ppodd Documentation

Axel Wellpott

Oct 03, 2018

Contents

Contents:

CHAPTER 1

PPODD: Post Processing Of Decades Data

1.1 Intro

The core data from the FAAM BAe146, and previously the C-130 have always been calibrated post flight with the TARDIS (**T**ranscription of **A**ircraft **R**aw **D**ata **I**nto **S**cientific units) routines. These are a suite of FORTRAN routines written in a modular way so as each instrument or small group of similar instruments are calibrated with a particular module. Raw data were all recorded in one standard raw data file, and the data were processed one second at a time, and written to an output file of an in house binary format (MRF2 or MRF5). The modules had to be run in a particular order so within the main CALIBRATE.FOR routine an array was set up defining inputs and outputs of each module from where an order could be calculated.

For FAAM it was decided to output in the more standard NetCDF so routines were written to convert the format, but leaving the basic output as MRF5. There has also been a need to include data not recorded by the standard DRS (data recording system), and hence in a different raw format. PSAP photometer, Buck hygrometer and notably the GIN (combined GPS and inertial navigation system) had to be squeezed in after the main processing. The GIN in particular had to be merged into the output dataset, and then parts of the processing re-run to calculate winds.

The interactive side of the processing was abandoned with the move to processing on Linux rather than the original VMS. Some data still couldn't be calculated one second at a time, notably the Total Water Content, which needs to be fitted against another hygrometer for all or part of the flight. These extra routines were written in PVwave/IDL and run after the rest of the code.

1.2 Motive

The main reason for changing the code is the arrival of a new data recording system, DECADES so there will be no data recorded in the original raw data format. There are other problems alluded to already that could be addressed in the upgrade:

- The use of IDL, which often presents licensing issues, and is being moved away from by the Met Office in favour of Python.
- Old FORTRAN code, not well understood by the FAAM team so difficult to maintain adequate level of support.

- Reliance on old in house data formats, which were designed for a different operating system.
- There is a mix of input data formats, so having the main processing relying on just one format means all the other data have to be treated as bolt ons.

1.3 Initial Plan

The new code will be chiefly Python/numpy. This is becoming the standard scientific analysis tool in the Met Office and is widely used in academia. It also has the advantage that we can still use original FORTRAN modules via f2py so that the transition can be more gradual. Modules can be rewritten as a background task or as and when they are needed.

The modular approach used previously will be kept, but with a more flexible internal representation of the data, and a wider variety of routines for reading data in. The modules will hopefully be more self describing, so that they can define their own inputs and outputs etc.

Internal data will be in numpy arrays, with associated timestamps. Matching these timestamps and extracting the relevant data to run through the various modules will be the basis of the new suite.

Modules will be written to read in both new, and old format data for comparison purposes, and a limited level of backwards compatibility. However not every older data format has been ported, but should the need arise new modules can be added.

1.4 Implementation

Now that the processing is in a usable state it is worth explaining the basic structure.

The basic unit is the ppodd.core.decades_dataset, which is a collection of parameters and a dictionary of processing modules. The processing modules are initialized from python files in ppodd.pod, which holds a dictionary of classes (ppodd.pod.modules) which are only instantiated when a dataset is created.

There are several types of parameters. The most important are the data parameters which are time stamped arrays with meta data. There are also constant parameters for storing flight calibration data, file parameters for information on input files and attribute parameters, that will become global attributes in the NetCDF.

There was some effort put in to creating a class of timed_data arrays. It is possible that this will be replaced in the future with a variation of pandas TimeSeries or DataFrame objects, but there is no pressing need for this now.

Every module will define a list of input_names. If all the input_names are in the data-set the module can be run. The run method extracts all the named inputs from the data-set, and then runs the process method of the module. There are two alternative ways of running the processing:

- Process all modules that it is able to with available inputs.
- Run the minimum modules needed to produce a particular set of output data parameters.

Most modules will define what outputs they produce at the initialization stage, but some notably those that read in data from files may not know what outputs they have until they are processed. Equally some modules may be able to run with smaller subset of inputs than is ideal. This is a slightly trickier problem, as we would not want the module to run until the best set of inputs are available if they are going to be, but we would still want it to run if the minimum are there. For this case it is usually best to only define the minimum inputs, and in some cases a second stage of processing would be written for when the extra inputs are available.

CHAPTER 2

PPODD Command

The command PPODD will start processing data, the arguments are a list of input files. The types of these files can be set explicitly with a colon after the file name, or they will be assumed by the program, based one file suffix etc. (see file_read.patterns and file_read.filetest).

There are various options to select which modules to use, which parameters to calculate, and the output file. If no output options are selected or if no input files the GUI will open.:

```
Usage: PPODD [options] [input_file1[:type1] input_file2[:type2] .. input_filen[:typen]


Options:

  -h, --help            show this help message and exit

  -w full, and or 1hz, --write=full, and or 1hz

                        Output file types - if not writing open GUI

  -o output, --output=output

                        Output file or folder - if not writing open GUI

  -p p1,p2...pn or all, --parameters=p1,p2...pn or all

                        Parameters to output

  -m mod1,mod2..modn, --modules=mod1,mod2..modn

                        Only run these modules

  -n mod1,mod2..modn, --not_modules=mod1,mod2..modn

                        Don't run these modules
```

(continues on next page)

(continued from previous page)

```
-l filename, --logfile=filename

                      Log file

--nctype=NCTYPE       NetCDF type of output
```

The output can be at 1Hz or full frequency, or both with the -w,–write option. The -o, –output option chooses the folder or the full file name of the output (if not chosen, the default folder is $NCDATA, and the default file name created from the flight number and date as prescribed by BADC).

2.1 PPODD GUI

The graphical interface for PPODD can be run via the command line (if no output options are selected) or from a Desktop shortcut. If the Desktop shortcut is used it is possible to drag an input file onto the shortcut to open the program with that input.

All the standard functions are available through the GUI, but it is a little less flexible in terms of output options. It will only try and write to $NCDATA with the normal BADC naming convention. You do however have the ability to change input files, archive data, and have a quick look/quality check the data.

CHAPTER 3

Parameter types

There are three types of parameters defined, all a parameter needs as a minimum is a name, some data, and a type. This is all the constants_parameter has. What it then defines is that getting items or slices from the parameter is equivalent to doing the same from its data attribute.

The file parameter is much the same, but defines the data as a set (of filenames).

The parameter or data parameter has all the common attributes of a NetCDF variable. The data however is of the type timed_data so that time matching can be done before calculations.

CHAPTER 4

Data types

There are three main data types defined. They may in the future, be rewritten using some parts of the standard pandas library, as some of the functionality is similar.

The basis of the timed data classes. It is an ndarray with some extra functionality.

- Can hold a frequency value, and has methods to calculate times at different frequencies
- Can match itself to other times
- Can convert between different time bases

Optionally initialized from a start and end time

```
>>> t=timestamp([1507216300,1507216301,1507216302,1507216303,1507216304])
>>> t
timestamp(['2017-10-05T15:11:40', '2017-10-05T15:11:41', '2017-10-05T15:11:42',
   '2017-10-05T15:11:43', '2017-10-05T15:11:44'], dtype='datetime64[s]')
>>> t2=timestamp((np.datetime64('2017-10-05T15:11:37'),np.datetime64('2017-10-
↪05T15:11:42')))
>>> t2
timestamp(['2017-10-05T15:11:37', '2017-10-05T15:11:38', '2017-10-05T15:11:39',
   '2017-10-05T15:11:40', '2017-10-05T15:11:41', '2017-10-05T15:11:42'], dtype=
↪'datetime64[s]')
>>> t.match(t2)
timestamp(['2017-10-05T15:11:40', '2017-10-05T15:11:41', '2017-10-05T15:11:42'],␣
↪dtype='datetime64[s]')
>>> t.ismatch(t.match(t2))
array([True,  True,  True, False, False], dtype=bool)
>>> t.at_frequency(2)
timestamp([['2017-10-05T15:11:40.000000000', '2017-10-05T15:11:40.500000000'],
   ['2017-10-05T15:11:41.000000000', '2017-10-05T15:11:41.500000000'],
   ['2017-10-05T15:11:42.000000000', '2017-10-05T15:11:42.500000000'],
   ['2017-10-05T15:11:43.000000000', '2017-10-05T15:11:43.500000000'],
   ['2017-10-05T15:11:44.000000000', '2017-10-05T15:11:44.500000000']], dtype=
↪'datetime64[ns]')
```

(continues on next page)

(continued from previous page)

```
timed_data
```

This attaches a timestamp to a data array so that similar time matching, frequency changes etc are possible.

Some useful additional methods are:

- ravel() which will flatten out a 2d array assuming the second dimension is frequency.
- timesort() which will sort the data into time order.
- asmasked() which will return a masked array
- get1Hz() gets the data average per second if 2d.

```
>>> d=timed_data([[10,11],[11,12],[12,13],[13,14]],timestamp((1507216300,1507216303)))
>>> d
timed_data([[10, 11],
    [11, 12],
    [12, 13],
    [13, 14]])
>>> d.frequency
2
>>> d.times
timestamp(['2017-10-05T15:11:40', '2017-10-05T15:11:41', '2017-10-05T15:11:42',
   '2017-10-05T15:11:43'], dtype='datetime64[s]')
>>> d.ravel()
timed_data([10, 11, 11, 12, 12, 13, 13, 14])
>>> d.ravel().times
timestamp(['2017-10-05T15:11:40.000000000', '2017-10-05T15:11:40.500000000',
   '2017-10-05T15:11:41.000000000', '2017-10-05T15:11:41.500000000',
   '2017-10-05T15:11:42.000000000', '2017-10-05T15:11:42.500000000',
   '2017-10-05T15:11:43.000000000', '2017-10-05T15:11:43.500000000'], dtype=
↪'datetime64[ns]')
>>> d=timed_data([1,4,5,6],timestamp([1507216300,1507216305,1507216301,1507216302]))
>>> d
timed_data([1, 4, 5, 6])
>>> d.times
timestamp(['2017-10-05T15:11:40', '2017-10-05T15:11:45', '2017-10-05T15:11:41',
   '2017-10-05T15:11:42'], dtype='datetime64[s]')
>>> d.timesort()
>>> d
timed_data([1, 5, 6, 4])
>>> d.times
timestamp(['2017-10-05T15:11:40', '2017-10-05T15:11:41', '2017-10-05T15:11:42',
   '2017-10-05T15:11:45'], dtype='datetime64[s]')

flagged_data
```

Much like timed_data, but also ties in a flag array. The flag array must be of the same size as the data array.

CHAPTER 5

Processing Modules

Like previous versions the processing is divided up in to a collection of modules. Each one will most likely deal with one instrument, or small group thereof. The modules can also perform more basic functions (eg. Unzipping files), what they have in common is identified inputs and outputs, and a run method to go from one to the other.

The standard modules are all stored in the package ppodd.pod. For the most common uses, these need not be changed, but there will be new instruments, changes to existing, and even bugs found in the current. These are the basis of any processing, but there is facility to amend these or add in modules from elsewhere. Within ppodd.pod each module with a name with p_* is assumed to contain a class. These are placed in the dictionary ppodd.pod.modules which the decades_dataset will load into it's own modules attribute.

There are three basic types of module.

- cal_base – this is the basis of all modules, defines a method for checking inputs, and running processes.
- file_read – a file reading class, will try and read in a list of files by calling its readfile method.
- fort_cal – for running the legacy FORTRAN routines.

These would then be sub-classed by something which actually does the work.

5.1 cal_base

cal_base is the root of all calibration routines, and stores information on the current running state, the dataset it is being applied to a version number and running history. It would need to be subclassed to do any calculations, either directly or via fort_cal or file_read.

The running state (.runstate) will start off as ready moving to running, and end as either success or fail depending on whether the inputs could be found.

If a new module needs to be tested it can be added to the modules that the processing uses, without being placed in the standard module folder.

An example of a test module sub-classing from cal_base:

```
from ppodd.core import *

class potential_temp(cal_base):
    """ Test module for calculating potential temperature """
    def __init__(self,dataset):
        self.input_names=['PS_RVSM','TAT_DI_R']
        self.outputs=[parameter('POT_TEMP',units='K',
          frequency=32,long_name='Potential Temperature')]
        self.version=1.00
        cal_base.__init__(self,dataset)

    def process(self):
        d=self.dataset
        match=d.matchtimes(['PS_RVSM','TAT_DI_R'])
        p1=d['PS_RVSM'].data.ismatch(match)
        t1=d['TAT_DI_R'].data.ismatch(match)
        pote=flagged_data(t1*(1000.0/p1)**(2.0/7.0),p1.times,p1.flag) #!Potential␣
↪temp (K)
        pote.flag[t1.flag>p1.flag]=t1.flag[t1.flag>p1.flag]
        self.outputs[0].data=pote

"""
An example of how to add a module for testing before being
added to the main processing suite in ppodd.pod
"""

import ppodd.pod

ppodd.pod.addmodule(potential_temp)
d=decades_dataset()
d.add_file('decades_data/b111.zip')
d.add_file('decades_data/flight-cst_faam_20131001_r0_b111.txt')
d.process()
```

We define a module a sub-class of cal_base which calculates potential temperature from pressure (PS_RVSM) and temperature (TAT_DI_R). The input names are defined, and the output parameter defined. In the process method the times of the 2 inputs are matched, the calculation performed and the flags set.

This new module can then be added by the line:

```
ppodd.pod.addmodule(potential_temp)
```

So that when a decades_dataset is created the new module is available, and will be calculated if the inputs are there.

5.2 fort_cal

A sub-classed fort_cal would require little more than a definition of input, and outputs, and the name of the FORTRAN routine (assumed to be the same as the class name). It should then put the data into the correct size and shape arrays, and via f2py and ppodd.pod.c_runmod will run the FORTRAN. Once the calculations have completed the data will be extracted from the array and put into the output parameters. Only the __init__ method need be overridden.

Fortran calling example:

```
from ppodd.core import *

```

(continues on next page)

(continued from previous page)

```
class ozone1(fort_cal):

    def __init__(self,dataset):
        self.input_names=['CALO3', 'CALO3P', 'CALO3T', 'CALO3F', \
          'CALO3MX', 'Horace_O3', 'Horace_RVAS']
        self.outputs=[parameter('O3_TECO',units='ppb',frequency=1,number=574, \
          long_name='Mole fraction of ozone in air from the TECO 49 instrument')]
        self.version=1.00
        fort_cal.__init__(self,dataset)
```

All the matching of timestamps etc should be handled by the base class, this only works for FORTRAN previously compiled and linked to the c_runmod.for using f2py. If files in the pod/fortran_modules folder are modified, then c_runmod.so will need to be remade (there is a makefile), however it is not anticipated that changes should be made to the FORTRAN, but to rather to either the python wrapper, or rewritten using python.

5.3 file_read

A sub-classed file_read needs to override the __init__, and implement the readfile method. readfile should take a file name as input, and may be called a number of times with different files. It should also be noted that the first input name should name the file type that this reads in, and this will be the name of the parameter which lists these files. A combination of the patterns tuple, and filetest method will be used to guess the file type when not specified, and fixfilename, will alter a full file path to something the readfile method understands. file_read will try to parse any file names for flight number and date, if they are not already in the data.

File reading example:

```
from ppodd.core import *
import numpy as np
import ppodd

class readincloud(file_read):
    """
    Routine for reading in some imaginary instrument data
    """
    def __init__(self,dataset):
        self.input_names = ['INCLOUD']
        self.patterns = ('incloud*.txt',)
        self.outputs  =[parameter('INCL_TEMP',units='K',frequency=1, \
          long_name='In cloud temperature from imaginary instrument')]
        self.data = None
        file_read.__init__(self,dataset)


    def readfile(self,filename):
        x = np.genfromtxt(filename,delimiter=',',names=['Time','temp','volts'],skip_
↪header=1)
        data = timed_data(x['temp'],x['Time'])
        if(self.outputs[0].data):
            self.outputs[0].data=np.append(self.outputs[0].data,data)
        else:
            self.outputs[0].data=data
        self.outputs[0].data.timesort()
```

The readfile method may be called more than once if there is a list of input files, and should deal with this appropriately – likely adding new data, and sorting if necessary. It defines patterns, which is a tuple of file search strings, to help other processes guess file types when not specified.

