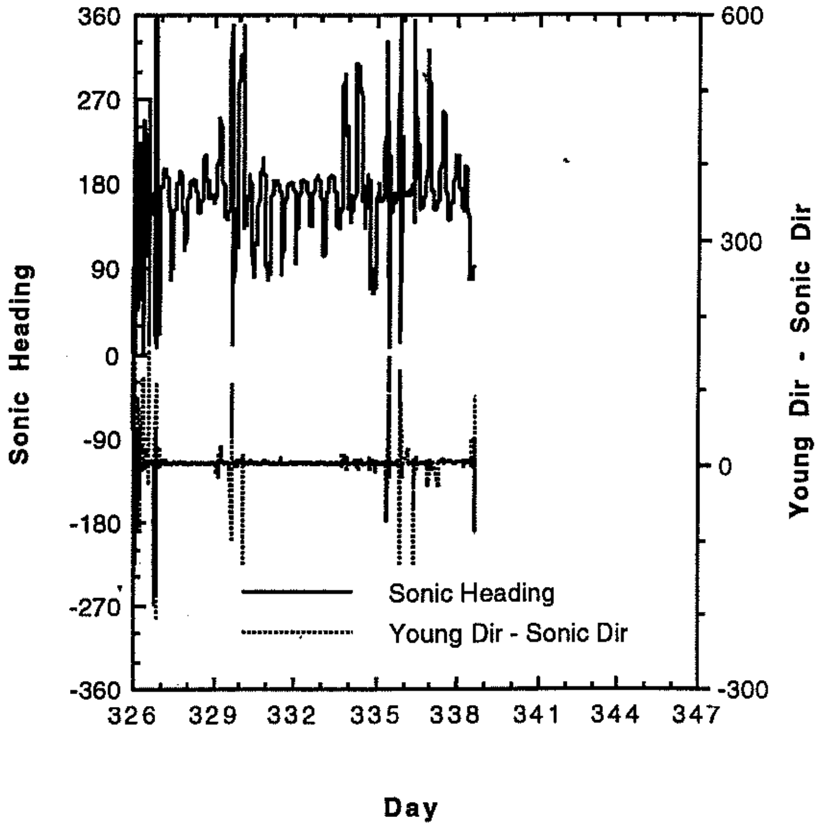

Figure 15. Sonic Vertical WS/MWS and Direction vs. Day for 1st Deployment

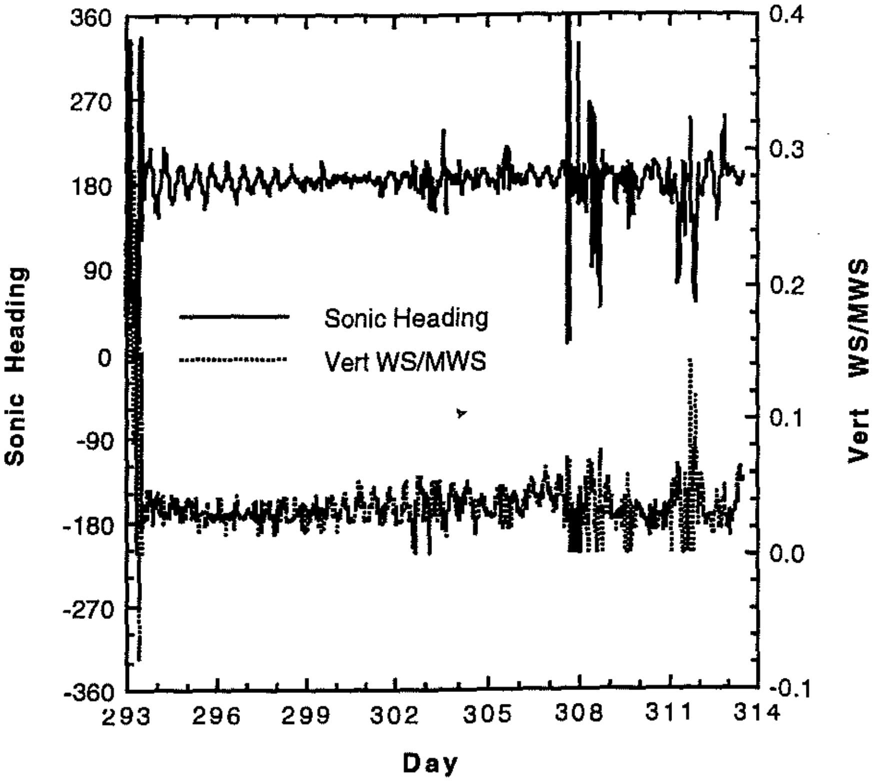

Figure 16. Sonic VWS/MWS and Direction vs. Day for 2nd Deployment

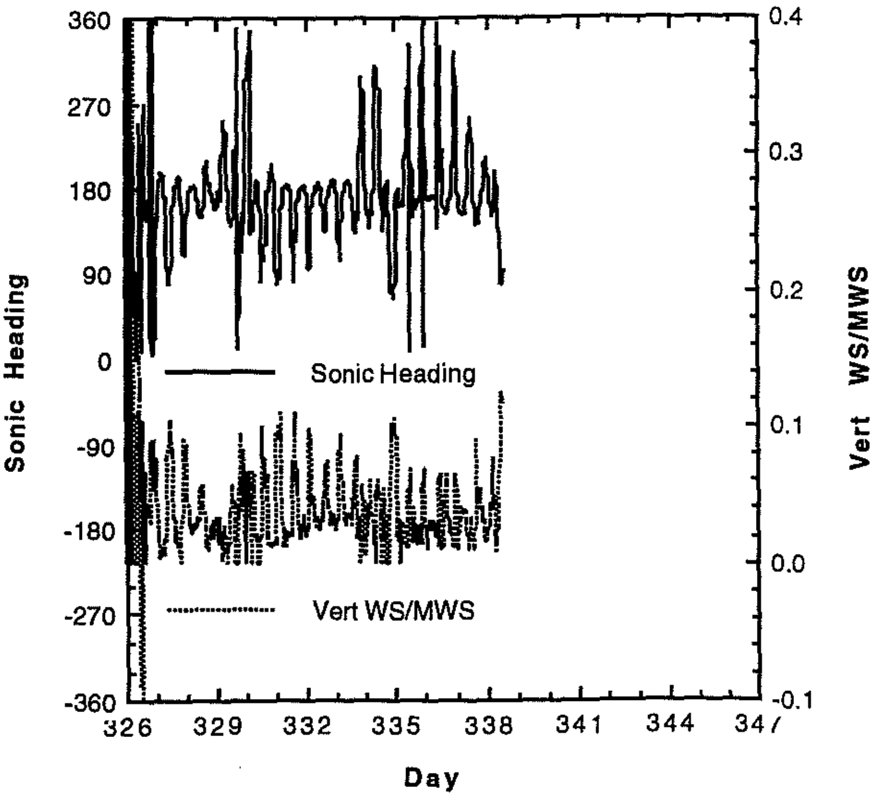

Figure 17. Calculated Wind Direction (to) and r.m.s. heading vs. Day for 1st Deployment