CHAPTER 6

Usage

6.1 Data Preparation

6.2 Post Processing

The routines can be run a number of different ways depending on need. Simple python scripts can run the processing if access to data within a program is needed. A command line, can run the processing and produce an output, or a gui can be run which enables a certain amount of interactive exploration of the data.

A simple programming example:

```
import ppodd.core

d=ppodd.core.decades_dataset()
d.add_file('decades_data/b111.zip')
d.add_file('decades_data/flight-cst_faam_20131001_r0_b111.txt')
d.process()
d.write_nc()
```

d would then hold all the processed data. It could be plotted using matplotlib or calculations performed. Each parameter accessed via:

```
d['parameter name']
```

there being various attributes. The ".data" attribute actually holding the values and can be viewed in several ways.:

```
d['JW_LWC_U'][:] is equivalent to d['JW_LWC_U'].data

d['JW_LWC_U'].times is equivalent to d['JW_LWC_U'].data.times
```

See the explanation of the timed_data array, and parameters.

The same could be run from a command line:

```
PPODD decades_data/flight-cst_faam_20131001_r0_b111.txt decades_data/b111.zip -w full
```

To wrap around this structure there is also a graphical user interface. This should make the process of calibrating and checking core data relatively painless.

To achieve the same result:

```
PPODD (to open the GUI - or use the desktop shortcut)
```

From menu Files, Files then Add File (to add the relevant file). Click Process, followed by File, Write_nc.

CHAPTER 7

Processing python Modules

7.1 rio_geneas.py

class ppodd.pod.p_rio_geneas.**rio_geneas**(*dataset*)

FORTRAN routine C_GENEAS DEC$ IDENT 'V1.02'

ROUTINE C_GENEAS SUBROUTINE FORTVAX

PURPOSE Derivation of Dew point

DESCRIPTION Calculation of Dew Point in K from General Eastern Hygrometer

529- Dew Point [K]

The General Eastern Hygrometer (Parameter 58) is recorded in binary with a range of 0 to 4095 DRS bits. A control signal (Parameter 59) is also recorded which gives an indication of the amount of heating or cooling of the mirror. The instrument should be in control if the signal is between certain limits. Outside these limits it still produces a dew point reading,though of doubtful accuracy, and derived data is flagged - FLAG = 2.

VERSION 1.00 240190 J HARMER

1.01 17-01-96 D Lauchlan

ARGUMENTS

Constants GEMAX Maximum control condition signal limit RCONST(1)

GEMIN Minimum control condition signal limit RCONST(2)

CALGE(1) GE Dew point calib. constant x0 RCONST(3)

CALGE(2) GE Dew point calib. constant x1 RCONST(4)

CALGE(3) GE Dew point calib. constant x2 RCONST(5)

Inputs GENERAL EASTERN 1011 DEW POINT [drs units] Para 58

GENERAL EASTERN CONTROL SIGNAL [drs units] Para 59

Outputs DEW POINT [K] Para 529

SUBPROGRAMS ITSTFLG Examines bits 16,17 for flags

ISETFLG Sets flag bits 16,17 = 0 –> 3

S_QCPT Performs range and rate of change check

REFERENCES Code adapted from MRF1/MRF2

CHANGES v1.01 17-01-96 D Lauchlan; Unused variables removed

V1.02 27/09/02 W.D.N.JACKSON Changed to include handling of 16 bit data from the new DRS.

7.2 rio_buck_cr2

class ppodd.pod.p_rio_buck.**rio_buck_cr2**(*dataset*)

Routine to process data from the BUCK CR2 Hygrometer.

7.3 rio_co_mixingratio.py

class ppodd.pod.p_rio_comr.**rio_co_mixingratio**(*dataset*)

Routine to calculate the Carbon Monoxide concentration from the AL52002 Instrument.

The routine works with the data from the TCP packages that are stored on Fish and Septic. Flagging is done using the cal_status flag (AL52CO_cal_status) and the pressure measurement in the calibration chamber of the instrument (AL52CO_calpress).

7.4 rio_cpc.py

class ppodd.pod.p_rio_cpc.**rio_cpc**(*dataset*)

Routine for processing the data from the CPC (Condensation Particle Counter) instrument TSI 3786.

FLAGGING

0. Data OK
1. Saturator temperature more than 6 degrees C, Growth or Optics Temp more/less than 10% from prescribed value
2. Aerosol (Sample) flow more/less than 10% from prescribed value
3. Sheath flow more/less than 10% from prescribed value

OUTPUT CPC_CNTS

7.5 rio_geneas.py

class ppodd.pod.p_rio_geneas.**rio_geneas**(*dataset*)

FORTRAN routine C_GENEAS DEC$ IDENT 'V1.02'

ROUTINE C_GENEAS SUBROUTINE FORTVAX

PURPOSE Derivation of Dew point

DESCRIPTION Calculation of Dew Point in K from General Eastern Hygrometer

529- Dew Point [K]

The General Eastern Hygrometer (Parameter 58) is recorded in binary with a range of 0 to 4095 DRS bits. A control signal (Parameter 59) is also recorded which gives an indication of the amount of heating or cooling of the mirror. The instrument should be in control if the signal is between certain limits. Outside these limits it still produces a dew point reading,though of doubtful accuracy, and derived data is flagged - FLAG = 2.

VERSION 1.00 240190 J HARMER

1.01 17-01-96 D Lauchlan

ARGUMENTS

Constants GEMAX Maximum control condition signal limit RCONST(1)

GEMIN Minimum control condition signal limit RCONST(2)

CALGE(1) GE Dew point calib. constant x0 RCONST(3)

CALGE(2) GE Dew point calib. constant x1 RCONST(4)

CALGE(3) GE Dew point calib. constant x2 RCONST(5)

Inputs GENERAL EASTERN 1011 DEW POINT [drs units] Para 58

GENERAL EASTERN CONTROL SIGNAL [drs units] Para 59

Outputs DEW POINT [K] Para 529

SUBPROGRAMS ITSTFLG Examines bits 16,17 for flags

ISETFLG Sets flag bits 16,17 = 0 –> 3

S_QCPT Performs range and rate of change check

REFERENCES Code adapted from MRF1/MRF2

CHANGES v1.01 17-01-96 D Lauchlan; Unused variables removed

V1.02 27/09/02 W.D.N.JACKSON Changed to include handling of 16 bit data from the new DRS.

7.6 p_rio_heiman.py

class ppodd.pod.p_rio_heiman.**rio_heiman**(*dataset*)

FORTRAN routine C_HEIMAN

ROUTINE C_HEIMAN SUBROUTINE FORTVAX

PURPOSE To derive uncorrected Heimann temperatures

DESCRIPTION Converts rawdata input from the Heimann radiometer and black body source into uncorrected surface tempratures, parameter 537.

The Heimmann is recorded by para 141, the blackbody reference temperature by para 142, and bit 0 of the signal register (para 27) indicates whether the Heimann is set to calibrate.

ARGUMENTS IRAW input raw data IFRQ raw data frequencies RCONST flight constants corresponding to PRTCCAL and HEIMCAL RDER output data

SUBPROGRAMS

REFERENCES

VERSION 1.00 D.R.Lauchlan

CHANGES

DESCRIPTION Converts the two input parameters 141 (raw Heimann) and 142 (black body reference temperature) into one, the uncorrected HEIMAN temp (para 537).

The Heimann Radiometer data is converted using a quadratic fit : Surface temp = RCONST(4) + RCONST(5)*x + RCONST(6)*x**2 RCONST(4 to 6) correspond to the constants with the keyword HEIMCAL in the flight constants file.

The black body signal (para 142) is converted using a quadratic fit:

```
BB = a + b*x + c*x**2
```

where constants a, b and c correspond to RCONST(1 to 3) from the keyword PRTCCAL in the flight constant file.

Signal Register (para 27) bit 0 indicates the position of the black body; 0 = b/b out of FoV, 1 = b/b in FoV.

If signal register bit 0 is set to 1 and black body reference temprature has been steady for the previous 3 seconds (mean of each second differs by no more than 0.1K), the b/b reference temperature is output. Otherwise the HEIMAN temprature is output. An offset is assigned accordingly

		Para 27 bit 0	
233.26 to 313.06	Heimann-	0	(o/s = 0)
1233.26 to 1313.06	Ref/BB -	0	(o/s = 1000)
2233.26 to 2313.06	Heimann-	1	(o/s = 2000)
3233.26 to 3313.06	Ref/BB -	1	(o/s = 2000 + 1000)

(NOTE: an offset of 1000 is never assigned under this scheme)

Heimann data is output for the time that the reference temperature is output. This is done in 4 second bursts imediately after the reference sequence and overwrites the ramping sections. Non reference or dwelling calibration temeratures are flagged as 2. Dwell Heimann data is output for the corresponding calibration reference period after the instrument has switched back to measure, ie para 27 bit 0 becomes 0.

Bits 16 and 17 of the output temperature RDER(x,537) are used to flag certain data conditions as follows :

```
Bits 16 and 17 = 00 - Good data, MEASURE/HEIMANN
        = 01 - Good data, but Heimann on CALIBRATE
                and outputing DWELL temp
                or looking at the Black Body temps
                or BB moved out of field of view
                and data are last Heimann dwell
                data.
        = 02 - Suspect or absent signal register
                data, non-reference calibration
                temperatures and non-dwelling
                calibration temperatures.
```

= 03 - Absent data, passed through from
IRAW(x,141)

ARGUMENTS

IRAW(f,141) Raw Heimann data
IRAW(f,27) Raw signal register data
IRAW(f,142) Raw black body data
IFRQ(141) Recorded frequency of Heimann Radiometer
IFRQ(27) Recorded frequency of signal register
IFRQ(142) Recorded frequency of black body
RCONST(1-6) Constants for quadratic fit
RDER(f,537) Uncorrected Heimann temps (deg K)

SUBPROGRAMS

ITSTFLG - Returns the value of bits 16 & 17 - SCILIB
ISETFLG - Sets the value of bits 16 & 17 - SCILIB
IBITS - Extracts selected bits from input - FORTRAN
BTEST - Tests value of selected single bit - FORTRAN
C_HEIMAN_LTST_BB - Checks array elemenets are within +/- 86 - LOCAL

REFERENCES

VERSION 1.00 09-11-94 D.R.Lauchlan Based on C_BARNES V2.00 by D.P. Briggs

CHANGES V1.01 10/02/99 W.D.N.JACKSON Bug in flag checking of raw data fixed.

V1.02 27/09/02 W.D.N.JACKSON Changed to include handling of 16 bit data from the new DRS. Also now expects calibrator temp cal to be in deg C.

V1.03 11/11/04 J.P. TAYLOR Changed to account for 16bit representation of DRS parameters. Allowable range of BB ref means changed from +/- 6 to +/- 86 this is equivalent to 0.1K with the new DRS 16bit data stream.

7.7 rio_lwc.py

class `ppodd.pod.p_rio_lwc.`**`rio_lwc`**(*dataset*)
FORTRAN routine C_LWC

ROUTINE C_LWC SUBROUTINE FORT VAX [C_LWC.FOR]

PURPOSE To calibrate DRS parm 42 to tardis parm 535 (LWC)

DESCRIPTION The Liquid Water Content (LWC) is a four hertz parameter. It requires the True Air Speed (Parm 517), True De_iced Temperature (parm 520) and Static Pressure (parm 576). All these derived parameters (517, 520, 576) are at 32 Hertz. So for each quarter point of the LWC requires a sample of eight of the derived paramters to be averaged. This is done using only good data points. If there are not eight samples but more than one then the flag for the derived LWC is set to 1. If the frequency of the DRS parm (42) is not equal to 4 then no values are calculated and all four points of the LWC are set to -9999.0, with a flag of 3. If a point cannot be calculated then the value of it is set to -9999.0 with a flag value of 3. If the instrument is saturated then the flag value is 1. If the derived value for the LWC falls out of the bounds of -10 to 10 then the flag is set to 2.

VERSION 1.02 17-01-96 D Lauchlan

ARGUMENTS IRAW(64,512) I*4 IN Raw data for the parameters

FRQ(512) I*4 IN Frequencies of the data

RCONST(64) R*4 IN Constants required by routine,(1-28)

RDER(64,1024)R*4 OUT Tardis parameters

SUBPROGRAMS ISETFLG (linked automatically)

REFERENCES MRF2 Specification for Total Water Hygrometer 4 Dec 1989 Ouldridge Feb 1982 Johnson 1979

CHANGES 110190 Documentational changes only M.Glover

v 1.02 17-01-96 D Lauchlan Unused parameters removed

V1.03 27/09/02 W.D.N.JACKSON Changed to include handling of 16 bit data from the new DRS.

7.8 rio_nephelometer.py

`class ppodd.pod.p_rio_nephelometer.`**`rio_nephelometer`**`(`*dataset*`)`

Module for the TSI 3563 Nephelometer

No post processing is done to the data that are recorded and streamed by the RIO modules. However, the units of the scatter values are converted from megameters-1 to meters-1 and the flags are set using the AERACK_neph_status parameter.

The status flag definition is described in the TSI 3563 manual on page 6-22.

Flagging is done for the:

1. scattering values
2. the relative
3. temperature
4. pressure measurements.

Flagging RF returns a four-character hexadecimal value representing the state of the Nephelometer. The values for the sixteen flags are as follows:

0. Data OK
1. Not used
2. Not used
3. Lamp not within 10% of SP setting * Valve fault * Chopper fault * Shutter fault * Heater active but not stabilized * Pressure out of range * Sample Temp out of range * Inlet temp out of range * RH out of range

7.9 rio_ozone.py

`class ppodd.pod.p_rio_ozone.`**`rio_ozone`**`(`*dataset*`)`

DESCRIPTION Calibrated TEI Ozone instrument data The flagging of the data uses mainly the flow measurements in the two chambers | TEIOZO_FlowA | TEIOZO_FlowB Data points while on the ground (WOW_IND = 1) are also flagged as '3' and so are all data points for the first 20 seconds after take-off

The TECO_49 also sends its own flag in the data stream (TEIOZO_flag) which is used for flagging

FLAGGING Flag values are set using flow_a, flow_b, weight on wheels index and the instruments own flag value | flag=flag!='1c100000' | flag[conc < -10]=2 | flag[flow_a < flow_threshold]=3 | flag[flow_b < flow_threshold]=3 | flag[wow_ind != 0]=3

7.10 rio_press.py

`class ppodd.pod.p_rio_press.`**`rio_press`**`(`*dataset*`)`

FORTRAN routine C_PRESS1

ROUTINE C_PRESS1 SUBROUTINE FORTVAX

PURPOSE Calibrates the cabin pressure sensor and the S9 static port.

DESCRIPTION Apply calibration the combined transducer and DRS coefficients to DRS parameters 14 and 221 to obtain derived parameters 579 and 778. Invalid data is flagged with 3, data outside limits is flagged with 2.

METHOD For each DRS parameter to be calibrated

1. If data is FFFF or FFFE then flag 3
2. Apply the calibration constants
3. Check the results for being within acceptable values.
4. Set data flag bits (16+17)

FLAGGING

0 = Good data
1 = Data of lower quality
2 = Probably faulty, exceed lims
3 = Data absent or invalid.

Flagging - If a value can't be computed, due to missing data missing constants, divide be zeroes, etc, a value of 0 is used, flagged with a three. If a value is outside its limits for range or rate of change, it is flagged with a two. If there are no problems with the data it is flagged with 0.

VERSION 1.00 23/07/03 W.D.N.JACKSON

ARGUMENTS

Inputs DRS para 14 CABP 1 Hz Cabin pressure

DRS para 221 S9SP 32 Hz S9 static pressure

Constants RCONST(1 to 3) Para 14 cal constants X0 to X2

RCONST(4 to 6) Para 221 cal constants X0 to X2

Outputs Derived para 579 CABP mb 1 Hz Cabin pressure

Derived para 778 S9SP mb 32 Hz S9 static pressure

SUBPROGRAMS ISETFLG

REFERENCES

CHANGES V1.00 23/07/03 WDNJ Original version Note that V1.00 has no application of S9 position errors.

7.11 rio_psap.py

class `ppodd.pod.p_rio_psap.`**`rio_psap`**(*dataset*)

PSAP processing module

INPUTS

AERACK_psap_flow
AERACK_psap_lin
AERACK_psap_log
AERACK_psap_transmission

OUTPUTS

PSAP_LIN - Uncorrected absorption coefficient at 565nm, linear, from PSAP
PSAP_LOG - Uncorrected absorption coefficient at 565nm, log, from PSAP
PSAP_FLO - PSAP Flow
PSAP_TRA - SAP Transmittance

FLAGGING

using flow and transmission thresholds
flag[(psap_transmission<0.5) | (psap_transmission>1.05)]=1
ix=np.where(psap_flow < 1.0)[0]
#add two second buffer to the index
ix=np.unique(np.array([list(ix+i) for i in range(-2,3)]))
ix=ix[(ix >= 0) & (ix < n-1)]
flag[ix]=2
flag[((psap_transmission<0.5) | (psap_transmission>1.05)) & (psap_flow<1.0)]=3

7.12 rio_radal.py

class `ppodd.pod.p_rio_radal.`**`rio_radal`**(*dataset*)

FORTRAN routine C_RADAL1

ROUTINE C_RADAL1 SUBROUTINE FORTVAX

PURPOSE To subroutine to calculate the aircraft altitude from the radar altimeter.

DESCRIPTION The raw radar altimeter data is provided as a 16 bit signed number from the ARINC 429 data bus, with a least bit resolution of 0.25 ft.

FLAGGING The derived data is quality controlled to ensure that

data outside the range 0 to 8191.75 ft are flagged 3
more than two values the same are flagged 3
more than 1000' change between values is flagged 3

TO COMPILE $FORT C_RADAL1

VERSION 1.00 02/10/02 W.D.N.JACKSON

ARGUMENTS

IRAW(X,37) - where x=1 or 2, on entry this contains the raw radar height.
IFRQ(37) - on entry contains 2, the frequency of the raw radar height.
RDER(X,575) - where x= 1 or 2, on exit contains the derived radar height in meters.

CHANGES V1.01 WDNJ 05/11/04 Flagging criteria improved

7.13 rio_rvsm.py

class `ppodd.pod.p_rio_rvsm.`**`rio_rvsm`**(*dataset*)

ROUTINE C_RVSM SUBROUTINE FORTVAX

PURPOSE Computes static pressure, pitot-static pressure, and pressure height from the 146 RVSM altitude and airspeed data.

DESCRIPTION RVSM altitude is available in ARINC-429 message 203 and is recorded as by the DRS as a 16 bit signed word, with the LSB representing 4 feet.

RVSM computed airspeed is available in ARINC-429 message 206 and is recorded by the DRS as a 16 bit signed word, with the LSB representing 1/32 kt, but always zero.

These values should be within the system accuracy specification and do not require calibration.

Note that altitude is updated by the RVSM at about 20 Hz and airspeed is updated at about 10 Hz. Both signals are sampled by the DRS at 32 Hz so there will be multiple values and aliasing effects.

METHOD For each DRS parameter to be calibrated:

1. If data is FFFF or FFFE or out of range then flag 3
2. Decode the altitude and use the tables in NASA TN D-822 to back compute the static pressure.
3. Decode the airspeed and use fundamental equations to compute pitot-static pressure.
4. Check the results for being within acceptable values.
5. Set data flag bits (16+17)

   0: Good data
   1: Data of lower quality
   2: Probably faulty, exceed lims
   3: Data absent or invalid.

FLAGGING If a value can't be computed, due to missing data missing constants, divide be zeroes, etc, a value of 0 is used, flagged with a three. If a value is outside its limits for range, it is flagged with a two. If there are no problems with the data it is flagged with 0. Any flags on input data are propagated through subsequent calculations.

Note that routine does not currently apply position error corrections, nor interpolate missing data.

VERSION 1.00 23/07/03 W.D.N.JACKSON

ARGUMENTS

Inputs

DRS para 222 RVAL 32 Hz RVSM altitude
DRS para 223 RVAS 32 Hz RVSM computed airspeed

Outputs

Derived para 576 SPR mb 32 Hz Static pressure
Derived para 577 PSP mb 32 Hz Pitot-static pressure
Derived para 578 PHGT m 32 Hz Pressure height

Flags

Missing/corrupt data output as 0 flagged 3.
Out of range derived data flagged 2.

SUBPROGRAMS S_PSP, ALT2PRESS, ISETFLG

REFERENCES NASA Technical Note D-822, Aug 1961, Tables of airspeed, altitude, and mach number.

Interface Control Document, Air Data Display Unit, ISS 1G-80130-22.

CHANGES V1.00 23/07/03 WDNJ Original version

V1.01 23/10/03 WDNJ Now replicates data when missing

V1.02 11/12/03 WDNJ Fixes bug if initial data missing

V1.03 11/03/04 DAT Flags data outside altitude range 3

V1.04 17/03/04 WDNJ Now handles negative heights correctly and uses more accurate flagging criteria

7.14 rio_so2_mixingratio.py

class `ppodd.pod.p_rio_so2.`**`rio_so2_mixingratio`**(*dataset*)
Routine to process the SO2 measurements from the SO2-TECO analyser.

7.15 rio_sols.py

class `ppodd.pod.p_rio_sols.`**`rio_sols`**(*dataset*)

ROUTINE C_SOLS SUBROUTINE FORTVAX

PURPOSE CALIBRATE PYRANOMETER & PYRGEOMETER RAW SIGNALS AND THERMISTORS.

DESCRIPTION Apply calibration coefficients to RAW parameters 81-89 and 91-99 to obtain uncorrected values of signal flux, zero offset signal (W/m-2) and thermistor output (deg K) for each of the upward-facing and downward-facing sets of: clear dome & red dome pyranometers and pyrgeometer.

METHOD For each RAW parameter to be calibrated, for the six instruments:

1. Check all its required constants are present (Flag <3) (if not, the calibration of that parameter will not proceed) [Also check that the normal configuration of instruments is to be used. Any changes are indicated by the presence of a large offset to the second calibration constant for

any instrument. If this is present the offset is decoded to generate a revised ICONF indicator for that instrument. See note below.]

2. Obtain the raw signal/zero value and float the result.
3. Calibrate by applying the appropriate instrument cal in RCALB (which is loaded from the RCONST arguments) to both raw signal flux and zero offset, which use the same coefficients The gains are in W/m-2 /DRS count. DRS counts are related to radiometer output Voltage. Note that the output Voltage from the instrument is the value after being amplified by the head amplifier.
4. Range check and Rate-of-change check: (S/R QCPT)

   the calibrated signal (Wm-2)
   Zero offset (DRS units)
   temperature (deg K)
5. Calibrate the thermistor input using two RCALB coefficients. Add 273.15 deg to thermistor results to express the instrument thermopile temperature in degrees Kelvin.
6. Check the result is within pre-defined limits

FLAGGING Set the calibrated values flag bits (16+17) as follows

0: Good data
1: Data of lower quality
2: Data probably faulty, exceeding limits
3: Data absent or known to be invalid.

VERSION 1.04 250692 A D HENNINGS

ARGUMENTS

RCONST(1) - REAL*4 IN Upper Clear dome Signal & Zero const.
* RCONST(2) - REAL*4 IN Upper Clear dome Signal & Zero gain.
RCONST(3) - REAL*4 IN Upper Clear dome Thermistor: constant
RCONST(4) - REAL*4 IN Upper Clear dome Thermistor: coeff x.
RCONST(5) - REAL*4 IN Upper Red dome Signal & Zero const.
* RCONST(6) - REAL*4 IN Upper Red dome Signal & Zero gain.
RCONST(7) - REAL*4 IN Upper Red dome Thermistor: constant
RCONST(8) - REAL*4 IN Upper Red dome Thermistor: coeff x.
RCONST(9) - REAL*4 IN Upper I/R dome Signal & Zero const.
* RCONST(10) - REAL*4 IN Upper I/R dome Signal & Zero gain.
RCONST(11) - REAL*4 IN Upper I/R dome Thermistor: constant
RCONST(12) - REAL*4 IN Upper I/R dome Thermistor: coeff x.
RCONST(13) - REAL*4 IN Lower Clear dome Signal & Zero const.
* RCONST(14) - REAL*4 IN Lower Clear dome Signal & Zero gain.
RCONST(15) - REAL*4 IN Lower Clear dome Thermistor: constant
RCONST(16) - REAL*4 IN Lower Clear dome Thermistor: coeff x.
RCONST(17) - REAL*4 IN Lower Red dome Signal & Zero const.
* RCONST(18) - REAL*4 IN Lower Red dome Signal & Zero gain.
RCONST(19) - REAL*4 IN Lower Red dome Thermistor: constant
RCONST(20) - REAL*4 IN Lower Red dome Thermistor: coeff x.
RCONST(21) - REAL*4 IN Lower I/R dome Signal & Zero const.
* RCONST(22) - REAL*4 IN Lower I/R dome Signal & Zero gain.

RCONST(23) - REAL*4 IN Lower I/R dome Thermistor: constant

RCONST(24) - REAL*4 IN Lower I/R dome Thermistor: coeff x.

(* also contains an offset evaluated to ICONF()).

IFRQ(par) - INT*4 IN Input frequency of each sample. IRAW(n,par)- INT*4 IN Raw instrument voltage conversion. (samples n=1; par=81-89, 91-99) RDER(op,opar)REAL*4 OUT Raw flux signal, zero-offset signal and instrument temperature. (samples op=1; opar=673-690)

SUBPROGRAMS ITSTFLG, ISETFLG

REFERENCES Equations from MRF Instrument section.

CHANGES 020490 Revised range limits introduced. ADH

100191 ADH a) Range limits revised to allow for Pyranometer changes b) New arrays to hold raw input, constants etc for more straightforward indexing. c) Include ICONF to aid reconfiguring instrument types.

010891 Range limits for ZERO now in terms of DRS units, revised limits in Wm-2 for signal.

030292 Rates of change checks instituted on all BBR inputs. ADH

120698 Bug fixed in quality control processing when using non- standard configurations. MDG/WDNJ

270600 I/R signal maximum increased to stop flagging good data value arbitary, as no explanation of numbers found. 1050. > 1500. DAT

V1.06 02/10/02 Changed to use 16 bit DRS data.

V1.07 27/11/02 Now takes X0 sensitivity constant as well as X1

V1.08 22/07/04 Bug so doesn't crash if first data flagged 3

V1.09 13/08/04 Quality Control zero limits increased for 16 bit data

7.16 rio_sreg.py

class ppodd.pod.p_rio_sreg.**rio_sreg**(*dataset*)

Routine to calculate signal register from DECADES parameters

7.17 rio_sun.py

class ppodd.pod.p_rio_sun.**rio_sun**(*dataset*)

FORTRAN routine C_GSUN

ROUTINE C_SUN SUBROUTINE FORTVAX C_SUN.FOR

PURPOSE PUT SOLAR ZENITH AND AZIMUTH ANGLES IN MFD

DESCRIPTION Given date, time and location on the earth's surface this routine puts a solar zenith and azimuth angle in the array of derived parameters. It computes a value once every second. The angles are only obtained if all the flags are set to less than 3 and the date, time and location are all within sensible limits. Any flags set on input are also set in the solar angles derived. If the input is in error or the flags are set to 3 a value of -99. is returned for ZEN and AZIM. To test the routine: $ FOR C_SUN $ FOR TEST_C_SUN $ LINK TEST_C_SUN,C_SUN Ensure contents of files RCONST.DAT and TEST_C_SUN.DAT contain simulated data you require to test the routine with.

VERSION 1.02 1st May 1992 J.A.Smith

ARGUMENTS

RDER(1,515) R*4 IN Time GMT (seconds from midnight)
RDER(1,550) R*4 IN Omega latitude degrees (north +ve)
RDER(1,551) R*4 IN Omega longitude degrees (east +ve)
or RDER(1,541) R*4 IN INU latitude degrees (north +ve)
or RDER(1,542) R*4 IN INU longitude degrees (east +ve)
RCONST(1) R*4 IN Day in month (1-31)
RCONST(2) R*4 IN Month in year (1-12)
RCONST(3) R*4 IN Year (eg 1984)
RDER(1,642) R*4 OUT Solar azimuth in degrees
RDER(1,643) R*4 OUT Solar zenith in degrees

SUBPROGRAMS S_SUN , ITSTFLG, ISETFLG

CHANGES 01 Range checks for input data now done in S_SUN RWS 30/10/90

1.02 Check added if time RSECS has reached midnight and if so to reduce RSECS to less than 86400 s and increase the date. JAS 1/5/92

1.03 Following the demise of the Omega, now uses INU position for flights after 30/09/97. Note that this routine is now always called by CALIBRATE, even if neither Omega or INU were available. WDNJ 20/10/97

1.04 Now strips flags from data before use. WDNJ 22/12/97

1.05 Can take GIN input 05/09/07

1.06 Changes made how lon/lat input is derived AxW 29/03/10

7.18 rio_temps.py

class `ppodd.pod.p_rio_temps.`**`rio_temps`**(*dataset*)
FORTRAN routine C_TEMPS2

ROUTINE C_TEMPS2 SUBROUTINE FORTVAX

PURPOSE Produces calibrated deiced and non-deiced temperatures

DESCRIPTION Calculates indicated and true air temperatures in K for the Deiced and Non-Deiced temperature sensors as follows:

519 - Indicated Air Temperature from Deiced [K] at 32Hz
520 - True Air Temperature from Deiced [K] at 32Hz
524 - Indicated Air Temperature from Non-deiced [K] at 32Hz
525 - True Air Temperature from Non-deiced [K] at 32Hz

Note that this module only processes data recorded on the 146 which only uses one parameter per temperature.

The Deiced Temperature is recorded on the DRS at 32Hz as parameter 10 and the Non-deiced Temperature is recorded on the DRS as parameter 23.

Indicated Air Temperature is derived by application of the appropriate second order calibration coefficients to the raw data.

A correction for heating from the deicing heater is made to the deiced indicated air temperature if the heater is switched on, as indicated by bit 5 of the signal register (parameter 27) being clear. This heating correction is obtained from graphs of Temperature vs Machno in Rosemount Technical Reports 7597 & 7637. If Machno is less than 0.1 the data is flagged 1, because the Rosemount graph is invalid below 0.1, and if Machno below 0.05, a value of 0.05 is use to ensure a valid logarithm. The algorithm used for heating correction is:

```
(exp(exp(1.171+(log(Machno)+2.738)*(-0.000568*(s+q)-0.452))))*0.1
where: s=static pressure       [mbs]
       q=pitot static pressure [mbs]
```

True Air Temperature is derived as:

```
TAT[K] = (Indicated Air Temperature[K]) /
                     (1.0 +(0.2 * MACHNO * MACHNO * TRECFTR))
where: MACHNO  is computed by scientific subroutine S_MACH.
       TRECFTR is the Temperature recovery factor - used to
         compensate for effects of kinetic heating.
         This is supplied as a constant from the
         flight constants file to this routine.


It can be calculated from flight results of slow/fast runs as::

  (Tindfast-Tindslow)/(Ffast*Tindslow-Fslow*Tindfast)
    where: Tind = indicated temperature [K]
           F    = 0.2 * Machno * Machno
```

FLAGGING

Both deiced and non-deiced temperature calculations follow a similar scheme for error flagging, with worst case flags being propagated through the calculations. Sources of error flags are:

Absence of calibration constants - flag 3
Absence of recovery factor constant - flag 3
Static pressure errors - Parameter 576 flag
Pitot pressure errors - Parameter 577 flag
Max/min/rate of change errors - flag 2
Mach No less than 0.1 - flag 1

Not all the above errors need affect all measurements. For instance pressure errors will not affect Indicated Air Temperatures, unless the deicing heater is on. Note that this module cannot be called if any of the raw (not derived) parameters are missing. Also note that no raw data on which this module can be used will be carrying flags (only raw data transcribed on the Gould computer can carry flags). If any temperature has a flag of three, its value is set to 0.0 K (and flagged with a three).