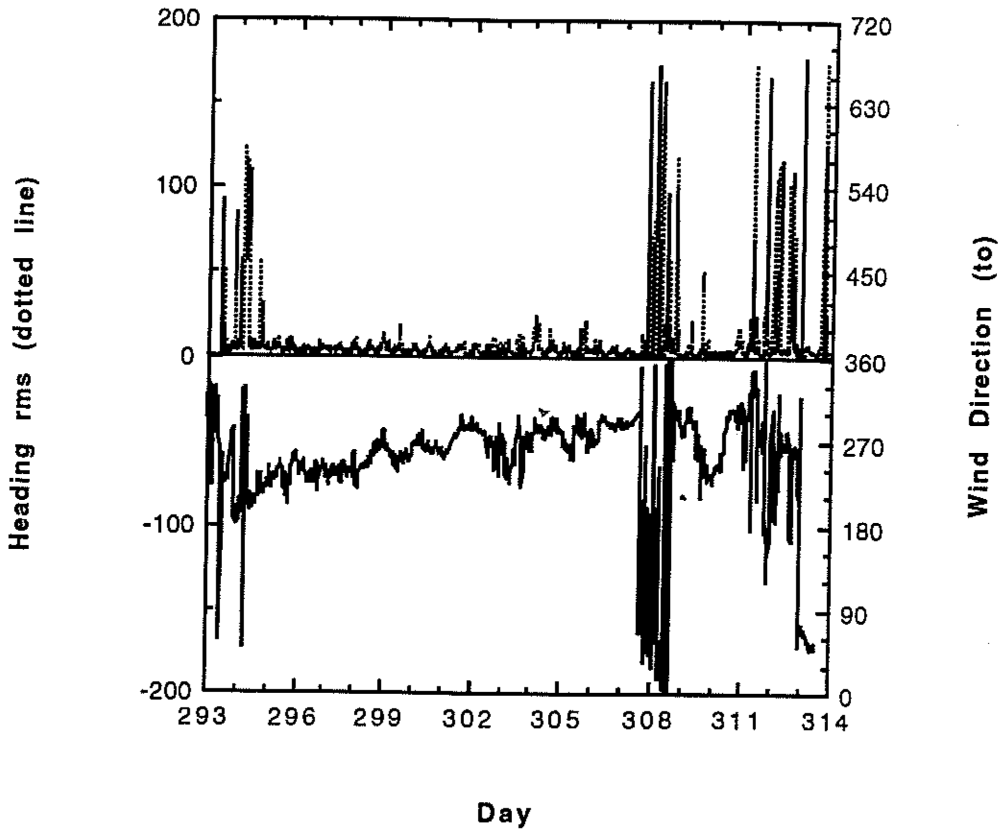

Figure 18. Calculated Wind Direction (to) and r.m.s. heading vs. Day for 2nd Deployment

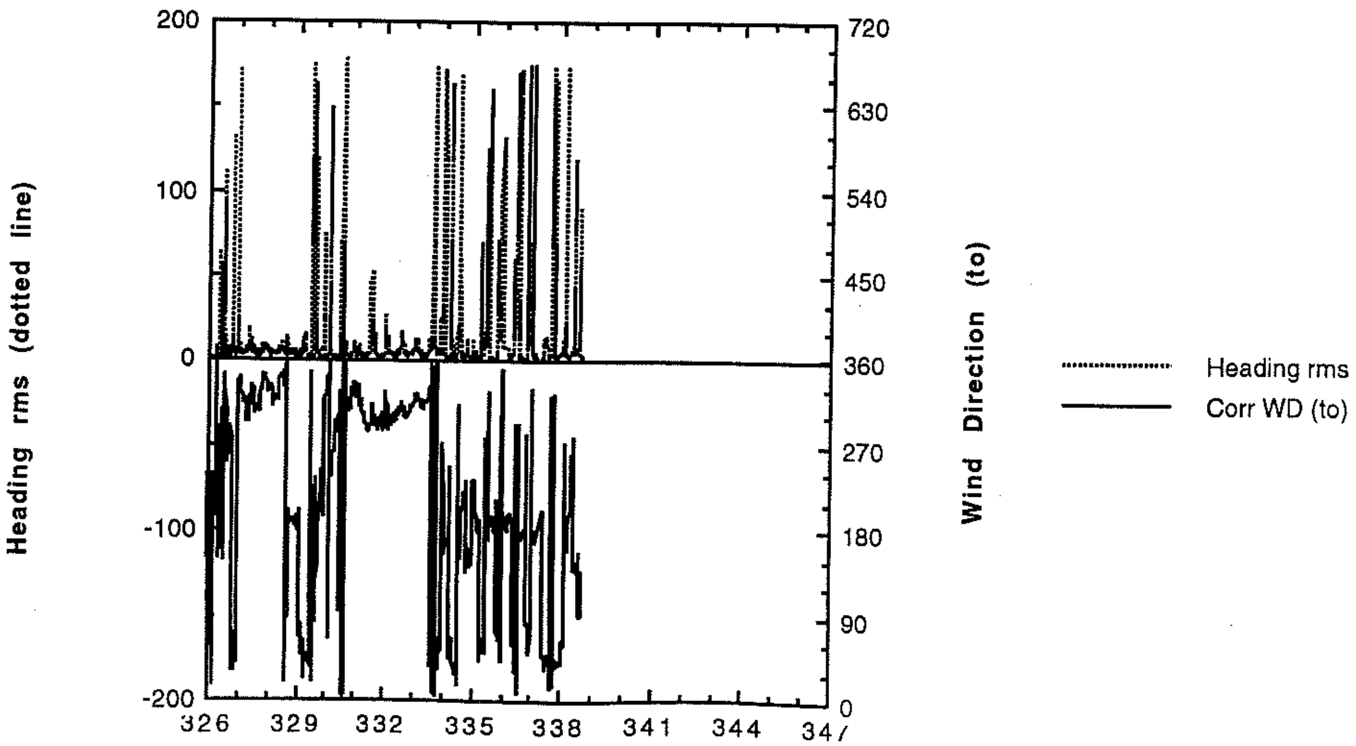

Figure 14 shows the difference between the relative wind directions as measured by the Sonic and Young sensors during the second deployment; Sonic Direction is also shown for reference. This shows the errors in the Young directions where direction passed through the $360^o/0^o$ discontinuity, due to the lack of quadrant direction averaging in the Multimet wind direction processing.

From the ratio of Vertical WS (VWS) to MWS, a measure of Sonic sensor vertical axis alignment can be achieved. Double axis plots of VWS/MWS and Sonic Direction (figures 15, 16) showed an average VWS/MWS of approximately 0.02 (corresponding to about 1^o error in alignment), with some correlation of the two variables. The correlation was particularly noticeable for the second deployment data, when larger deviations from buoy alignment with the wind direcion occurred. This suggests that the sensor axis was about 5^o from vertical in the buoy East-West plane; this could have occurred due to a static or wind-induced list of the buoy, the sensor alignment on the mast was unlikely to have been more than about 1^o from vertical.

Summary of Data Produced

Raw Data Files

The raw PCMCIA Flash Card data are in three binary files

FORMSWAL.1MG	Directory and 1st deployment to day 306.2500
FORMSWAL.2MG	remainder of 1st deployment and most of 2nd
FORMSWAL.3MG	end of 2nd deployment (not useful due to low batteries)

The fourth file, FORMSWAL.4MG, was not retained as the data all lies within the first three files

CricketGraph Data

1st Deployment CG final	Day 293.0000 - Day 314.0313
2nd Deployment CG final	Day 325.5521 - Day 338.7083

Data Time Stamping

The Multimet Real Time Clock, being a battery backed up hardware clock unaffected by interrupt conflicts and being known to have a history of good stability, was the best on-board clock. It was checked on day 356 after the final recovery and found to have lost 197 seconds over the 31 days since it was previously set up on day 325.