VERSION 1.00 10/09/92 W.D.N.JACKSON

ARGUMENTS

Constants

RCONST(1) Recovery factor for Deiced sensor

RCONST(2) Recovery factor for Non-deiced sensor
RCONST(3) Deiced X0 calibration constant (deg C)
RCONST(4) Deiced X1 calibration constant (deg C)
RCONST(5) Deiced X2 calibration constant (deg C)
RCONST(6) Non-deiced X0 calibration constant (deg C)
RCONST(7) Non-deiced X1 calibration constant (deg C)
RCONST(8) Non-deiced X2 calibration constant (deg C)

Inputs

DEICED TEMPERATURE [bits 0-15] Para 10 32Hz
NON DEICED TEMPERATURE [bits 0-15] Para 23 32Hz
SIGNAL REGISTER [drs units-bcd] Para 27 2Hz
STATIC PRESSURE [mbs] Para 576 32Hz
PITOT STATIC PRESSURE [mbs] Para 577 32Hz

Outputs

INDICATED AIR TEMPERATURE (Deiced) [K] Para 519 32Hz
TRUE AIR TEMPERATURE (Deiced) [K] Para 520 32Hz
INDICATED AIR TEMPERATURE (NonDeiced)[K] Para 524 32Hz
TRUE AIR TEMPERATURE (NonDeiced)[K] Para 525 32Hz

SUBPROGRAMS

S_MACH Calculates Mach no
ITSTFLG Examines bits 16,17 for flags
ISETFLG Sets flag bits 16,17 = 0 –> 3
S_QCPT Performs range and rate of change check

REFERENCES Code adapted from C_TEMPS module. See MRF Internal Note 55 - ‘Temperature Measurement Working Group Report’ for full details of C-130 temperature measurement.

CHANGES V1.01 27/09/02 W.D.N.JACKSON Changed to handle 16 bit temperature recording.

V1.02 23/05/05 D.A.TIDDEMAN Temperature heater correction changed to opposite sense Now raw para 27 bit 5 on = heater on

7.19 p_rio_tpress.py

class ppodd.pod.p_rio_tpress.**rio_tpress**(*dataset*)

FORTRAN routine C_TPRESS

ROUTINE C_TPRESS SUBROUTINE FORTVAX

PURPOSE Calibrates the five turbulence probe pressure transducers into mb.

DESCRIPTION Apply calibration the combined transducer and DRS coefficients to DRS parameters 215 to 219 to obtain derived parameters 773 to 777. Invalid data is flagged with 3, data outside limits is flagged with 2.

METHOD For each DRS parameter to be calibrated

1. If data is FFFF or FFFE then flag 3
2. Apply the calibration constants
3. Check the results for being within acceptable values.

4. Set data flag bits (16+17) 0: Good data

1. Data of lower quality
2. Probably faulty, exceed limits
3. Data absent or invalid.

Flagging If a value can't be computed, due to missing data missing constants, divide be zeroes, etc, a value of 0 is used, flagged with a three. If a value is outside its limits for range or rate of change, it is flagged with a two. If there are no problems with the data it is flagged with 0.

Missing/corrupt data output as 0 flagged 3. Out of range data flagged 2.

VERSION 1.00 23/07/03 W.D.N.JACKSON

ARGUMENTS

Inputs

para 215 TBP1 32 Hz Turbulence probe centre port
para 216 TBP2 32 Hz Turbulence probe attack ports
para 217 TBP3 32 Hz Turbulence probe sideslip ports
para 218 TBP4 32 Hz Turbulence probe attack check
para 219 TBP5 32 Hz Turbulence probe sideslip check

Constants

RCONST(1 to 4) Para 215 cal constants X0 to X3
RCONST(5 to 8) Para 216 cal constants X0 to X3
RCONST(9 to 12) Para 217 cal constants X0 to X3
RCONST(13 to 14) Para 218 cal constants X0 to X1
RCONST(15 to 16) Para 219 cal constants X0 to X1

Outputs

para 773 TBP0 mb 32 Hz Centre pressure
para 774 TBPA mb 32 Hz Attack pressure
para 775 TBPB mb 32 Hz Sideslip pressure
para 776 TBPC mb 32 Hz Attack check pressure
para 777 TBPD mb 32 Hz Sideslip check pressure

SUBPROGRAMS ISETFLG

REFERENCES

CHANGES V1.00 23/07/03 WDNJ Original version Note that V1.00 has no limit checking and no use is made of the check pressures.

V1.01 25/03/04 WDNJ Now takes third order calibration constants for the main transducers, and first order for the check transducers.

V1.02 26/01/06 Phil Brown Realistic min/max values provided for centre-port, Pa, Pb for flagging purposes. Values alsoe provided for check pressures Ca, Cb based on current (and probably wrong) calibration coefficients.

V1.03 09/02/11 Axel Wellpott From an email from Phil Brown: "The P0-S10 differential pressure (para 773) is flagged 2 if it exceeds 130.0 hPa. This is easily exceeded when we do acceleration to max speed (min Angle of Attack) so all the subsequent parameters calculated n C_TURB.for end up with a flag-3 saetting. I reckon a better value would be 180.0 hPa."

7.20 rio_wow.py

class ppodd.pod.p_rio_wow.**rio_weight_on_wheels**(*dataset*)

Weight on wheels indicator.

The weight on wheel status is logged on the PRTAFT CRIO.

DESCRIPTION

1 aircraft is on the ground

0 aircraft is in the air

FLAGGING No flagging needed for this variable. A dummy flag is added to the core netCDF for consistency (all other variables have a flag variable). A lot of code will expect a _FLAG variable for every variable in the dataset.

7.21 p_gin.py

class ppodd.pod.p_gin.**gin**(*dataset*)

GIN processing

Purpose: Resample GIN data to 32Hz so it matches the time frame of turbulence and temperature data

Description: Use numpy.interpolation to resample frequency from 50 to 32Hz Missing out times where the gap is greater than 0.5 sec (see resample.createtimes)

@author: Dave Tiddeman

7.22 p_turb.py

class ppodd.pod.p_turb.**turb**(*dataset*)

FORTRAN routine C_TURB

ROUTINE C_TURB

PURPOSE To calibrate and apply designated correction factors to angle of attack (AOA), angle of sideslip (AOSS) and the centre-static differential pressure (to derive TAS)).

DESCRIPTION Calibration of AOA and AOSS is assumed to be of the form:

```
PA/q = a0(M) + a1(M)*alpha
PB/q = b0(M) + b1(M)*beta
```

where q is the pitot(dynamic) pressure.

Calculations follow the scheme described in BAES doc ADE-46S-R-463-34 1233 (Page 78 of 116). Initial value of pitot pressure is taken from RVSM and used to calculate first guess AOA and AOSS. These are to derive corrections to the centre-port along with separate calculation of static position error in the centre-port measurement. AOA and AOSS are recalculated with iteration continuing until specified tolerance is achieved or max.iteration count exceeded. Corrected centre-port pressure is then used to calculate TAS (currently only the dry value) using:

```
TAS = Corrtn.fac * 340.294*M*SQRT(T/288.15)
```

VERSION 1.01 Phil Brown 24/5/2004

CHANGES 1.02 Phil Brown 11/6/2004 Logic changed to reproduce PVWAVE test version MRFB:[BROWN.PVWAVE]TURB.PRO at this date

1.03 Phil Brown 28/6/2004 Check flags and values following return of calls to S_MACH. Unacceptable causes C_TURB to return its default values of output parameters (flag 3)

1.04 Phil Brown 2/7/04 Uses G_MACH routine to calculate Mach no. and avoid complications due to flagging.

1.05 Phil Brown 08/07/04 Uses simpler Mach-dependent PE.Corrtn derived empirically from B001-012 s&l legs.

1.06 Phil Brown 09/07/04 No position error correction currently applied to P0 differential pressure.

1.07 Phil Brown 26/08/04 Change sign of AOSS. Cals were done against INS drift angle (-ve for +ve AOSS).

1.08 Phil Brown 27/8/04 AOSS calcs revert to original, but assumed to use new fit coefficients for B0 and B1

1.09 26/01/06 Phil Brown Min/max limits provided for AoA, AoSS and TAS for flagging purposes.

1.10 20/06/06 Phil Brown Takes additional input of non-deiced temp, used as alternative when de-iced is flagged 2 or more.

1.11 24/10/06 Phil Brown Fix bug setting flag on TTND to zero before use. Define 4 additional flight constants to apply fudge factors to the calculated AoA / AoSS These have the form: AoA_new = AoA * ALPH1 + ALPH0 AoSS_new= AoSS * BET1 + BET0

1.12 08/10/2010 Axel Wellpott added line “DATA TAS/-9999./” Missing TAS data values were set to -999. and not to -9999. as specified in the netcdf files.

SUBROUTINES S10_PECORR, ITSTFLG, ISETFLG, G_MACH

FILES

REFERENCES

CONSTANTS RCONST(1-3) Coefficients of 2nd order polynomial in Mach to calculate AOA offset, A0

RCONST(4-6) Coefficients of 2nd order polynomial in Mach to calculate AOA sensitivity, A1

RCONST(7-9) Coefficients of 2nd order polynomial in Mach to calculate AOSS offset, B0

RCONST(10-12) Coefficients of 2nd order polynomial in Mach to calculate AOSS sensitivity, B1

RCONST(13) Tolerance for AOA/AOSS iteration

RCONST(14) True Airspeed correction factor (fudge factor to remove residual along-heading wind errors).

RCONST(15-16) Coefficients of linear correction to calculated AoA

RCONST(17-18) Coefficients of linear correction to calculated AoSS

INPUT

PARAMETERS
516 IAS 32Hz
520 TTDI 32Hz
525 TTND 32Hz

576 SPR 32Hz
577 PSP 32Hz
578 PHGT 32Hz
773 TBP0 32Hz
774 TBPA 32Hz
775 TBPB 32Hz
776 TBPC 32Hz
777 TBPD 32Hz

OUTPUT

PARAMETERS
548 AOA 32Hz deg
549 AOSS 32Hz deg
779 TASD 32Hz ms-1
780 TASW 32Hz ms-1
781 TPSP 32Hz mb

7.23 p_twc_calc.py

class `ppodd.pod.p_twc_calc.`**`twc_calc`**(*dataset*)
Use the fit to create a Mass Mixing ratio for the TWC and then a dewpoint.

7.24 p_twc_fit_ge

class `ppodd.pod.p_twc_fit_ge.`**`twc_fit_ge`**(*dataset*)
Fit TWC sensor to GE chilled mirror

Calculates Vapour pressures from GE where less than a threshold (to screen out liquid and ice) Calculate a theoretical Oxygen absorption from pressure Fit the TWC detector less the oxygen correction against the GE vapour pressure

7.25 p_twc_fit_wvss

class `ppodd.pod.p_twc_fit_wvss.`**`twc_fit_wvss`**(*dataset*)
Fit TWC sensor to WVSS

Calculates Vapour pressures from GE where less than a threshold (to screen out liquid and ice) Calculate a theoretical Oxygen absorption from pressure Fit the TWC detector less the oxygen correction against the GE vapour pressure

7.26 p_noturb_windvectors.py

class `ppodd.pod.p_noturb_windvectors.`**`noturb_windvectors`**(*dataset*)
Calculation of windvectors that do not rely on the turbulence probe in the radom of the aircraft. The data are especially useful in icing conditions, when the pressure sensors in the radom are blocked by ice and no tub

INPUTS

VELE_GIN
VELN_GIN
HDG_GIN
TAT_DI_R
TAS_RVSM
ROLL_GIN

tas_scale_factor = 0.9984

FLAGGING The flag is inherited from the input variables. The flag is determined by choosing the worst flag from the input variables. In addition to this a roll angle threshold is used, which is set to 1.5 degrees. All values with an absolute roll angle greater than this are flagged "3".

CHAPTER 8

FORTRAN Modules Source

Below are the headers from the FORTRAN modules of the decades-pp library. The source code can be found in the repository on github.

8.1 c_airspd.for

```
C
C ROUTINE           C_AIRSPD SUBROUTINE FORTVAX
C
C PURPOSE           Derives Indicated and True Airspedd (paras 516 & 517)
C
C DESCRIPTION       True Air Speed is the component of air flow parallel to
C                   the Aircraft's longitudinal axis.
C
C                   IAS =  340.294
C                          * Mach no.
C                          * SQRT(Static Pressure[mb]/ 1013.25)      in ms-1
C
C                   TAS =  A/S correction factor
C                          * 340.294
C                          * Mach no.
C                          * SQRT(De-iced True Air Temp[K] / 288.15) in ms-1
C
C                   where:
C
C                     288.15   is ICAO Standard temperature [K] at zero altitude.
C                     340.294  is speed of sound [ms-1]         at zero altitude.
C                     Mach no. is computed by subroutine S_MACH.
C
C                   Flagging - If a value can't be computed, due to missing data
C                   missing constants, divide be zeroes, etc, a value of 0 is
C                   used, flagged with a three.  If a value is outside its
C                   limits for range or rate of change, it is flagged with a two.
```
(continues on next page)

(continued from previous page)

```
C                      If there are no problems with the data it is flagged with 0.
C                      Any flags on input data are propagated through subsequent
C                      calculations.
C
C VERSION              1.00  020190   A.D.HENNINGS
C
C ARGUMENTS            For Indicated Airspeed:
C                        RDER(IN,576) REAL*4 IN  Derived static pressure in mb
C                        RDER(IN,577) REAL*4 IN  Derived Pitot static pressure in mb
C                                                (samples IN = 1,32 )
C                        RDER(OP,516) REAL*4 OUT Derived Indicated Airspeed  ms-1
C                                                (samples OP = 1,32 )
C                      For True Airspeed:
C                        RCONST(1)    REAL*4 IN  True Airspeed correction factor
C                        RDER(IN,576) REAL*4 IN  Derived static pressure in mb
C                        RDER(IN,577) REAL*4 IN  Derived Pitot static pressure in mb
C                        RDER(IN,520) REAL*4 IN  Derived De-iced True air temp deg K
C                                                (samples IN = 1,32 )
C                        RDER(IN,525) REAL*4 IN  Derived Non-Deiced True air temp deg K
C                                                (samples IN = 1,32 )
C                        RDER(OP,517) REAL*4 OUT Derived True Airspeed  ms-1
C                                                (samples OP = 1,32 )
C
C SUBPROGRAMS          S_MACH, S_QCPT, ITSTFLG, ISETFLG
C
C REFERENCES           Code adapted from MRF1/HORACE
C                      n.b. RCONST(1) (Air speed correction factor 'K' should be
C                           determined by 'K & Gamma' aircraft runs. The value
C                           in RCONST(1) is unity. Experimental values have been
C                           found between 0.98 and 1.02; (HORACE used 1.002 from
C                           June 1988 - Jan 1990., value suggested by S.Nicholls
C                           after JASIN experiment).
C
C CHANGES              V1.01  02/06/93  Limit on max rate of change between
C                      adjacent samples has been increased to 3.3 m/s.  This is
C                      based on analysis of the high turbulence A257 flight, where
C                      the histogram of rates of change showed meaningful changes
C                      of up to 3.0 m/s between adjacent samples. (WDNJ)
C                      Also changed so that data with flags of 2 are processed
C                      rather than rejected and flags are stripped from data before
C                      processing. (WDNJ)
C
C                      V1.02 20/06/06 If TAT_DI flag is 2 or more, then takes
C                      temperature input from TAT_NDI (Phil Brown)
C
C*******************************************************************************
      SUBROUTINE C_AIRSPD(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.02'
C
      IMPLICIT  INTEGER*4 (I)
      IMPLICIT  REAL*4    (R)
      INTEGER*4 IRAW(64,512),IFRQ(512)
      REAL*4    RCONST(64),RDER(64,1024)

      DATA      R516ERCNT,R517ERCNT /2*1.0/       !Stores S_QCPT error counts
      DATA      RLV516,RLV517       /2*0.0/       !Stores latest values
      DATA      RLT516,RLT517       /2*0.0/       !Stores latest times
```