Timing checks of the Multimet Real Time Clock and of the Sonic Processor system clock relative to the Formatter clock are included in the Meteosat data (+TMET and +TSON); these have +ve sign if fast relative to the Formatter.

During the first deployment, the initial values of +TMET and +TSON on day 293 were 0 and +1 minutes and, just prior to the capsize on day 313, the final values were +2 and +1 minutes. Assuming a linear drift of the Multimet clock, it would have been 127 seconds slow on day 313. From this one can deduce that, on day 313, the Formatter clock was 4 (± 1) minutes slow and the Sonic Processor clock was 3 (± 1) minutes slow.

During the second deployment, the initial values of +TMET and +TSON on day 325 were 0 and +1 minutes and on day 338, prior to battery failure, the values were +1 and -5406 minutes; the latter figure resulted from the progressive clock jumps due to the RTCN application. Assuming a linear drift of the Multimet clock, it would have been 53 seconds slow on day 338. From this one can deduce that, on day 338, the Formatter clock was 2 minutes slow. This demonstrates consistency in the Formatter drift rates of approximately -12 (± 3) seconds/day for the first deployment and -9 (± 2) seconds/day for the second deployment.

In producing the tabular (CricketGraph) data file for the first deployment, the time stamps given in 'Day' were simply derived from the Sonic message time stamps which were, in turn, derived from the FASTCOM RAMdisk file data header, i.e. the Sonic data acquisition start time from the Sonic Processor system clock. Thus one could apply a linear time correction varying from +1 minute to +3 minutes over the period day 293 to day 313; this has not been applied, as it was considered barely significant.

In producing the tabular (CricketGraph) data file for the second deployment, the time stamps given in 'Day' were derived from a combination of the Sonic message time stamps and the Formatter clock time stamps. The result of this is that there may be occasional time errors of up to ± 5 minutes in individual records but, overall, the time correction, if applied, should be from +1 minute to +2 minutes over the period day 325 to day 338; again this has not been applied, as it was considered barely significant.

Acknowledgements

The SWALES data set was the result of the concerted efforts of many, including the IOSDL Centre for Ocean Technology Development members of the Met Team, the IOSDL Moorings Team and the JRC members of the Met Team. The experimental work was funded by the MAFF Flood and Coastal Defence Division under commission FD0603; analysis of the data will be under commission FD0601.

References

1. Clayson, C.H. 1994, Sonic Buoy Formatter Handbook, IOSDL Internal Document

2. Clayson, C.H. and Pascal, R.W. 1994, SWALES Sonic Buoy - Sonic Anemometer Spectral and Raw Data Report, IOSDL Internal Document

APPENDIX A SOFTWARE LISTINGS

A.1 ARGSONFILE

```
REM QuickBasic program to decode ARGOS Dispose File data
REM copied from Telnet into engineering data
REM
REM Use program SORT RECS to further process into final data
REM
REM Author CHC Date 21-09-1993

DIM b$(8)
DIM w(32)
DIM day%(5), hrs%(5), mins%(5), mdays%(12)
DIM psd(5), mws(5)
DIM fita(5),va(5)

ON ERROR GOTO Handler    'for opening new output file

REM load days of month array (used to find Julian day)
FOR n% = 1 TO 12:READ mdays%(n%):NEXT
DATA 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30

INPUT"Enter filename for input data:";f$
OPEN f$ FOR INPUT AS #1
Outfile:
cflag% = 0
INPUT"Enter filename for output data:";g$
10 FILES g$    ' results in error if not existing already, handled by Handler
IF cflag% = 0 THEN
   INPUT "This file exists.  Append data";r$
   IF (r$ = "n") OR (r$ = "N") THEN GOTO Outfile
END IF
OPEN g$ FOR APPEND AS #2

REM process the whole file
WHILE NOT EOF(1)
REM First check for start line and correct PTT
Readheader:
IF EOF(1) THEN END
LINE INPUT#1, h$
IF (LEFT$(h$,11) <> "00296 05060") THEN GOTO Readheader

CLS
PRINT "PTT: ";MID$(h$, 7, 5);

REM start line will be over 30 chars if it contains a fix
IF LEN(h$) < 30 THEN fixflag% = 0 ELSE fixflag% = 1

REM there will be nlines%-1 of data, 8 lines per frame
nlines% = VAL(MID$(h$, 14, 2)):nframes% = (nlines% - 1)/8
PRINT " Lines:";nlines%;" Frames:";nframes%

IF fixflag% = 1 THEN
   fixtime$= MID$(h$, 24, 19)
   lat$=MID$(h$, 45, 6):long$=MID$(h$, 53, 7)
   PRINT "Fix date/time "; fixtime$
```

```
  PRINT "Latitude:"; lat$; " Longditude:"; long$
END IF

REM now get the data, decoding each frame individually into 5 records
REM note that acqtime$ is the time of reception of an individual frame

FOR frame% = 1 TO nframes%
  FOR m% = 1 TO 8
    INPUT#1, b$(m%)
    IF m% = 1 THEN acqtime$= MID$(b$(m%), 1, 19):PRINT acqtime$
      FOR n% = 1 TO 4
        REM 1st line is decimal, others are hex
        IF (m% > 1) THEN
          b$(m%) = RIGHT$(b$(m%), 41)
          p% = 1+13*(n%-1):n$=MID$(b$(m%), p%, 2)
          n1% = ASC(LEFT$(n$,1))
          IF (n1% < 58) THEN n1% = n1% - 48 ELSE n1% = n1% - 55
          n2% = ASC(RIGHT$(n$, 1))
          IF (n2% < 58) THEN n2% = n2% - 48 ELSE n2% = n2% - 55
          w(4 * (m% - 1) + n%) = 16 * n1% + n2%
        ELSE
          b$(m%) = RIGHT$(b$(m%), 42)
          p% = 1+13*(n% - 1):n$=MID$(b$(m%), p%, 3)
          w(4 * (m% - 1) + n%) = VAL(n$)
        END IF
      NEXT n%
  NEXT m%

  REM w(1) to w(32) now contain the 32 bytes of the frame
  REM first calculate the acquisition date/time information
  ayear% = VAL(LEFT$(acqtime$, 4))
  aday% = VAL(MID$(acqtime$, 9, 2))
  amonth% = VAL(MID$(acqtime$, 6, 2))
  ahr% = VAL(MID$(acqtime$, 12, 2))
  amin% = VAL(MID$(acqtime$, 15, 2))
  asec% = VAL(MID$(acqtime$, 18, 2))
  ahr = ahr% + amin%/60 + asec%/3600

  jday% = 0
  FOR n% = 1 TO amonth%
    jday% = jday% + mdays%(n%)
    IF (n% = 3) AND (INT(ayear%/4) = 0) THEN jday% = jday% + 1
  NEXT
  jday% = jday% + aday%

  PRINT "Acquisition Day:"; jday%;" Time in hrs:";
  PRINT USING "##.###"; ahr

  REM now decode the frame into the 5 records of
  REM starttime (hrs%,mins%), PSD, MWS, Fit_A, V_MWS, N2
  REM inserting 999 type values where parity errors are detected

  FOR rec% = 1 TO 5
    n% = 6 * rec%
    word& = w(n% - 5)   'word& is the quarter-hours since midnight
    bits% = 8: GOSUB Paritycheck
    IF word& < 99 THEN
      hrs%(rec%) = INT(word&/4)
      mins%(rec%) = 15*(word& - 4*hrs%(rec%))
    ELSE
      hrs%(rec%) = 99:mins%(rec%) = 99
    END IF
```

```
word& = w(n% - 4)*4 + (w(n% - 3) AND 192)/64    'PSD
bits% = 10: GOSUB Paritycheck
IF word& < 9999 THEN
  psd(rec%) = .01*word& - 6
ELSE
  psd(rec%) = -9.99
END IF

word& = (w(n% - 3) AND 63)*16 + (w(n% - 2) AND 240)/16    'MWS
bits% = 10: GOSUB Paritycheck
IF word& < 9999 THEN
  mws(rec%) = .1*word&
ELSE
  mws(rec%) = 99.9
END IF

word& = (w(n% - 2)AND 15)*64 + (w(n% - 1)AND 252)/4    'FIT_A
bits% = 10: GOSUB Paritycheck
IF word& < 9999 THEN
  fita(rec%) = .01*word& - 6
ELSE
  fita(rec%) = -9.99
END IF

word& = (w(n% - 1) AND 3)*256 + w(n%)    'V_MWS
bits% = 10: GOSUB Paritycheck
IF word& < 9999 THEN
  vm(rec%) = .02*(word& - 256)
ELSE
  vm(rec%) = +9.99
END IF

NEXT rec%

word& = w(31) * 256 + w(32)    '16 bit word for N2 values
bits% = 16: GOSUB Paritycheck
IF (word& < 99999&) THEN
  n% = 4096
  FOR rec% = 1 TO 5
    n2(rec%) = INT(word& / n%) AND 7
    n% = n%/8
  NEXT
ELSE
  FOR rec% = 1 TO 5
    n2(rec%) = 9
  NEXT
END IF

PRINT "DAY HH:MM +P.SD MW.S +F.IT +V.MW N"

REM impute the data day number from the acq day and the record time
FOR rec% = 1 TO 5
  rhr = hrs%(rec%) + mins%(rec%)/60
  IF ABS(ahr - rhr) > 6 THEN
    day%(rec%) = jday% - 1
    IF day%(rec%) = 0 THEN day%(rec%) = 365
  ELSE
    day%(rec%) = jday%
  END IF

  REM print to screen in format DAY HH:MM +PSD MW.S +F.IT +V.MW N
  REM julian day, hours and minutes of data start time + parameters
  REM PSD MWS Fit_A, Vert _MWS and N2
```

```
        REM
        REM print data to output file  in format
        REM
JDA.YREC<T>JDA.YACQ<T>LA.TITU<T>LON.GDIT<T>+P.SD<T>MW.S<T>+F.IT<T>+V.MW
<T>N<CR>
        REM where <T> is a TAB character and <CR> is Carriage Return

        PRINT USING "### ";day%(rec%);
        PRINT USING "##:";hrs%(rec%);
        PRINT USING "## ";mins%(rec%);

        IF (hrs%(rec%) < 99) AND (mins%(rec%) < 99) THEN
          PRINT #2, USING "###.####";day%(rec%) + hrs%(rec%)/24 + mins%(rec%)/1440;
        ELSE
          PRINT#2,"999.9999";
        END IF
        PRINT#2,CHR$(9);

        IF (fixflag% = 1) THEN
          PRINT #2, USING "###.####";jday% + ahr%/24 +amin%/1440 + asec%/86400&;
          PRINT#2,CHR$(9);
          PRINT #2, lat$;CHR$(9);long$;CHR$(9);
        ELSE
          PRINT #2,"999.9999";CHR$(9);"99.999";CHR$(9);"999.999";CHR$(9);
        END IF

        PRINT USING "+#.## ";psd(rec%);
        PRINT #2, USING "+#.##";psd(rec%);
        PRINT#2,CHR$(9);

        PRINT USING "##.# "; mws(rec%);
        PRINT #2, USING "##.#";mws(rec%);
        PRINT#2,CHR$(9);

        PRINT USING "+#.## ";fita(rec%);
        PRINT #2, USING "+#.##";fita(rec%);
        PRINT#2,CHR$(9);

        PRINT USING "+#.## ";vm(rec%);
        PRINT #2, USING "+#.##";vm(rec%);
        PRINT#2,CHR$(9);

        PRINT USING "#"; n2(rec%)
        PRINT #2, USING "#"; n2(rec%)
    NEXT rec%
NEXT frame%
WEND

CLOSE#1
CLOSE#2
END

REM Subroutines
Paritycheck:
    REM checks for even parity
    p% = 0:b& = 1
    FOR bit% = 1 TO bits%
        IF (word& AND b&) THEN p% = p% XOR 1
        b& = b&*2
    NEXT

    IF p% = 0 THEN
        word& = word& AND (2^(bits% - 1) - 1)
```

```
  ELSEIF bits% = 8 THEN
    word& = 99
  ELSEIF bits% = 10 THEN
    word& = 9999
  END IF
  IF p% = 0 AND bits% = 16 THEN word& = 99999&
  REM at present error in n2 parity bit
RETURN


Handler:
  IF (ERL = 10) AND (ERR = 53) THEN
    OPEN g$ FOR OUTPUT AS #2
    cflag% = 1
    CLOSE#2
  END IF
  RESUME NEXT
```