(continues on next page)

(continued from previous page)

```
      PARAMETER IDATFRQ=32             !Output frequency
      PARAMETER R516MX=140.            !Maximum value for IAS - m/s
      PARAMETER R516MN=0.              !Minimum value for IAS - m/s
      PARAMETER R516RG=3.3             !Max diff between IAS 32 Hz samples - m/s
      PARAMETER R517MX=215.            !Maximum value for TAS - m/s
      PARAMETER R517MN=0.              !Minimum value for TAS - m/s
      PARAMETER R517RG=3.3             !Max diff between TAS 32 Hz samples - m/s
C
C Note that if this routine does not compute the IAS or TAS for any reason then
C CALIBRATE will automatically use values of zero flagged with threes.
C
      SAVE
      RSEC=RDER(1,515)                            !Time, secs past midnight
      ICORFLG=ITSTFLG(RCONST(1))                  !Note correction factr flg
      DO IS=1,IDATFRQ                             !For each data sample
        ISTPFLG=ITSTFLG(RDER(IS,576))             !Note Static press flag
        IPSPFLG=ITSTFLG(RDER(IS,577))             !Note Pitot Static flag
        ITATFLG_DI=ITSTFLG(RDER(IS,520))          !Note DI True Air temp flag
        ITATFLG_NDI=ITSTFLG(RDER(IS,525))         !Note NDI True Air temp flag
        RSTP=RDER(IS,576)                         !Take latest STP
        RTAT_DI=RDER(IS,520)                         !Take latest DI TAT
        RTAT_NDI=RDER(IS,525)                         !Take latest NDI TAT
        CALL ISETFLG(RSTP,0)                      !Clear flag from data
        CALL ISETFLG(RTAT_DI,0)                      !Clear flag from data
        CALL ISETFLG(RTAT_NDI,0)                      !Clear flag from data
C
C Derive Mach Number
C
        IMACFLG=3                                 !Default flag on Mach no
        ITMPFLG=MAX(ISTPFLG,IPSPFLG)
        IF(ITMPFLG.LT.3.AND.RSTP.GT.0) THEN       !If no flag 3 data
          CALL S_MACH(RDER(IS,576),RDER(IS,577),RMACH) !Compute Mach number
          IMACFLG=ITSTFLG(RMACH)                  !Note Mach no flag
          CALL ISETFLG(RMACH,0)                   !Clear flag from data
        END IF
C
C Derive Indicated Air Speed
C
        IIASFLG=MAX(IMACFLG,ISTPFLG)
        IF(IIASFLG.LT.3.AND.RSTP.GT.0) THEN       !If no flag 3 data
          RDER(IS,516)=340.294*RMACH*SQRT(RSTP/1013.25) !Derive IAS
          CALL S_QCPT(RSEC,RLT516,RDER(IS,516),RLV516, !Quality control point
     -        R516MX,R516MN,R516RG,3.,R516ERCNT,IQFLAG)
          IIASFLG=MAX(IIASFLG,IQFLAG)             !Check Q/C flag
          CALL ISETFLG(RDER(IS,516),IIASFLG)      !Apply flag
        END IF
C
C Derive True Air Speed
C
        RDER(IS,517)=0.0
        CALL ISETFLG(RDER(IS,517),3)              ! default zero-flag3
C
        IF(ITATFLG_DI.LE.1) THEN                  ! If DI TAT OK use for calcs
          ITASFLG=MAX(IMACFLG,ITATFLG_DI,ICORFLG)
          IF(ITASFLG.LT.3.AND.RTAT_DI.GT.0) THEN       !If no flag 3 data
            RDER(IS,517)=RCONST(1)*340.294*RMACH*SQRT(RTAT_DI/288.15) !Derive TAS
```

(continues on next page)

(continued from previous page)

```
          CALL S_QCPT(RSEC,RLT517,RDER(IS,517),RLV517, !Quality control point
     -        R517MX,R517MN,R517RG,3.,R517ERCNT,IQFLAG)
          ITASFLG=MAX(ITASFLG,IQFLAG)             !Check Q/C flag
          CALL ISETFLG(RDER(IS,517),ITASFLG)      !Apply flag
        ENDIF
      ELSE                                       ! otherwise use NDI TAT
        ITASFLG=MAX(IMACFLG,ITATFLG_NDI,ICORFLG)
        IF(ITASFLG.LT.3.AND.RTAT_NDI.GT.0) THEN        !If no flag 3 data
          RDER(IS,517)=RCONST(1)*340.294*RMACH*SQRT(RTAT_NDI/288.15) !Derive TAS
          CALL S_QCPT(RSEC,RLT517,RDER(IS,517),RLV517, !Quality control point
     -        R517MX,R517MN,R517RG,3.,R517ERCNT,IQFLAG)
          ITASFLG=MAX(ITASFLG,IQFLAG)             !Check Q/C flag
          CALL ISETFLG(RDER(IS,517),ITASFLG)      !Apply flag
        ENDIF
      ENDIF

      END DO                                     !Next sample
      RETURN
      END
```

8.2 c_check.for

```
C
C ROUTINE          C_CHECK SUBROUTINE FORTVAX
C
C PURPOSE          Lists the inputs and outputs to a module every call
C
C DESCRIPTION      Types out all the inputs to and all the outputs from
C                  a module in the CALIBRATION program that has been selected
C                  with the /CHECK command option.  This routine is executed
C                  once a second.
C
C VERSION          1.00  1-9-90  N.JACKSON
C
C ARGUMENTS        IM            I*4 IN The number of the module being checked
C                  IDRS(64,512)  I*4 IN The raw data array
C                  RDER(64,1024) I*4 IN The derived data array
C                  IMDINP(32,64) I*4 IN The input parameters(up to 32) for each modl
C                  IMDOUT(32,64) I*4 IN The output params (up to 32) for each module
C                  RCONST(64,64) R*4 IN The constants (up to 64) for each module
C                  INFREQ(512)   I*4 IN The frequency of each input parameter
C                  IOUTFRQ(1024) I*4 IN The frequency of each output parameter
C                  CMDNAME(64)   C*6 IN The name of each module
C
C CHANGES          1.01  13/05/93  W.D.N.JACKSON
C                  Now displays input data as 16 bit rather than 12.
C
********************************************************************************
      SUBROUTINE C_CHECK(IM,IDRS,RDER,IMDINP,IMDOUT,RCONST,INFREQ
     &   ,IOUTFRQ,CMDNAME)
CDEC$ IDENT 'V1.01'
      INTEGER*4   IM               !The number of the module being checked
      INTEGER*4   IDRS(64,512)     !The raw data array
      REAL*4      RDER(64,1024)    !The derived data array
```

(continues on next page)

(continued from previous page)

```
      INTEGER*4   IMDINP(32,64)   !The input parameters(up to 32) for each modl
      INTEGER*4   IMDOUT(32,64)   !The output params (up to 32) for each module
      REAL*4      RCONST(64,64)   !The constants (up to 64) for each module
      INTEGER*4   INFREQ(512)     !The frequency of each input parameter
      INTEGER*4   IOUTFRQ(1024)   !The frequency of each output parameter
      CHARACTER   CMDNAME(64)*6   !The name of each module
      INTEGER*4   ITSTFLG
      REAL*4      RQRQ

      CHARACTER   CTIM*8
      PARAMETER TT=6

      CALL C_SPMCTIM(NINT(RDER(1,515)),CTIM) !Get the time from para 515 as string
      WRITE(TT,*) CMDNAME(IM)//' for '//CTIM !Write module name and time
C
C Search backwards through constants for first valid one
C Message if none found
C Else write out each valid constant
C
      IC=64                           !Search backwards for constants
      DO WHILE(ITSTFLG(RCONST(IC,IM)).EQ.3.AND.IC.GE.1)
        IC=IC-1
      END DO
      IF(IC.EQ.0) THEN                !Message if none found
        WRITE(TT,*) 'No constants'
      ELSE
        WRITE(TT,*) 'Constants:'      !Else print them
        DO I=1,IC
          WRITE(TT,10,IOSTAT=IOS)
     &    RCONST(I,IM),ITSTFLG(RCONST(I,IM))
        ENDDO
      END IF
C
C For each parameter in the input list for the module, list all values
C together with the flag indicator value.
C
      II=1                            !Pointer in module list
      DO WHILE(IMDINP(II,IM).NE.0.AND.II.LE.32) !For each input param
        IP=IMDINP(II,IM)              !Parameter number
        WRITE(TT,*) 'Input parameter ',IP
        IF(IP.LE.512) THEN            !Write raw data as integers
           IF(INFREQ(IP).GT.0) THEN
            DO I=1,INFREQ(IP)
              WRITE(TT,11,IOSTAT=IOS)
     &        IDRS(I,IP).AND.'FFFF'X
     &        ,ITSTFLG(IDRS(I,IP))
            ENDDO
           END IF
        ELSE                          !Write derived data as real
          IF(IOUTFRQ(IP).GT.0) THEN
            DO I=1,IOUTFRQ(IP)
              RQRQ=RDER(I,IP)
              CALL ISETFLG(RQRQ,0)
              WRITE(TT,10,IOSTAT=IOS)
     &        RQRQ,ITSTFLG(RDER(I,IP))
            END DO
          END IF
```

(continues on next page)

(continued from previous page)

```
        END IF
        II=II+1
      END DO
C
C For each parameter in the output list for the module, list all values
C together with the flag indicator value.
C
      II=1                             !Pointer into module list
      DO WHILE(IMDOUT(II,IM).NE.0.AND.II.LE.32) !For each parameter
        IP=IMDOUT(II,IM)               !Parameter number
        WRITE(TT,*) 'Output parameter ',IP !Write out all values
        IF(IOUTFRQ(IP).GT.0) THEN
          DO I=1,IOUTFRQ(IP)
              RQRQ=RDER(I,IP)
              CALL ISETFLG(RQRQ,0)
            WRITE(TT,10,IOSTAT=IOS)
     &      RQRQ,ITSTFLG(RDER(I,IP))
          ENDDO
        END IF
        II=II+1
      END DO
      WRITE(TT,*) ' '                  !Blank line at end
      RETURN
10    FORMAT(5(1PE11.4E1,I2,2X))
11    FORMAT(8(I6,I2,2X))
      END
```

8.3 c_comr.for

```
C
C ROUTINE         C_COMR SUBROUTINE FORTVAX
C
C PURPOSE         A subroutine to calculate Carbon monoxide.
C
C DESCRIPTION         The CO analyser outputs one measurement.
C                 This is input to the program as DRS bits, and converted
C                 into PPB by multiplying the DRS bits by a calibration factor.
C
C
C TO COMPILE         $FORT C_COMR
C
C VERSION         1.00  8-Jul-2004          D.Tiddeman
C                1.01  27-OCT-2005
C                1.02
C                1.03 31-JAN-2007 R Purvis Changed timedelay after cal to 20
C                1.04 18-SEP-2007 R Purvis        RCONST(5) added for correction factor
C                1.05 30-JUL-2010 S Bauguitte increased CO flag count threshold from␣
↪8000 to 10000
C                1.06 15-OCT-2012 A Wellpott CO upper threshold flagging added. Now␣
↪values above
C                                 4995 are flagged with 3
C
C ARGUMENTS         IRAW(1,154) - on entry contains the raw CO signal
C                IRAW(1,223) - on entry contains raw RVSM airspeed
```

(continues on next page)

(continued from previous page)

```
C               IRAW(1,113) - cal info ?
C               RCONST(1,2,3,4) XO and X1 voltage cal for CO, v to ppb, ppb offs
C                RDER(1,782) - on exit contains the derived CO signal
C
C*******************************************************************************
      SUBROUTINE C_COMR(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.06'
      IMPLICIT NONE
      INTEGER*4 IRAW(64,1024),IFRQ(512)
      INTEGER*4   IFLG,IS,ITSTFLG
      REAL*4 COMR,RERR
      REAL*4 RCONST(64),RDER(64,1024)
      INTEGER*4 IWASCAL

      SAVE IWASCAL
C
C Set default values
C
      RERR=0.
      CALL ISETFLG(RERR,3)
      RDER(1,782)=RERR

C     Copy across raw signals
C
      COMR=FLOAT(IRAW(1,154))
C
C     Convert CO DRS signals first to voltage, then apply voltage to
C     ppb conversion, then add instrument offset.
C
      COMR=((RCONST(1)+COMR*RCONST(2))*RCONST(3)+RCONST(4))
      IF(ITSTFLG(RCONST(5)).EQ.0)COMR=COMR*RCONST(5)
C
      IFLG=0
      IF(ITSTFLG(RCONST(8)).EQ.0) THEN
        DO IS=1,32
          IF(IRAW(IS,223).LT.RCONST(8)*62) IFLG=1
        ENDDO
      ENDIF
      IF(COMR.LT.0.) IFLG=2
      IF(COMR.GT.4995.) IFLG=3
      IF(IRAW(1,154).EQ.0) IFLG=3
      IF(IRAW(1,154).EQ.'FFFF'X) IFLG=3
      IF(ITSTFLG(RCONST(6)).EQ.0.AND.ITSTFLG(RCONST(7)).EQ.0)THEN
        IF(COMR.LT.RCONST(6).OR.COMR.GT.RCONST(7))IFLG=3
      ENDIF
C      Changed on 30/07/2010 SB
C      IF(IRAW(1,113).GT.8000) IWASCAL=20
      IF(IRAW(1,113).GT.10000) IWASCAL=20
      IF(IWASCAL.GT.0)THEN
        IFLG=MAX(IFLG,2)
        IWASCAL=IWASCAL-1
      ENDIF
      CALL ISETFLG(COMR,IFLG)
      RDER(1,782)=COMR
C
      RETURN
      END
```

8.4 c_geneas.for

```
CDEC$ IDENT 'V1.02'
C
C ROUTINE              C_GENEAS     SUBROUTINE FORTVAX
C
C PURPOSE             Derivation of Dew point
C
C DESCRIPTION       Calculation of Dew Point in K from General Eastern Hygrometer
C
C                          529- Dew Point                              [K]
C
C                   The General Eastern Hygrometer (Parameter 58) is recorded
C                   in binary with a range of 0 to 4095 DRS bits.
C                   A control signal (Parameter 59) is also recorded which
C                   gives an indication of the amount of heating or cooling
C                   of the mirror.
C                   The instrument should be in control if the signal is between
C                   certain limits.
C                   Outside these limits it still produces a dew point
C                   reading,though of doubtful accuracy, and derived data
C                   is flagged - FLAG = 2.
C
C VERSION             1.00   240190    J HARMER
C                   1.01   17-01-96 D Lauchlan
C ARGUMENTS
C                   Constants:
C                   GEMAX      Maximum control condition signal limit RCONST(1)
C                   GEMIN      Minimum control condition signal limit RCONST(2)
C                   CALGE(1)   GE Dew point calib. constant x0        RCONST(3)
C                   CALGE(2)   GE Dew point calib. constant x1        RCONST(4)
C                   CALGE(3)   GE Dew point calib. constant x2        RCONST(5)
C
C
C                   Inputs   :
C                   GENERAL EASTERN 1011 DEW POINT    [drs units ] Para 58
C                   GENERAL EASTERN CONTROL SIGNAL    [drs units ] Para 59
C
C                   Outputs  :
C                   DEW POINT                           [K]      Para 529
C
C SUBPROGRAMS
C                   ITSTFLG          Examines bits 16,17 for flags
C                   ISETFLG          Sets flag bits 16,17 = 0 --> 3
C                   S_QCPT           Performs range and rate of change check
C
C REFERENCES             Code adapted from MRF1/MRF2
C
C CHANGES           v1.01 17-01-96 D Lauchlan
C                   Unused variables removed
C
C                   V1.02  27/09/02  W.D.N.JACKSON
C                   Changed to include handling of 16 bit data from the new
C                   DRS.
C-----------------------------------------------------------------------------
      SUBROUTINE C_GENEAS   (IRAW,IFRQ,RCONST,RDER)
      IMPLICIT    NONE
```