A.2 SORT RECS

```
REM QuickBasic Program SORT RECS
REM - this sorts Sonic Buoy ARGOS data (which has already been
REM decoded from DS format by the program ARGSONFILE)
REM  into chronological order and selects the best
REM (lowest weighted error) message if duplicates exist.
REM
REM Produces a file suitable for import into CricketGraph
REM
REM Author CHC Date 23-09-1993

REM Can process a file contaioning up to 1000 messages
DIM day(1000), flag%(1000),indx%(1000)

ON ERROR GOTO Handler    'for opening new output file

INPUT "Enter name of file to be sorted:";f$
OPEN f$ FOR INPUT AS #1
Outfile:
cflag% = 0
INPUT "Enter filename for output data:";g$
10 FILES g$    ' results in error if not existing already, handled by Handler
IF cflag% = 0 THEN
  INPUT "This file exists. Append data";r$
  IF (r$ = "n") OR (r$ = "N") THEN GOTO Outfile
END IF
OPEN g$ FOR APPEND AS #2

FOR n% = 1 TO 1000:flag%(l%) = 0:NEXT

REM First find the number of messages, lin%,
REM and allot a weighted error flag%() to each message
l% = 1
WHILE NOT EOF(1)
  Getline:
  LINE INPUT#1, h$
  IF LEFT$(h$,8) = "999.9999" THEN
    day(l%) = VAL(LEFT$(h$,8))
    flag%(l%)=15: l%=l%+1
    GOTO Getline
  END IF
  day(l%)=VAL(LEFT$(h$,8))
```

```
    IF MID$(h$,10,1) = "9" THEN flag%(l%) = flag%(l%) + 1   'no fix
    IF MID$(h$,35,1) = "9" THEN flag%(l%) = flag%(l%) + 4   'no psd
    IF MID$(h$,41,1) = "9" THEN flag%(l%) = flag%(l%) + 4   'no mws
    IF MID$(h$,46,1) = "9" THEN flag%(l%) = flag%(l%) + 3   'no fit_a
    IF MID$(h$,52,1) = "9" THEN flag%(l%) = flag%(l%) + 2   'no v_mws
    IF MID$(h$,57,1) = "9" THEN flag%(l%) = flag%(l%) + 1   'no n2
    l%=l%+1
  WEND
  CLOSE#1

n% = l% - 1: lin%=n%
PRINT "File contains ";lin%;" lines of data"

REM Now sort into chronological order by producing an index table
REM Method from Press, Flannery et al. "The Art of Scientific Computing"
FOR j% = 1 TO n%:indx%(j%) = j%:NEXT
IF (n% = 1) THEN END
l% = n%/2 + 1
ir% = n%

WHILE (ir% > 1)
  IF (l% > 1) THEN
    l% = l% - 1
    indxt% = indx%(l%)
    q = day(indxt%)
  ELSE
    indxt% = indx%(ir%)
    q = day(indxt%)
    indx%(ir%) = indx%(1)
    ir% = ir%-1
    IF (ir% = 1) THEN indx%(1) = indxt%
  END IF
  i% = l%:j% = 2*l%
  WHILE (j% <= ir%)
    IF (j% < ir%) AND (day(indx%(j%)) < day(indx%(j%+1))) THEN j% = j%+1
    IF (q < day(indx%(j%))) THEN
      indx%(i%) = indx%(j%)
      i% = j%
      j% = j% + i%
    ELSE
      j% = ir% + 1
    END IF
  WEND
indx%(i%) = indxt%
WEND

REM Now use the index table to identify groups messages having
REM the same day/time and to select the one in each group with
REM the lowest weighted error flag
REM This message  is then written to the output file in the
REM correct format for use in CricketGraph, etc.

OPEN f$ AS #1 LEN=58
FIELD#1, 8 AS day$,1 AS t$,8 AS aqd$,1 AS t$,6 AS lat$,1 AS t$,7 AS long$,1 AS t$,5 AS psd$,1
AS t$,4 AS mws$,1 AS t$,5 AS fita$,1 AS t$,5 AS vmws$,1 AS t$,1 AS n2$,1 AS cr$
lastday$ = "": stflag% = 0
FOR n% = 1 TO lin%
  dayn = day(indx%(n%))
  IF (n% = 1) THEN
    lastday = dayn
    n1% = n%
  ELSE
    IF (dayn <> lastday) THEN
```

```
        n2% = n% - 1
        best% = 100
        PRINT n% - n1%;" duplicate(s)"
        FOR p% = n1% TO n2%
          IF flag%(indx%(p%)) < best% THEN
            best% = flag%(indx%(p%))
            bestp% = p%
          END IF
        NEXT
        GET#1, indx%(bestp%)
        PRINT day$;" ";aqd$;" ";lat$;" ";long$;" ";psd$;" ";mws$;" ";fita$;" ";vmws$;" ";n2$
        PRINT#2,
day$;CHR$(9);aqd$;CHR$(9);lat$;CHR$(9);long$;CHR$(9);psd$;CHR$(9);mws$;CHR$(9);fita$;C
HR$(9);vmws$;CHR$(9);n2$
        lastday = dayn
        n1% = n%
      END IF
    END IF
    lastday$ = day$
NEXT n%
CLOSE#1
CLOSE#2

END

Handler:
    IF (ERL = 10) AND (ERR = 53) THEN
      OPEN g$ FOR OUTPUT AS #2
      cflag% = 1
      CLOSE#2
    END IF
    RESUME NEXT
```

A.3 METEO SORT

```
REM QuickBasic program METEOSORT
REM - sorts Meteosat Sonic Buoy data
REM in a day folder and produces an error file

REM Use the error file to show errors
REM then edit them and re-run this prog

REM 07-10-93
REM CHC

ON ERROR GOTO Handler

DIM l$(4)
DIM var(4,22)
DIM flag%(5)

REM open output file (Sorted)
INPUT "Enter day number (Julian) to be analysed";d$
f$ = "Wooig8-CHC:CHC-mac:Swales:meteo-" + d$ + ":Sorted"
g$ = "Wooig8-CHC:CHC-mac:Swales:meteo-" + d$ + ":Errors"
OPEN f$ FOR OUTPUT AS #2
OPEN g$ FOR OUTPUT AS #3

REM process all the files in the folder "meteo-" + d$
FOR hh% = 0 TO 23
    hh$ = STR$(hh%)
```

```
IF (hh% < 10) THEN MID$(hh$,1,1) = "0" ELSE hh$ = RIGHT$(hh$,2)
f$ = "Wooig8-CHC:CHC-mac:Swales:meteo-" + d$ + ":"+ hh$
OPEN f$ FOR INPUT AS #1

REM get the ESA header line
LINE INPUT #1, h$
IF (LEFT$(h$,4) = "\M /") THEN
  h$ = LEFT$(h$,40)    ' the useful part of the ESA header
  PRINT h$
  jday% = VAL(MID$(h$, 24, 3)): hr% = VAL(MID$(h$, 28, 2))
ELSE
  jday% = 999:hr% = 99
END IF

REM get the IOS header lines (B01, etc.)
ld% = 0
FOR l% = 1 TO 3
  LINE INPUT#1, h$
  PRINT h$
  IF (INSTR(h$, ",") > 0) THEN ld% = l%:l% = 3
NEXT
IF (VAL(h$) = jday%) THEN
  PRINT "Header Format OK"
ELSE
  PRINT "Header Format Faulty!"
  PRINT#3, hh%; " Header Format Faulty"
END IF
IF (ld% > 0) THEN
  PRINT "Missing line, re-reading 1st ";ld%
  PRINT#3, hh%; " Missing line(s)"
  CLOSE#1
  OPEN f$ FOR INPUT AS #1
  FOR l% = 1 TO ld%
    LINE INPUT #1, h$
  NEXT
END IF

REM the next 4 lines should be quarter-hourly data
try% = 0
FOR qhr% = 1 TO 4
  var(qhr%,1) = jday% + hr% / 24!: var(qhr%,2) = hr%
  LINE INPUT#1, l$(qhr%)
  IF (INSTR(l$(qhr%), ",") = 0) AND (try% < 4) THEN
    qhr% = 0:try% = try% + 1
    PRINT "Excess/faulty header line"

    GOTO Retry
  END IF
  PRINT l$(qhr%)
  sum% = 0
  FOR ch% = 1 TO 59
    IF (MID$(l$(qhr%), ch%, 1) = ",") THEN sum% = sum% + 1
  NEXT

  IF (sum% = 12) THEN
    flag%(qhr%) = 0: cpos% = 0
    FOR par% = 1 TO 13
      CALL CheckNext(qhr%, par%)
    NEXT
    REM PRINT
  ELSE
    PRINT "Line too corrupted to analyse"
    PRINT#3, hh%; " Line ";qhr%; " corrupted"
```

```
        flag%(qhr%) = -9999
      END IF
      var(qhr%,21) = flag%(qhr%)
      Retry:
    NEXT qhr%

    REM now get the housekeeping data line
    LINE INPUT#1, hskeep$
    PRINT hskeep$
    flag%(5) = 0: cpos% = 0
    FOR par% = 1 TO 5
      CALL CheckHskeep(par%)
    NEXT

    REM finally save to CricketGraph format output file
    FOR qhr% = 1 TO 4
      var(qhr%,22) = flag%(5)
      FOR par% = 1 TO 22
        PRINT#2, var(qhr%, par%);CHR$(9);
      NEXT
      PRINT#2, ""

      REM continue with next input file
    NEXT

    CLOSE #1

Missing:
NEXT hh%

CLOSE#2
CLOSE#3
END


SUB CheckNext(l%, par%) STATIC
SHARED l$(), var(), cpos%, flag%()
      l$(l%) = l$(l%) + CHR$(10)
      IF (par% = 13) THEN t$ = CHR$(10) ELSE t$ = ","

      parlen% = 4:den% = 10
      IF (par% = 1) THEN parlen% = 2:den% = 4
      IF (par% = 3) OR (par% = 12) OR (par% = 13) THEN parlen% = 3
      IF (par% = 2) OR (par% = 7) THEN den% = 100
      IF (par% = 13) THEN den% = 1
      cpos1% = INSTR(cpos% + 1, l$(l%), t$)
      REM PRINT cpos1%;" ";
      IF ((cpos1% - cpos%) = (parlen% + 1)) THEN
        var(l%, par% + 2) = VAL(MID$(l$(l%), cpos% + 1, parlen%)) / den%
      ELSE
        var(l%, par% + 2) = VAL(STRING$(parlen%, "9"))
        flag%(l%) = flag%(l%) + 2^(par% - 1)
      END IF
      cpos% = cpos1%
END SUB

SUB CheckHskeep(par%) STATIC
SHARED hskeep$, var(), cpos%, flag%
      hskeep$ = hskeep$ + CHR$(10)
      IF (par% = 5) THEN t$ = CHR$(10) ELSE t$ = ","

      parlen% = 3:den% = 1
      IF (par% = 1) THEN den% = 10
```

```
    IF (par% = 4) OR (par% = 5) THEN parlen% = 5
    cpos1% = INSTR(cpos% + 1, hskeep$, t$)
    REM PRINT cpos1%;" ";
    IF ((cpos1% - cpos%) = (parlen% + 1)) THEN
      FOR l% = 1 TO 4
        var(l%, par% + 15) = VAL(MID$(hskeep$, cpos% + 1, parlen%)) / den%
      NEXT
    ELSE
      FOR l% = 1 TO 4
        var(l%, par% + 15) = VAL(STRING$(parlen%, "9"))
      NEXT
      flag%(5) = flag%(5) + 2^(par% - 1)
    END IF
    cpos% = cpos1%
END SUB

Handler:
  Number = ERR
  IF (Number = 53) THEN
    IF (RIGHT$(f$, 1) <> "d") THEN
      PRINT "Hour "; hh$; " not found"
      PRINT#3,"Hour ";hh$; " not found"
      CLOSE #1
      RESUME Missing
    ELSE
      PRINT "Cannot open output file"
      CLOSE #2
      END
    END IF
  ELSE
    PRINT "Error "; Number
    INPUT "Press Enter to exit";r$
    CLOSE#1
    CLOSE#2
    CLOSE#3
    END
  END IF
```

A.4 4MTO1M.C

```
/* Source Code of 4MTO1M.C
for converting a 4 Mbyte flashcard file c:\thincard\test
(produced by reading card on thincard drive with batch file T.BAT)
to 4 separate 1 Mbyte files in current directory */

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

main()
{

FILE * fin;
FILE * fout;

int c;

long n;

if ( (fin = fopen("c:\\thincard\\test", "rb")) != NULL)
   {
```

```
if ( (fout = fopen("test.1mg", "wb+")) != NULL)
        {
        printf("Converting 1st Mbyte\n");
        for (n = 0; n < 1048576L; n++)            /* copy 1st meg to testfile */
            {
            c = fgetc(fin);
                {
                fputc(c, fout);
                }
            }
        printf("Transfer OK\n");
        }
else
        {
        printf("Failed to open O/P File\n");
        }
fclose(fout);
printf("Press a key to continue\n");
getch();
/* copy 2nd meg to test.2mg */

if ( (fout = fopen("test.2mg", "wb+")) != NULL)
        {
        printf("Converting 2nd Mbyte\n");
        for (n = 0; n < 1048576L; n++)
            {
            c = fgetc(fin);
            fputc(c, fout);
            }
        printf("Transfer OK\n");
        }
else
        {
        printf("Failed to open O/P File\n");
        }
fclose(fout);
printf("Press a key to continue\n");
getch();

/* copy 3rd meg to test.3mg */

if ( (fout = fopen("test.3mg", "wb+")) != NULL)
        {
        printf("Converting 3rd Mbyte\n");
        for (n = 0; n < 1048576L; n++)
            {
            c = fgetc(fin);
            fputc(c, fout);
            }
        printf("Transfer OK\n");
        }
else
        {
        printf("Failed to open O/P File\n");
        }
fclose(fout);
printf("Press a key to continue\n");
getch();

/* copy 4th meg to test.4mg */

if ( (fout = fopen("test.4mg", "wb+")) != NULL)
        {
```

```
                printf("Converting 4th Mbyte\n");
                for (n = 0; n < 1048576L; n++)
                    {
                    c = fgetc(fin);
                    fputc(c, fout);
                    }
                printf("Transfer OK\n");
                }
        else
                {
                printf("Failed to open O/P File\n");
                }
        fclose(fout);


        }
    else
        {
        printf("Failed to open I/P File\n");
        }

return 0;
}
```

A.5 FORMDECODE

```
REM QuickBasic Program FORMDECODE
REM for decoding the ARGOS database binary parts
REM of a Formatter file
REM CHC 24-01-94

DIM W(32), qtrs%(5),PSD(5),MWS(5),FITA(5),VM(5)
OPEN "Wooig8-CHC:CHC-mac:Swales:FORMSWAL.M2" AS #1 LEN = 416
OPEN "Wooig8-CHC:CHC-mac:Swales:decoded2" FOR OUTPUT AS #2
FIELD #1, 128 AS arg$, 288 AS met$
GET#1,1
db%=1
WHILE NOT EOF(1)
  PRINT db%
  GET#1,db%:db%=db%+1

REM next bit taken from argsonfile
  FOR frame% = 1 TO 4
    FOR m% = 1 TO 32
      W(m%) = ASC(MID$(arg$,32*(frame%-1) + m%,1))
    NEXT m%

  REM w(1) to w(32) now contain the 32 bytes of the frame

  REM now decode the frame into the 5 records of
  REM starttime (quarters), PSD, MWS, Fit_A, V_MWS, N2
  REM inserting 999 type values where parity errors are detected

    FOR REC% = 1 TO 5
      n% = 6 * REC%
      word& = W(n% - 5)    'word& is the quarter-hours since midnight
      bits% = 8: GOSUB Paritycheck
      IF word& < 99 THEN
        qtrs%(REC%) = word&
      ELSE
        qtrs%(REC%) = 99
```

```
        END IF

        word& = W(n% - 4)*4 + (W(n% - 3) AND 192)/64   'PSD
        bits% = 10: GOSUB Paritycheck
        IF word& < 9999 THEN
          PSD(REC%) = .01*word& - 6
        ELSE
          PSD(REC%) = -9.99
        END IF

        word& = (W(n% - 3) AND 63)*16 + (W(n% - 2) AND 240)/16   'MWS
        bits% = 10: GOSUB Paritycheck
        IF word& < 9999 THEN
          MWS(REC%) = .1*word&
        ELSE
          MWS(REC%) = 99.9
        END IF

        word& = (W(n% - 2)AND 15)*64 + (W(n% - 1)AND 252)/4   'FIT_A
        bits% = 10: GOSUB Paritycheck
        IF word& < 9999 THEN
          FITA(REC%) = .01*word& - 6
        ELSE
          FITA(REC%) = -9.99
        END IF

        word& = (W(n% - 1) AND 3)*256 + W(n%)   'V_MWS
        bits% = 10: GOSUB Paritycheck
        IF word& < 9999 THEN
          VM(REC%) = .02*(word& - 256)
        ELSE
          VM(REC%) = +9.99
        END IF

      NEXT REC%

      word& = W(31) * 256 + W(32)   '16 bit word for N2 values
      bits% = 16: GOSUB Paritycheck
      IF (word& < 99999&) THEN
        n% = 4096
        FOR REC% = 1 TO 5
          n2(REC%) = INT(word& / n%) AND 7
         n% = n%/8
        NEXT
      ELSE
        FOR REC% = 1 TO 5
          n2(REC%) = 9
       NEXT
      END IF
      FOR REC% = 1 TO 5
       PRINT
#2,qtrs%(REC%);",";PSD(REC%);",";MWS(REC%);",";FITA(REC%);",";VM(REC%);",";n2(REC%)
      NEXT

    NEXT frame%
    PRINT#2,met$

WEND
CLOSE #1

END

REM Subroutines
```

```
Paritycheck:
  REM checks for even parity
  p% = 0:b& = 1
  FOR bit% = 1 TO bits%
    IF (word& AND b&) THEN p% = p% XOR 1
    b& = b&*2
  NEXT

  IF p% = 0 THEN
    word& = word& AND (2^(bits% - 1) - 1)
  ELSEIF bits% = 8 THEN
    word& = 99
  ELSEIF bits% = 10 THEN
    word& = 9999
  END IF
  IF p% = 0 AND bits% = 16 THEN word& = 99999&
  REM at present error in n2 parity bit
RETURN
```