(continues on next page)

(continued from previous page)

```
      INTEGER*4   IRAW(64,512), IFRQ(512)
      REAL*4      RCONST(64), RDER(64,1024)

      INTEGER   ITSTFLG,IQFLAG,IFLAG3
      INTEGER   IFLAG, IFLAG1, IFLAG2, IS, IQ, IT
      REAL*4    GEMAX,GEMIN,CALGE(3)
      REAL*4    R529MX, R529MN, R529RG,RSEC,RVAL,R
      REAL*4    RLV529, RLT529 ,R529ERCNT              !Previous: Values/Time
      DATA      RLV529, RLT529 /2*0./                  !Init first time through
      DATA      R529ERCNT /1*1.0/                      !Init first time through

      PARAMETER (R529MX=324. , R529MN=195. , R529RG=1.)!Limits checks DEWPT [K]
C-----------------------------------------------------------------------------
C Check constants and set up arrays, clear flags
      SAVE
      IFLAG=0
      IFLAG1=0
      IFLAG2=0
      IFLAG3=0
      IQFLAG=0
      DO IT=1,5
        IF (ITSTFLG(RCONST(IT)).EQ.3)IFLAG=3               !Check constant flags
      END DO
C
      GEMAX=RCONST(1)
      GEMIN=RCONST(2)
      DO IQ=3,5
          CALGE(IQ-2)=RCONST(IQ)
      END DO
C
      RSEC = RDER(1,515)                               !Time: seconds from midnight
C
C Calc dew point temperature from General Eastern 1011 Hygrometer
C
      IF (IFRQ(58).GT.0.AND.IFRQ(59).GT.0) THEN
        DO IS = 1,IFRQ(58)                                 !Loop thro samples
          IFLAG1 = 0                                       !Check quality flag
          IF(IRAW(IS,58).EQ.0.OR.IRAW(IS,58).EQ.'FFFF'X) IFLAG1=3
          IFLAG  = IFLAG1
          IF (IFLAG .LT. 3 ) THEN                          !If there is some data
            R=FLOAT(IRAW(IS,58))                           !Get para 58 raw data
            RDER(IS,529)=CALGE(3)*R**2 + CALGE(2)*R        !Apply calib constants
     -                          + CALGE(1)                 ! [deg C]
            RDER(IS,529)=RDER(IS,529)+273.16               !Dew point [K]
            CALL S_QCPT (RSEC,RLT529,RDER(IS,529),RLV529,  !Quality control point
     -                R529MX,R529MN,R529RG,3.,R529ERCNT,IQFLAG)
            IF (IQFLAG .GT. IFLAG) IFLAG = IQFLAG          !Set worst flag
            IQ=((IS * IFRQ(59) - 1) / IFRQ(58)) + 1        !Find control signal
            IFLAG2 = 0                                     !Check quality flag
            IF(IRAW(IS,59).EQ.0.OR.IRAW(IS,59).EQ.'FFFF'X) IFLAG2=3
            IF (IFLAG2 .GT. IFLAG) IFLAG = IFLAG2          !Set worst flag
            RVAL=FLOAT(IRAW(IS,59))                        !Get para 59 raw data
            IF (RVAL.LT.GEMIN .OR. RVAL.GT.GEMAX) THEN     !Control cond. on
                IFLAG3=2
            ELSE
                IFLAG3=0
            ENDIF
```

(continues on next page)

(continued from previous page)

```
            IF  (IFLAG3.GT.IFLAG) IFLAG = IFLAG3            !Set worst flag
            CALL ISETFLG(RDER(IS,529),IFLAG)                !Put back worst flag
          ENDIF
        END DO
      ENDIF
C     RLV529= RDER(IFRQ(58),529)                            !Preserve last value
C                                                           !Done within S_QCPT
C                                                            !without the flag
C      SAVE RLT529,RLV529                                     !ANSI Fortran

      RETURN

      END
```

8.5 c_grflux.for

```
C------------------------------------------------------------------------------
C ROUTINE             C_RFLUX        SUBROUTINE FORTVAX       [C_RFLUX.FOR]
C
C PURPOSE              CORRECT RAW FLUXES FOR PYRANOMETERS AND PYRGEOMETERS
C
C DESCRIPTION      Flux corrections are performed for the six instruments
C                  which are normally configured:
C                  Upward-facing :-  Clear dome and Red dome pyranometers.
C                                    Silver dome pyrgeometer.
C                  Downward-facing:- Clear dome and Red dome pyranometers.
C                                    Silver dome pyrgeometer.
C
C                  The actual configuration is specified by the preset array
C                  ICONF, which has six elements whose meaning interpreted as:
C                    1,4 : Clear dome pyranometer  (upper/lower)
C                    2,5 : red    "        "            "     "
C                    3,6 : Silver "   pyrgeometer     "     "
C                  (normally: ICONF(1-3) Upper instruments.
C                             ICONF(4-6) Lower instruments.)
C
C                 Check that the normal configuration of instruments is to
C                 be used. Any changes are indicated by the presence of a large
C                 offset to the last calibration constant for any instrument
C                 (i.e. the obscurer indicator constant).
C                 If this is present the offset is interpreted as a revised
C                 ICONF indicator for that instrument. See note below.]
C
C                  n.b. Lower instruments were fitted w.e.f. Flight H797
C                       Upper instruments were fitted w.e.f. Flight H842
C
C                  This value solely determines the control path through the
C                  routine for the processing of each instruments inputs.
C                  Should the configuration aboard the aircraft be changed
C                  the array ICONF should be adjusted accordingly.
C                  e.g. If ICONF(1) was modified := 2; it would imply that the
C                  'channel' contained raw flux, zero-offset and thermistor
C                  values for a red dome - rather than clear - pyranometer.
C                  The value of ICONF(1) i.e. 2 would determine the processing
```

(continues on next page)

(continued from previous page)

```
C                  path, the selection of the appropriate set of constants
C                  to apply for correction and the range checking.
C
C NOTE             CHANGES FROM STANDARD CONFIGURATION.
C                  Should the configuration of BBR instruments aboard the
C                  aircraft be changed e.g. swapping a red dome for clear dome,
C                  the array ICONF is  adjusted accordingly. The mechanism used
C                  is to add an offset to the sixth constant in the calibration
C                  constants file (i.e. the obscurer) for that instrument.
C                  Example: If the second 'channel' (inputs 674,677,680) which
C                  in the standard configuration is a red dome pyranometer,
C                  was replaced with a second clear dome instrument, the sixth
C                  constant for the second line of the constants for C_RFLUX
C                  would be changed from 1.0000E+0 to 21.0000E+0, the offset
C                  decodes to "2" when detected by this program.
C                  This is assigned to ICONF(2) and would imply that the
C                  'channel' inputs contain raw flux, zero-offset and thermistor
C                  values for a red dome - rather than clear dome - pyranometer,
C                  and should be range-checked for that type of output only.
C
C                  Corrections applied:
C                  --------------------
C                  Pyranometers (Clear and Red dome) are corrected for:
C                  - Subtraction of a zero offset (mean over past 10 seconds)
C                  - Attitude (pitch and roll) -Upper instruments only.
C                    test if flux is above a critical limit indicating a direct
C                    solar beam component.
C                      If not direct, assume diffuse and apply no attitude corr.
C                      If DIRECT, a geometric correction is used to "level"
C                      the instrument to produce the equivalent hemispheric
C                      downward flux through a horizontal surface (without
C                      inclusion of diffuse component).
C                      The ratio of the Direct:Direct+Diffuse components is
C                      assumed to be 0.95 at present. This value could be
C                      optimised for a particular atmosphere depending on the
C                      turbidity as a function of height.
C
C                      Correct for COSINE effect. (MRF Technical note No.7).
C                      [Pitch and roll offsets of the instrument thermopiles
C                      relative to the aircraft INS platform are derived in
C                      flight by flying a box pattern in cloud-free skies -
C                      These offsets are then used in addition to the INS pitch
C                      and roll (meaned over two seconds). (See MRF Technical
C                      note No 4.) and these values are supplied as arguments
C                      four and five in each set of CONSTANTS below.
C                  - Time constant of thermopile relative to INS. The mean of
C                    last two seconds of INS pitch/roll angles are used in the
C                    attitude correction, giving an effective difference of
C                    0.5 seconds.
C                  - Correct flux output for proportion of hemispheric dome
C                    obscured by indicated obscurer pillar. (Rawlins 1986).
C
C                  Pyrgeometers (IR) are corrected for:
C                  - Zero offset (mean over past 10 seconds)
C                  - Temperature sensitivity  (Coefficients in CONSTANTS below)
C                  - Linear dependence 0.2% per degree with sensitivity defined
C                    as unity at zero C. applied to signal. (MRF Int note No 50)
```

(continues on next page)

(continued from previous page)

```
C                  - Calculation of flux (sigma T^4 correction)
C                    Flux = signal +(sigma* Tsink^4)
C                    where sigma = Stefan-Boltzmann constant.
C                  _ Upper instrument is corrected for dome transmission
C                    effects (MRF Tech note 3)
C
C VERSION           1.17 05-09-07   D Tiddeman
C
C METHOD           1. First time routine is called, assign constants to named
C                     program variables/arrays.
C                     Decide on basis of input constants whether upper instr.
C                     data is available to be processed.
C                  2. Derive/convert any intermediate results used multiply
C                     within several code sections following.
C                  3. Derive running mean zero-offsets over the past 10 seconds
C                     for each instrument
C
C                  4. Calculate mean pitch and roll values for the current
C                     second and use them to derive running means for the past
C                     two seconds.
C                  5. Correct thermistor temperatures for non-linearity.
C                  6. Cycle through each of six instrument input channels.
C                     Use the control variable in ICONF() to select execution
C                     of appropriate code sections.
C                     In all cases; derive a signal zero-offset and reduce the
C                     signal flux by this amount.
C                     Apply temperature-dependance corrections to pyranometers.
C                     For upward-facing pyranometers the 'critical' value to
C                     discriminate between diffuse and direct-sun conditions is
C                           FCRIT  = 920.*(COS(ZENRAD))**1.28
C                     where ZENRAD : solar zenith angle (in radians)
C                     [N.B. This approximates to the 'German' equation but is
C                      simpler, and does not produce negative values at low
C                      Sun elevations].
C                     Correct flux output for proportion of hemispheric dome
C                     obscured by indicated obscurer pillar. (Rawlins 1986).
C
C                  7. Range check flux output and set a flag accordingly.
C                     Apply flag values to resulting flux output dependent on
C                     relevant flag settings.
C
C ARGUMENTS        RCONST(1),( 7)..(31)  - REAL*4 IN Temperature Sens. coeff a
C                  RCONST(2),( 8)..(32)  - REAL*4 IN Temperature Sens. coeff b
C                  RCONST(3),( 9)..(33)  - REAL*4 IN Temperature Sens. coeff c
C                  RCONST(4),(10)..(34)  - REAL*4 IN Pitch offset of Instrument
C                  RCONST(5),(11)..(35)  - REAL*4 IN Roll  offset of Instrument
C                  RCONST(6),(12)..(36)  - REAL*4 IN Obscurer pillar type.
C
C                  RDER(1,par)             REAL*4  IN Six raw flux signals W/M-2
C                             (par=673-675,682-684)
C                  RDER(1,par)             REAL*4  IN six zero-offsets   (W/M-2)
C                             (par=676-678,685-687)
C                  RDER(1,par)             REAL*4  IN six instr. temperatures K
C                             (par=679-681,688-690)
C                  RDER(32,560)            REAL*4  IN INS Roll      (degrees)
C                  RDER(32,561)            REAL*4  IN INS Pitch     (degrees)
C                  RDER(32,562)            REAL*4  IN INS heading   (degrees)
```

(continues on next page)

(continued from previous page)

```
C                  RDER(1,642)             REAL*4  IN Solar azimuth (degrees)
C                  RDER(1,643)             REAL*4  IN Solar zenith  (degrees)
C
C                                                                Pos. Dome  Units
C                  RDER(1,1019)            REAL*4 OUT Corrected Upp Clear W/m-2
C                  RDER(1,1020)            REAL*4 OUT flux.      "  Red dome "
C                  RDER(1,1021)            REAL*4 OUT            "  I/R      "
C                  RDER(1,1022)            REAL*4 OUT           Low Clear    "
C                  RDER(1,1023)            REAL*4 OUT            "  Red dome "
C                  RDER(1,1024)            REAL*4 OUT            "  I/R      "
C
C SUBPROGRAMS             ITSTFLG, ISETFLG, S_RUNM, CORR_THM, RMEANOF, CIRC_AVRG
C
C REFERENCES            MRF Internal note  4.
C                   "     "      "   12.
C                   "     "      "   31.
C                   "     "      "   50.
C                   "     "      "   56.
C                  MRF Technical note 3. Pyrgeometer Corrections due to Dome
C                                        Transmission.  February 1991 Kilsby
C                  MRF Technical note 7. Report of Broad-band radiative fluxes
C                                        working group. 17/12/91      Saunders
C                  MRF Technical note 8. Pyramometer calibrationsin Ascension
C                                        of Feb.1992.   4/6/92        Seymour
C                  RAWLINS  R        D/Met.O.(MRF)/13/1      1986.
C                  SAUNDERS R         "        "   "         21/3/90
C                  SAUNDERS R        M/MRF/13/5              22/7/92
C
C CHANGES          10/01/91 A.D.Hennings.
C                    Ability to change ICONF to when reconfiguring instrument
C                    fit on A/C using the constants file.
C                  10/01/91 Pitch & Roll averaging changed from 3 to 2 seconds.
C                  25/01/91 Flags assessment changed; use of new flag IFLAG_SUN
C                  29/01/91 Roll limit checking:replace ROLBAR with ABS(ROLBAR).
C                           Flags assessment changed; IFLAG_OUTPUT being max of
C                           (signal,Pitch,Roll,Zenith) flags.
C                  30/07/91 FCRIT for Red dome now only used if no clear dome
C                  16/10/91 Corrected pyrgeometer temp sensitivity correction
C                  20/01/92 Use INS heading instead of obsolete Omega heading.
C                  03/02/92 New subroutine CIRC_AVRG to calc INS mean heading
C                  21/07/92 Levelling of upper pyranometers changed to use
C                           direct beam component, and cosine effect included.
C                           Recommendations of MRF Tech note 7.   (V1.13)
C                           references to Tech note 8. and M/MRF/13/5
C                  24/07/92 Pyrgeometer corrections for Dome transmission.
C                           (Downwelling) MRF Tech note 3.
C                  17/01/96 D Lauchlan
C                           Unused variables removed
C                  22/12/97 W D N Jackson, Flags cleared from all data before
C                           use.
C                  11/08/98 W D N Jackson, Upper pyranometer obscurer
C                           corrections changed to correct values.  The
C                           values have been incorrect in all previous versions
C                           of C_RFLUX. The error is only small. (Source
C                           P Hignett)
C                  05/09/07 D TIDDEMAN Will use GIN attitude if available rather
C                           then INU
```

(continues on next page)