A.6 SONDIRN

```
REM QuickBasic Program SONDIRN.BAS
REM to calculate sonic wind direction from vectors
REM using tabular data file as input
REM 180 deg added to directions to match Young
REM chc 15/2/94

OPEN "Wooig8-CHC:CHC-mac:Swales:Formatter:1st deployment:1st deployment CG copy"
FOR INPUT AS #1
OPEN "Wooig8-CHC:CHC-mac:Swales:Formatter:1st deployment:1st deployment CG mod"
FOR OUTPUT AS #2
REM Day   PSD   MWS (m/s)  North WS (m/s)  East WS (m/s)   Vert WS (m/s)   Fit-A   AT1
    AT2
REM ST1   ST2   Young WS (m/s)   Young Direction   Battery Voltage   Heading
    Heading rms
REM       Tmet   Tson   AT1-AT2   ST1-ST2

l%=0: radtodeg = 180/3.14159
WHILE NOT EOF(1)
  INPUT#1,day$,psd$,mws$,north$,east$,vert$,fita$,at1$,at2$
  INPUT#1,st1$,st2$,yws$,yd$,bat$,hdg$,hrms$
  INPUT#1,tmet$,tson$,atd$,std$
  l% = l% + 1
  REM PRINT day$,psd$,mws$,east$, north$,vert$;" ";
  PRINT l%, day$

  e = VAL(east$): n = VAL(north$)
  REM PRINT e, n;":";

  IF (n <> 0) THEN
    th = radtodeg * ATN(e / n)
  ELSE
    IF (e > 0) THEN th = 270 ELSE th = 90
    PRINT#2, th
  END IF

  IF n > 0 THEN
      th = th + 180
      PRINT#2, th
  END IF
```

```
  IF e < 0 AND n < 0 THEN
    th = th
    PRINT#2, th
  END IF
  IF e > 0 AND n < 0 THEN
    th = 360 + th
    PRINT#2, th
  END IF
  REM PRINT th
  REM INPUT r$
WEND

CLOSE#1
CLOSE#2
```

APPENDIX B DATA FORMATS

Appendix B.1 Raw Data Files

The first 256 kbytes of FORMSWAL.1MG consist of consecutive directory entries; these are each 32 bytes in length, starting from location 0, with the following format:

vjjjhhmmbffllnnv2JJJv1v3v400000

where

v is a marker character

jjj is the Formatter clock Julian Day number (0 - 365)

hh is the Formatter clock hours (0 - 23)

mm is the Formatter clock minutes (0 - 59)

b is the FlashCard Start Block (0 - 63)

ff is the FlashCard Offset (0 - 65535)

ll is the record length (0 - 65535)

nn is the record number (0 - 65535)

v2 is the Sonic MWS expressed as div((int) (10 * sonic_mws + 0.5), 512).rem

JJJ is Sonic Processor message Julian Day number

v1 is the PSD expressed as div((int) (100 * (6 + psd) - 0.5), 512).rem

v3 is the Fit_a expressed as div((int) (100 * (6 + fit_a) - 0.5), 512).rem

v4 is the Vertical WS expressed as div((int) (50 * vert_mean + 256.5), 512).rem

00000 are five null characters.

The remaining 768 kbytes of FORMSWAL.1MG, starting at location 262144, consist of consecutive ARGOS and Meteosat messages (128 and 288 bytes, respectively)

An example of the ARGOS message contents is given below as 4 frames of 32 bytes in hex ASCII format.

A36CE0EF14E890E5409E98E71169C0C6ACE812E340AE4CE893628096A0E8F7FF

145A207DECED95DF809E2EF19661409E40E817E340AE66EF18E5A0BEAAE9C7FF

9965E0B67AEF9A69E0DEC8E41BE820B690ED9CEA80CF02E91D6740CEB2EAC7FF

1E6760BECAEF9F6B60DEA8EBA06B60DEE2E9216D60E712F222EAA0DEDEF1C7FF

The 32 bytes of a frame are in a highly packed format, which contains five quarter-hourly sets of values of PSD, MWS, Fit A and Vertical MWS; thus a satellite pass will normally acquire all four frames, i.e. twenty quarter-hourly sets, or 5 hours of data.

We shall denote the five quarter-hourly sets in a frame by suffices a - e

Each quarter-hourly set of data in a message contains:

Q — time of data acquisition period start in quarter-hours since midnight
(this has the range 0 to 95, which can be expressed as a 7 bit binary number
bits Q0 (lsb) to Q6 (msb), with an added even parity bit PQ)

PSD — log10(Power Spectral Density * f^5/3
(this is converted to a 9 bit binary value 000h to 1FFh,
bits PSD0 (lsb) to PSD8 (msb), with added parity bit PPSD,
by taking the remainder of [(100 * (6 + PSD) - 0.5) divided by 512].
This gives a nominal range of -6.00 to -0.89, for 000h to 1FFh,
although secondary ranges, such as -0.88 to +4.23, exist)

MWS — Mean Wind Speed
(this is converted to a 9 bit binary value 000h to 1FFh,
bits MWS0 (lsb) to MWS8 (msb), with added parity bit PMWS,
by taking the remainder of [(10 * MWS + 0.5) divided by 512]
This gives the nominal range of 0.0 to 51.1 m/s, for 000h to 1FFh,
although secondary ranges, such as 51.2 to 102.3, exist)

Fit_A — Coefficient 'a', for linear regression fit of PSD vs log10(frequency)
over the frequency range 2 - 4 Hz, PSD = a + b.log10(frequency)
(this is converted to a 9 bit binary value 000h to 1FFh,
bits Fit_A0 (lsb) to Fit_A8 (msb), with added parity bit PFit_A, as for PSD)

V_M — Vertical Mean Wind Speed
(this is converted to a 9 bit binary value 000h to 1FFh,
bits V_M0 (lsb) to V)M8 (msb), with added parity bit PV_M,
by taking the remainder of [(50 * V_M + 256.5) divided by 512]
This gives a nominal range of -5.12 to +5.11, for 000h to 1FFh,
although secondary ranges, such as +5.12 to +15.34, exist)

A quarter-hourly set of the above parameters amounts to 48 bits (6 bytes), so that 5 sets amount to 30 bytes, bytes 1 to 30, leaving 2 bytes in the frame free for additional data. These two bytes are used to convey the number of Multimet messages successfully used in the averaging process over the Sonic acquisition period, N2. However, since N2 has the range 0 to 10 (4 bit binary), we can not fit five x 4 bits into 2 bytes and we have to subtract 3 from the N2 values, setting negative values to 0. i.e. the resulting (N2-3) range is 0 to 7 (3 bit binary), leaving one bit free for an even parity check for the two bytes.