(continued from previous page)

```
C
C-------------------------------------------------------------------------------
      SUBROUTINE C_GRFLUX     (IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.17'
C
      IMPLICIT NONE
      INTEGER*4 IRAW(64,512), IFRQ(512)
      REAL*4    RCONST(64), RDER(64,1024)

      INTEGER   ITSTFLG
      REAL      CIRC_AVRG              !Function returning average of angles
C
C working input data and processed output arrays
C
      REAL*4 ZIN(6),                   !Zero offset samples
     &       RTHM(6),                  !Uncorrected thermistor samples
     &       RFLX(6),                  !Uncorrected flux samples
     &       THM(6),                   !Corrected thermistor samples
     &       FLX(6),                   !Corrected flux samples
     &       PITINS,ROLINS,            !Input pitch & roll (mean of 32hz) degs
     &       PITCH ,ROLL,              !Corrected pitch and roll (Rads)
     &       HDGINS,                   !Input INS heading (degrees)
     &       SOLAZM,SOLZEN,            !Input Solar Azimuth & zenith angle. Rads
     &       HDGRAD,                   !Convert Omega heading to radians
     &       ZENRAD,                   !Convert Solar Zenith ang to radians
     &       AZMRAD,                   !Convert Solar Zenith ang to radians
     &       SUNHDG                    !Sun Heading (Sol Azm-A/c Omega hdg.)Rads
C
C C0NSTANT information
C
      REAL*8    TSA(6)                       !Temperature senstvty alph,beta gm
     -         ,TSB(6)                       !
     -         ,TSG(6)                       !
     -         ,PITOFF(6)                    !Angular offset   " Pitch.
     -         ,ROLOFF(6)                    !Angular offset   " Roll.
      INTEGER*4 IOBTYP(6)                    !Obscurer type (0: none 1:short
                                             !               2: tall)
C
C flags signifying validity of input arguments and derived values.
C
      INTEGER*4 IFLAG_ANG              !Test of sun angle too low
     -         ,IFLAG_ROLL             !INS Roll
     -         ,IFLAG_PIT              !INS Pitch
     -         ,IFLAG_AZM              !Solar azimuth angle
     -         ,IFLAG_ZEN              ! "    zenith   "
     -         ,IFLAG_INHDG            !INS Heading
     -         ,IFLAG_SHDG             !Sun hdg.  Max(IFLAG_AZM and IFLAG_INHDG)
     -         ,IFLAG_SUN              !Sun attitude Max(Pitch/Roll/Zen/Ang)
     -         ,IFLAG_FLX              !Raw flux input
     -         ,IFLAG_THM              !Corrected thermistor
     -         ,IFLAG_ZER              !Meaned zero-offset
     -         ,IFLAG_SIGNAL           !Max of (IFLAG_FLX and IFLAG_ZER)
     -         ,IFLAG_CORRN            !Max of (all correction flags relevant)
     -         ,IFLAG_OUTPUT           !Max of (IFLAG_SIGNAL and IFLAG_CORRN)
                                       ! and result of range tests on output.
     -         ,IDUM                   !Argument, return value of no interest.
     &         ,IHDG,IPIT,IROL
```

(continues on next page)

(continued from previous page)

```
C  arrays , counters and pointer arguments for Zero-offset mean derivation

      REAL*4    ZBAR(6)                  !Output means over past 10 seconds
      REAL*4    ZBUF(10,6), ZSUM(6)      !Buffer and total holder
      INTEGER*4 IZP(6),     IZCNT(6)     !Buffer pointer and counter of samples.
      DATA      IZP/6*1/,   IZCNT/6*0/   !Initialise ptrs, count of good samples
C
C  arrays , counters and pointer arguments for Pitch and Roll mean derivation
C
      REAL*4    PITBAR,ROLBAR            !Output means over past 2 seconds. degs
      REAL*4    PBUF(3),RBUF(3),PSUM,RSUM!Buffers and total holders
      INTEGER*4 IPPT,IRPT,IPCNT,IRCNT    !Buffer pointer and counter of samples.
      DATA      IPPT,IRPT/1,1/           !Initialise buffer pointers
      DATA      IPCNT,IRCNT/2*0/         !Initialise count of good samples


      LOGICAL   OFIRST/.TRUE./           !Indicator as to first time through rtn

      INTEGER*4 ICONF(6)                !6 input channels (instruments).
      DATA      ICONF/                  !Control variables- Currently set as:
     -                1,                !Upper clear dome pyranometer in chan 1
     -                2,                !      red   dome pyranometer in chan 2
     -                3,                !      silverdome pyrgeometer in chan 3
     -                4,                !Lower clear dome pyranometer in chan 4
     -                5,                !      red   dome pyranometer in chan 5
     -                6/                !      silverdome pyrgeometer in chan 6

      REAL*4   RMAXFLX(6),RMINFLX(6)    !Range limits on corrected flux.
      DATA     RMAXFLX/                 !Max. admissible corrected flux output
     -            1380.,                !Upward-facing  clear  dome pyranometer
     -             700.,                !               red    dome pyranometer
     -             550.,                !               silver dome pyrgeometer
     -            1380.,                !Downward-facing clear dome pyranometer
     -             700.,                !                red   dome pyranometer
     -             750 /                !                silverdome pyrgeometer
      DATA     RMINFLX/                 !Min. admissible corrected flux output
     -             -20.,                !Upward-facing  clear  dome pyranometer
     -             -20.,                !               red    dome pyranometer
     -             -20.,                !               silver dome pyrgeometer
     -             -20.,                !Downward-facing clear dome pyranometer
     -             -20.,                !                red   dome pyranometer
     -              50./                !                silverdome pyrgeometer

      REAL*4    THETA,RCOSTH              !Angle between Sun and Normal to Instr
      REAL*4    ROLLIM,THTMAX             !Roll max limit: Sun-angle max limit
      PARAMETER (ROLLIM=7.0, THTMAX=80.0) !in degrees.
C
C local variables.
C
      LOGICAL   UPPERS                   !Upper instruments fitted?
      INTEGER*4 IS,IE                    !First and last instrument 'channel'
C       SAVE IS,IE
      INTEGER*4 IN,I                     !Instrument (channel); loop index
      REAL*4    FCRIT,FCRITVAL           !Critical flux value (direct/diffuse)
      REAL*8    SIGMA,                   !Stefan-Boltzmann constant.
     -          FOBSC,                   !Obscurer value for any instrument
```

(continues on next page)

(continued from previous page)

```
     -          TH,                         !Place holder for corrected thermistor
     -          FL                          !Place holder for corrected flux
      REAL*4    DEG2RD                      !Degrees to radians conversion factor
      REAL*4    RTEMP,                      !Temp vrb: used with ICONF changes.
     -          ROBTYP                      ! "    " : specify Obscurer type used.
      INTEGER*4 ITYPE,ISIG,ICOR             !Indices to data tables
C
C  levelling corrections
C                                          !Select INDX of solar zenith angle
      INTEGER*4 INDX                       !where  INDX = NINT(SOLZEN/10) + 1
                                           !INDX
                                           !1-3: (0 -29.9 deg)
                                           !4-6: (30-59.9 deg)
                                           !7-9: (60-89.9 deg)
                                           !10:  (  >89.9 deg)

      REAL*4    CEFF(10)/1.010, 1.005, 1.005, !Correction to pyranometers for
     &                   1.005, 1.000, 0.995, !COSINE effect dependant on solar
     &                   0.985, 0.970, 0.930, !zenith angle. Determined by expt
     &                   0.930/               !Ref: Tech note 8. Table 4

      REAL*4    FDIR(10)/.95,.95,.95,      !(Proportion of flux from direct source
     &                   .95,.95,.95,      !for varying solar zenith angles.)
     &                   .95,.95,.95,      !Addressed by INDX as above.
     &                   .95/              !Ref: M/MRF/13/5


C     table of proportion of hemispheric dome obscured by each pillar-type

      REAL*4    ROBSC(3,6)                  !Obscurer corrections (Type,Up|Loc)
      DATA ((ROBSC(ITYPE,IN),IN=1,6),ITYPE=1,3)/     !Ref:RAWLINS 1986
      !    Upper Instruments  |   Lower instruments
      !Port    Starbd  Centre  Port    Starbd  Centre
     & 00.000, 00.000, 00.000, 00.000, 00.000, 00.000,  ! No pillar (Ind=1)
     & 00.010, 00.010, 00.000, 00.000, 00.000, 00.000,  ! Short "    ( "  2)
     & 00.040, 00.040, 00.000, 00.000, 00.000, 00.000/  ! Tall  "    ( "  3)
! The following lines contain the incorrect upper pyranometer corrections which
! have been used in all previous versions of C_RFLUX (WDNJ 11/8/98).
!     & 00.016, 00.016, 00.000, 00.000, 00.000, 00.000,  ! Short "    ( "  2)
!     & 00.046, 00.046, 00.000, 00.000, 00.000, 00.000/  ! Tall  "    ( "  3)

C     logic table combining two group input flag conditions resulting in an
C     output flag.

      INTEGER*4 IFLAG_TABLE(0:3,0:3)
      DATA ((IFLAG_TABLE(ISIG,ICOR),ICOR=0,3),ISIG=0,3)/
      !        CORRECTION
      !     0   1   2   3
      !   -------------------                 See Saunders LM 1990 for
     -      0,  1,  3,  3,  ! 0               details of this table.
     -      1,  2,  3,  3,  ! 1  SIGNAL
     -      2,  2,  3,  3,  ! 2
     -      3,  3,  3,  3  /! 3

      PARAMETER (SIGMA  = 5.669E-08)
      PARAMETER (DEG2RD = 57.295776)
      SAVE
```

(continues on next page)

(continued from previous page)

```
!-----------------------------------------------------------------------------
!+
!  1. First time routine is called, assign constants to named
!     program variables/arrays.


      IF (OFIRST) THEN
        OFIRST= .FALSE.
!
!       Prior to Flight H842 no upper radiometers were recorded in this form;
!       hence no data constants are passed to this routine. Check for condition.
!
        UPPERS = .FALSE.
        DO IN = 1 ,18                               !Any non-zero value indicates
        IF (RCONST(IN) .NE. 0.)  UPPERS = .TRUE.  !constants are being passed
        END DO                                      !for upper instruments too.
!
!       Set 'channel' limits accordingly.
!
        IF (UPPERS) THEN
          IS = 1                     !all six instrument present
          IE = 6
        ELSE
          IS = 4                     !only lower instruments fitted
          IE = 6
        ENDIF

!                       Put RCONST values into program variables.)
        DO IN = IS,IE
          TSA(IN)    = RCONST((IN-1)*6 +1) !Temperature sensitivity coefficents
          TSB(IN)    = RCONST((IN-1)*6 +2) ! Alpha, Beta, Gamma
          TSG(IN)    = RCONST((IN-1)*6 +3) !
          PITOFF(IN) = RCONST((IN-1)*6 +4) !Pitch offset of instrument
          ROLOFF(IN) = RCONST((IN-1)*6 +5) !Pitch offset of instrument

!         Check whether the configuration has been modified by examining the
!         last constant for each instrument (=IOBTYP). If it is >10 an offset
!         has been added to it; identify this and restore correct constant.
!
          RTEMP = RCONST((IN-1)*6 +6)             !Get obscurer value (+offset?)
          IF (ABS(RTEMP) .GE. 10.0) THEN          !An offset has been added.
              RTEMP      = RTEMP/10.              !Bring the offset into the
              ICONF(IN)  = INT(RTEMP)             !truncate range |1 - 6|>ICONF()
              ROBTYP     = (RTEMP-ICONF(IN))*10. !Restore the Obscurer const.
              ICONF(IN)  = IABS(ICONF(IN))        !Config indicator must be +ve.
              IOBTYP(IN) = NINT(ROBTYP)           !assign Obscurer type in use
                                                  !(1: none, 2: short, 3: tall)

          ELSE                                    !use default ICONF values
              IOBTYP(IN) = NINT(RTEMP)            !Obscurer type in use
          ENDIF

        END DO                            !next instrument.
      ENDIF                               !of First-time-through actions.
!-

!+
```

(continues on next page)

(continued from previous page)

```
!  2. Derive/convert any intermediate results used several times
!     within code sections following.
!
!     Put input data into arrays.

      IF (UPPERS) THEN
        DO IN = 1,3                               !Upper instruments
          RFLX (IN)  = RDER(1,673+IN -1)          !  Signal       w/m-2
          ZIN  (IN)  = RDER(1,676+IN -1)          !  zero         w/m-2
          RTHM (IN)  = RDER(1,679+IN -1)          !  thermistor   deg K
        END DO
      ENDIF

      DO IN = 1,3                                 !Lower instruments
        RFLX (IN+3) = RDER(1,682+IN -1)           !  Signal       w/m-2
        ZIN  (IN+3) = RDER(1,685+IN -1)           !  zero         w/m-2
        RTHM (IN+3) = RDER(1,688+IN -1)           !  thermistor   deg K
      END DO

      IROL=560
      IPIT=561
      IHDG=562
      if(ITSTFLG(RDER(1,616)).EQ.0)IROL=616
      if(ITSTFLG(RDER(1,617)).EQ.0)IPIT=617
      if(ITSTFLG(RDER(1,618)).EQ.0)IHDG=618

      HDGINS = CIRC_AVRG( RDER(1,IHDG), 32)        !Mean of INS Heading samples
                                                  !(special for circular values)
      SOLAZM        = RDER(1,642)                 !Solar azimuth angle
      SOLZEN        = RDER(1,643)                 !Solar zenith   "
!-

!+    set flags for corrections

      IFLAG_INHDG = ITSTFLG (HDGINS)              !Flag of INS   heading
      CALL ISETFLG(HDGINS,0)                      !Strip flag
      IFLAG_ZEN   = ITSTFLG (SOLZEN)              !Flag of solar zenith angle
      CALL ISETFLG(SOLZEN,0)                      !Strip flag
      IFLAG_AZM   = ITSTFLG (SOLAZM)              !Flag of solar azimuth angle
      CALL ISETFLG(SOLAZM,0)                      !Strip flag
      IFLAG_SHDG  = MAX(IFLAG_INHDG,IFLAG_AZM)    !Choose higher heading flag
!-

!+    Convert samples to radians measure.

      HDGRAD = HDGINS /DEG2RD             !Convert INS   heading to radians
      ZENRAD = SOLZEN/DEG2RD              !Convert Solar Zenith ang to radians
      AZMRAD = SOLAZM/DEG2RD              !Convert Solar Zenith ang to radians
      SUNHDG = AZMRAD - HDGRAD            !Sun Heading (Solar Az-A/C hdg (INS))
!-

      IF (SOLZEN .GT. 0. .AND. SOLZEN .LT.90.)THEN !Prevent exponentiation error
      FCRIT  = 920.*(COS(ZENRAD))**1.28  !Critical flux value (direct/diffuse)
      ENDIF

!+  3. Derive running mean of zero offsets for each instrument over ten seconds
```

(continues on next page)