Bytes 31 and 32 are packed with the (N2-3) values as follows:

The data format is summarised in tabular form below, showing each byte as one line with the most significant bit to the left, from byte1 to byte32:

PQa	Q6a	Q5a	Q4a	Q3a	Q2a	Q1a	Q0a
PPSDa	PSD8a	PSD7a	PSD6a	PSD5a	PSD4a	PSD3a	PSD2a
PSD1a	PSD0a	PMWSa	MWS8a	MWS7a	MWS6a	MWS5a	MWS4a
MWS3a	MWS2a	MWS1a	MWS0a	PFit Aa	Fit A8a	Fit A7a	Fit A6a
Fit A5a	Fit A4a	Fit A3a	Fit A2a	Fit A1a	Fit A0a	PV Ma	V M8a
V M7a	V M6a	V M5a	V M4a	V M3a	V M2a	V M1a	V M0a
PQb	Q6b	Q5b	Q4b	Q3b	Q2b	Q1b	Q0b
PPSDb	PSD8b	PSD7b	PSD6b	PSD5b	PSD4b	PSD3b	PSD2b
PSD1b	PSD0b	PMWSb	MWS8b	MWS7b	MWS6b	MWS5b	MWS4b
MWS3b	MWS2b	MWS1b	MWS0b	PFit Ab	Fit A8b	Fit A7b	Fit A6b
Fit A5b	Fit A4b	Fit A3b	Fit A2b	Fit A1b	Fit A0b	PV Mb	V M8b
V M7b	V M6b	V M5b	V M4b	V M3b	V M2b	V M1b	V M0b
PQc	Q6c	Q5c	Q4c	Q3c	Q2c	Q1c	Q0c
PPSDc	PSD8c	PSD7c	PSD6c	PSD5c	PSD4c	PSD3c	PSD2c
PSD1c	PSD0c	PMWSc	MWS8c	MWS7c	MWS6c	MWS5c	MWS4c
MWS3c	MWS2c	MWS1c	MWS0c	PFit Ac	Fit A8c	Fit A7c	Fit A6c
Fit A5c	Fit A4c	Fit A3c	Fit A2c	Fit A1c	Fit A0c	PV Mc	V M8c
V M7c	V M6c	V M5c	V M4c	V M3c	V M2c	V M1c	V M0c
PQd	Q6d	Q5d	Q4d	Q3d	Q2d	Q1d	Q0d
PPSDd	PSD8d	PSD7d	PSD6d	PSD5d	PSD4d	PSD3d	PSD2d
PSD1d	PSD0d	PMWSd	MWS8d	MWS7d	MWS6d	MWS5d	MWS4d
MWS3d	MWS2d	MWS1d	MWS0d	PFit Ad	Fit A8d	Fit A7d	Fit A6d
Fit A5d	Fit A4d	Fit A3d	Fit A2d	Fit A1d	Fit A0d	PV Md	V M8d
V M7d	V M6d	V M5d	V M4d	V M3d	V M2d	V M1d	V M0d
PQe	Q6e	Q5e	Q4e	Q3e	Q2e	Q1e	Q0e
PPSDe	PSD8e	PSD7e	PSD6e	PSD5e	PSD4e	PSD3e	PSD2e
PSD1e	PSD0e	PMWSe	MWS8e	MWS7e	MWS6e	MWS5e	MWS4e
MWS3e	MWS2e	MWS1e	MWS0e	PFit Ae	Fit A8e	Fit A7e	Fit A6e
Fit A5e	Fit A4e	Fit A3e	Fit A2e	Fit A1e	Fit A0e	PV Me	V M8e
V M7e	V M6e	V M5e	V M4e	V M3e	V M2e	V M1e	V M0e
PN2a-e	N22a	N21a	N20a	N22b	N21b	N20b	N22c
N21c	N20c	N22d	N21d	N20d	N22e	N21e	N20e

Table B.1 Bit Map of 32 Byte ARGOS Frame

An example of the Meteosat message contents (288 bytes) is given below:

B01<CR><LF>

51005<CR><LF>

256<CR><LF>

32,-170,013,-001,+001,-005,-159,+126,+125,+117,+120,009,010<CR><LF>

33,-162,014,+002,+005,-003,-147,+132,+128,+118,+121,010,009<CR><LF>

34,-173,013,+000,+001,-003,-160,+136,+134,+119,+122,010,010<CR><LF>

35,-164,014,-005,-003,-005,-146,+137,+136,+120,+123,011,011<CR><LF>

231,122,000,-0000,+0001<CR><LF><CR><LF>

The format of this message is quite simple; the first three lines are, respectively, the buoy ID, the niminal latitude (51° North) and longditude (005° West) and the Julian Day number (256).

The next four lines include a combination of Sonic and Multimet data in the format:

QQ,+PSD,MWS,N_M,E_M,V_M,FtA,AT1,AT2,ST1,ST2,YW1,YD1

where QQ = Quarter-hours since midnight (range 00 - 96)

+PSD = 100 * log10(Power Spectral Density * f^5/3

MWS = 10 * (Mean Wind Speed in m/s)

N_M = 10 * (North Mean component of Wind Speed in m/s)

E_M = 10 * (East Mean component of Wind Speed in m/s)

V_M = 10 * (Vertical Mean component of Wind Speed in m/s)

FtA = 100 * Coefficient 'a', for linear regression fit of PSD vs log10(frequency)

(over the frequency range 2 - 4 Hz, PSD = a + b.log10(frequency))

AT1 = 10 * (Mean Air Temperature from sensor 1 in ^{o}C)

AT2 = 10 * (Mean Air Temperature from sensor 2 in ^{o}C)

ST1 = 10 * (Mean Sea Temperature from sensor 1 in ^{o}C)

ST1 = 10 * (Mean Sea Temperature from sensor 2 in ^{o}C)

YW1 = 10 * (Mean Young AQ1 Wind Speed in m/s)

YD1 = Mean Young AQ1 Wind Direction in degrees.

The final line includes housekeeping data in the format:

BBB,HHH,HSD,+TMET,+TSON

where

BBB = 10 * Mean Battery Voltageon the 24V bus

HHH = Mean Buoy Heading in degrees magnetic

HSD = Standard Deviation of Heading in degrees

+TMET = Time difference between Multimet and Formatter Real Time Clocks

(+ve for Multimet clock ahead of Formatter clock)

+TSON = Time difference between Sonic and Formatter Real Time Clocks

(+ve for Sonic clock ahead of Formatter clock)