(continued from previous page)

```
      DO I=IS,IE
      CALL S_RUNM(ZBUF(1,I),IZP(I),IZCNT(I),10,ZIN(I),ZSUM(I),ZBAR(I))
      END DO

!-

!+  4. means of 32hz INS PITCH & ROLL arguments for one second.


      CALL RMEANOF(32 ,RDER(1,IROL), ROLINS, IDUM) !Mean of INS Roll samples.
      CALL RMEANOF(32 ,RDER(1,IPIT), PITINS, IDUM) !Mean of  "  Pitch  "    .

!    then derive running mean of pitch and roll values. (meaned over two secs)

      CALL S_RUNM(RBUF,IRPT,IRCNT,2,ROLINS,RSUM,ROLBAR)          !Roll
      CALL S_RUNM(PBUF,IPPT,IPCNT,2,PITINS,PSUM,PITBAR)          !Pitch

!     Set Pitch flag, no acceptability test currently used.

      IFLAG_PIT = ITSTFLG (PITBAR)
      CALL ISETFLG(PITBAR,0)                    !Strip flag

!     Roll limit acceptable?

      IFLAG_ROLL= ITSTFLG (ROLBAR)              !Flag of meaned Roll.
      CALL ISETFLG(ROLBAR,0)                    !Strip flag
      IF ( ABS(ROLBAR) .GT. ROLLIM)             !Comparison in degrees
     -    IFLAG_ROLL= MAX(IFLAG_ROLL,1)         !Flag if Roll too great

!  5.  Correct thermistor values for linearity

      CALL CORR_THM (RTHM,THM)                  !Input temps deg K, output deg C

!-----------------------------------------------------------------------------

      DO IN = IS,IE                         !Cycle through available instruments

      FOBSC     = ROBSC(IOBTYP(IN),IN)      !select correction for obscurer
      IFLAG_CORRN  = 0                      !Set corrections flag to valid
      IFLAG_FLX = ITSTFLG (RFLX(IN))        !Flag of raw flux input
      CALL ISETFLG(RFLX(IN),0)              !Strip flag
      IFLAG_ZER = ITSTFLG (ZBAR(IN))        !Flag of meaned zero-offset
      CALL ISETFLG(ZBAR(IN),0)              !Strip flag
      IFLAG_THM = ITSTFLG (THM (IN))        !Flag of corrected thermistor.
      CALL ISETFLG(THM (IN),0)              !Strip flag

      IFLAG_SIGNAL= MAX(IFLAG_FLX,IFLAG_ZER) !Obtain worst of (flx,zero) flag.

      IF (IFLAG_SIGNAL .EQ. 3) THEN          !**** Check Flux validity
         FLX(IN) =  -99.                     !Set output to 'failed' value.
         IFLAG_OUTPUT=  3                    !'Failed' flag.

      !------------------------------------------------------------------
      ELSE                                   ! OK to begin correcting flux.

        FLX(IN) = RFLX(IN) - ZBAR(IN)       !Subtract meaned zero-offset.
```

(continues on next page)

(continued from previous page)

```
!       Perform temperature sensitivity correction.

          IF (IFLAG_THM .LT. 2) THEN                !Thermistor temperatures
            FL      = FLX(IN)                       !have been corrected and
            TH      = THM(IN)                       !converted  to C by CORR_THM.
            FLX(IN) = FL /
     -                (1.+ TH*(TSA(IN)
     -                   + TH*(TSB(IN)
     -                   + TH* TSG(IN) )))
          ENDIF

        !----------------------------------------------------------------------
        IF (ICONF(IN) .EQ. 3 .OR. ICONF(IN) .EQ. 6) THEN  !*** Pyrgeometers only
        !----------------------------------------------------------------------

!         Perform 'sigma* Tsink^4' correction

          IF (IFLAG_THM .LT. 2) THEN
            FL = FLX(IN)
            FLX(IN) =FL * (1.0/(1.0-FOBSC))+SIGMA*(TH+273.16)**4
          ENDIF
                                          !Correction to upper Pyrgeometer for
                                          !dome transmission of downwelling I/R.
          IF (ICONF(IN) .EQ. 3 )THEN
            FLX(IN) = FLX(IN) + (-6.0 + 0.0175* FLX(IN))!see Tech note 3. page 2
          ENDIF

          IFLAG_CORRN = IFLAG_THM                      !Relevant corrections
          IFLAG_OUTPUT = IFLAG_TABLE(IFLAG_SIGNAL,IFLAG_CORRN)

        !----------------------------------------------------------------------
        ELSE                      !Upper and Lower Pyranometer corrections
        !----------------------------------------------------------------------

          IF (ICONF(IN) .EQ. 4 .OR. ICONF(IN) .EQ. 5) THEN  !Lower pyranometers

             FLX(IN)= FLX(IN)*(1.0/(1.0- FOBSC))            !Obscurer corr'n.
                                                            !All corr'n complete
             IFLAG_CORRN = 0                                !no relevant corrs
             IFLAG_OUTPUT = IFLAG_TABLE(IFLAG_SIGNAL,IFLAG_CORRN)

          ELSE                                              !Upper Pyranometers

!+          Compare incoming flux with Fcrit (Critical value) of expected flux.
!           IF Flux > Fcrit; treat irradiation as being DIRECT.
!           ELSE             assume it is DIFFUSE irradiation.
!             (n.b. for RED dome, Fcrit value used is 1/2 normal Fcrit.)

            FCRITVAL = FCRIT
            IF( ICONF(1)  .NE.  1) FCRITVAL = FCRIT * .5    !1/2 For RED dome.

            IF (FLX(1)  .GT. FCRITVAL) THEN                 !*Direct or Diffuse?
!-

!+            DIRECT is appropriate; check angle between Sun & normal-to-
!             instrument is not > 80 deg, before correction for platform level.
```

(continues on next page)

(continued from previous page)

```
              PITCH=PITBAR + PITOFF(IN) !Combine  A/C mean and Inst offset Pitch
              PITCH=PITCH/DEG2RD        !.. and convert to radians
              ROLL =ROLBAR + ROLOFF(IN) !Combine A/C  mean and Inst offset Roll
              ROLL = ROLL/DEG2RD        !.. and convert to radians

!             Find angle between Solar zenith and normal-to-Instrument.
                                                    !Ref:Tech note 7 Page 10
                                                    !Derive cosine of angle.
              RCOSTH = SIN(ROLL)*SIN(ZENRAD)*SIN(SUNHDG)
     &              +  COS(ROLL)*COS(PITCH) *COS(ZENRAD)
     &              -  COS(ROLL)*SIN(PITCH) *SIN(ZENRAD)*COS(SUNHDG)
              THETA = ACOS(RCOSTH)            !Express angle in radians

!             Compare with maximum allowable angle. ( must be < 80 Deg)

              IF (THETA .GT. THTMAX/DEG2RD) THEN
                IFLAG_ANG =  2                !Failed Low sun test; Flag value
              ELSE
                IFLAG_ANG =  0                !Angle Sun/Instr acceptable.
              ENDIF

!             Apply levelling correction using combined pitch and roll, if
!             necessary conditions are met:-

              IFLAG_CORRN = MAX (IFLAG_PIT,  IFLAG_ROLL)  !A/c  Attitude flags.
              IFLAG_CORRN = MAX (IFLAG_CORRN,IFLAG_ANG)
              IFLAG_SUN   = MAX (IFLAG_SHDG ,IFLAG_ZEN)
              IFLAG_CORRN = MAX (IFLAG_CORRN,IFLAG_SUN)

              IFLAG_OUTPUT = IFLAG_TABLE(IFLAG_SIGNAL,IFLAG_CORRN)

              IF ( IFLAG_CORRN .LT. 2 .AND. RCOSTH .NE.0.) THEN

! *OLD VERSION* FLX(IN) = FLX(IN) * (COS(ZENRAD)/RCOSTH)   !levelling correction

!            Correct the flux for attitude of aircraft for direct component of
!            beam. Also include COSINE effect correction.    (Ref: M/MRF/13/5)

                INDX = NINT(SOLZEN/10) + 1
                INDX = MIN (INDX,10)

                FLX (IN) =              FLX(IN)/
!                         --------------------------------------------
     &            (1.- FDIR(INDX)*(1.- CEFF(INDX)*(RCOSTH/COS(ZENRAD))))
              ENDIF

            ELSE                            !* Critical value, (flux less than.)
                                            !  Diffuse case;   make Obscurer
                                            !  correction if signal is valid.

              IFLAG_CORRN  = MAX(IFLAG_PIT,  IFLAG_ROLL)
              IFLAG_CORRN  = MAX(IFLAG_CORRN,IFLAG_ZEN)
              IFLAG_OUTPUT = IFLAG_TABLE (IFLAG_SIGNAL,IFLAG_CORRN)
              FLX(IN) = FLX(IN)*(1.0/(1.0- FOBSC))

            ENDIF                           !* Critical value for direct?
```

(continues on next page)

(continued from previous page)

```
            IF ( IFLAG_SIGNAL .EQ. 3) THEN
              FLX(IN) = -99.                      !set  invalid flux to obvious
            ENDIF                                 !known value.

          ENDIF                              !**  Upper or Lower pyranometers?
        ENDIF                                !*** pyranometer or pyrgeometer?

!       Perform range checks on valid output fluxes.

        IF (IFLAG_OUTPUT .LT. 3 ) THEN
            IF (FLX(IN) .GT. RMAXFLX(ICONF(IN)) .OR.
     -          FLX(IN) .LT. RMINFLX(ICONF(IN))     ) THEN
               IFLAG_OUTPUT = 2              !Failed, flag result as 'suspect'
            ENDIF
        ENDIF
      ENDIF                                  !**** Flux signal validity?

!     Assign processed flux to output parameter

      RDER(1,1018 + IN) = FLX(IN)                        !Fill output argument
      CALL ISETFLG (RDER(1,1018 + IN), IFLAG_OUTPUT)   !Set output flag

      IFLAG_CORRN       = 0
      IFLAG_SIGNAL      = 0
      IFLAG_OUTPUT      = 0
      END DO                                           !(..Control value IN)

      RETURN
      END


C-----------------------------------------------------------------------------
C ROUTINE          CORR_THM     SUBROUTINE     FORTVAX            [C_RFLUX.FOR]
C
C PURPOSE          Correct thermistors for non-linearity using a quintic eqn.
C
C DESCRIPTION      The thermistors used in the pyrgeometer/pyranometers all
C                  have characteristic non-linear temperature dependence
C                  due to the manufacturing process. If not corrected for,
C                  this can lead to errors in temperature of up to 1 deg C.
C                  The thermistor manufacturers provide a curve of the the
C                  correction needed to be applied for a range of
C                  temperatures.  A quintic equation has been fitted to this
C                  curve to give the best fit coefficients used by this routine.
C
C METHOD           The routine takes an array of six thermistor values in deg K.
C                  In turn; notes each ones flag then clears the flag.
C                  Fits -50 deg C to +40 deg C to within +/- .07 deg C.
C                  Eqn: RT + (RCON +V.RT +W.RT^2 +X.RT^3 +Y.RT^4 +Z.RT^5)
C                  where RT  : Raw thermistor value  (converted to Celsius)
C                        RCON: A constant
C                   V,W,X,Y,Z: Coefficients of quintic equation correcting temp.
C
C                  Loop through six thermistor values:
C                  a)   note each one's flag
C                  b)   if flag indicates input is valid (flag <3)
```

(continues on next page)

(continued from previous page)

```
C                       - clear the flag bits from the raw thermistor value
C                       - assign the value (converted to deg C.) to a working
C                         variable,  which becomes the input variable to a the
C                         quintic equation above.
C                       - derive the corrected output using that equation.
C                       - set input flag value in output thermistor temperature.
C                       else; for an 'invalid' flag
C                       - set the output thermistor value to zero C
C                       - set its output's flag to 3 (= invalid)
C                  next loop.
C
C                  n.b. The corrected thermistor values are not saved at the
C                       end of calibration and are only calculated for local
C                       use in deriving corrected solar fluxes.
C
C VERSION          1.02  30-07-91  A.D HENNINGS
C
C REFERENCES       Best-fit coefficients and constants taken from fitting to
C                  manufacturers calibration data sheet.
C
C ARGUMENTS        REAL*4 RTHM(6)  IN   Six uncorrected thermistor values. deg K
C                  REAL*4 THM (6) OUT   Six  corrected thermistor values.  deg C
C
C SUBPROGRAM       ITSTFLG ISETFLG
C
C CHANGES          1.01 201190 Documentation.
C                  1.02 300791 Documentation.
C                  1.03 17-01-96 D Lauchlan
C                  Unused variables removed
C                  1.04 22-03-04 D Tiddeman flag stripping before calculation
C                  changed to prevent crashes.
C-------------------------------------------------------------------------------
      SUBROUTINE CORR_THM (RTHM,THM)
CDEC$ IDENT 'V1.04'
C
      IMPLICIT NONE
      REAL*8 V,W,X,Y,Z,               !Coefficients of powers 1, 2, 3, 4 & 5
     -       RT,RCON                  !placeholder  for thermistor for calc.
      REAL*4 RTHM(6),THM(6)           !Raw Thermistor, corrected thermistor.
      INTEGER*4 I,IFLAG ,ITSTFLG
c      LOGICAL OFIRST_TIME/.TRUE./  !   "      "
      PARAMETER (RCON = -0.774,
     -              V =  6.08E-02,
     -              W =  2.47E-03,
     -              X = -6.29E-05,
     -              Y = -8.78E-07,
     -              Z =  1.37E-08)
!

      DO  I=1,6
      IFLAG = ITSTFLG(RTHM(I))
      CALL ISETFLG(RTHM(I),0)                      !Clear flag before calc.
      IF (IFLAG .LT. 3) THEN
        RT     = RTHM(I) - 273.16                  !convert to Celsius
        THM(I) = RT + (RCON + RT*(V+ RT*(W+RT*(X+RT*(Y+RT*Z)))))
        CALL ISETFLG(THM(I),IFLAG)                 !Replace original flag.
      ELSE
```

(continues on next page)

(continued from previous page)

```
        THM(I) = 0.0                               !Set thermistors to failed.
        CALL ISETFLG(THM(I),3)                     !and flag as such
      ENDIF
      END DO

      RETURN
      END


C--------------------------------------------------------------------------------
C ROUTINE          RMEANOF        SUBROUTINE     FORTVAX         [C_RFLUX.FOR]
C
C PURPOSE          Calculate the mean of an array of real values.
C
C DESCRIPTION      An array containing NOELS  real elements is received.
C                  Each element is checked and, if it has a Flag value
C                  (bits 16+17) of zero, is accumulated to a total, and
C                  the  count of good elements incremented.
C                  When all elements have been checked, the mean is derived
C                  such that:
C                  If no good elements were found, the mean is zero, flagged 3.
C                  Otherwise, the mean is the total/count, flagged 0.
C
C ARGUMENTS        INTEGER*4  NOELS IN   Number of elements in array passed
C                  REAL*4     RARR  IN   Array of reals - dimensioned to NOELS
C                  REAL*4     RMEAN OUT  Arithmetic mean of good samples, or 0.
C                  INTEGER*4  IFLAG OUT  Flag value of mean, 0:good 3:invalid.
C
C VERSION          1.00   19-03-90  A.D.HENNINGS
C
C SUBPROGRAMS      ITSTFLG  ISETFLG
C
C REFERENCES       None
C
C------------------------------------------------------------------------------

      SUBROUTINE RMEANOF(NOELS,RARR,RMEAN,IFLAG)
CDEC$ IDENT 'V1.00'
C
      IMPLICIT NONE
      INTEGER*4 NOELS,IX,ITSTFLG,ICOUNT,IFLAG
      REAL*4    RARR(NOELS),RMEAN,SUMM

      SUMM = 0.
      ICOUNT   = 0
      DO IX= 1,NOELS
      IF (ITSTFLG(RARR(IX)) .EQ. 0) THEN
        SUMM = SUMM + RARR(IX)
        ICOUNT = ICOUNT+1
      ENDIF
      END DO

      IF (ICOUNT .GT. 0 )THEN
        RMEAN = SUMM/FLOAT(ICOUNT)
        IFLAG = 0
      ELSE
        RMEAN  = 0.
```

(continues on next page)

(continued from previous page)

```
         IFLAG  = 3
      ENDIF
      CALL ISETFLG(RMEAN,IFLAG)

      RETURN
      END
*---------------------------------------------------------------
C ROUTINE          CIRC_AVRG  FUNCTION   FORTVAX
C
C PURPOSE          CALCULATE MEAN OF A SET (>2 <1000)  OF ANGLES, IN DEG.
C
C ARGUMENTS        REAL*4    ARR  IN         Array of Angles (in  Degrees)
C                  INTEGER*4 NUM  IN         Number of angle in array ARR.
C                  REAL*4    CIRC_AVANG OUT  Average angle of set (0-360 deg)
C
C DESCRIPTION      Given a set of angles (0-360 Deg) calculates their mean.
C                  Handles values spanning 0 or 180.
C                  Returns mean    Flagged 0: If >2 and <= 1/2 of inputs valid
C                                          1: If < 1/2 of inputs valid.
C                                          3: If no valid inputs.
C                  N.B  ASSUMES ALL INPUT ANGLES ARE BETWEEN 0 & 360 DEG.
C
C VERSION          1.0   JAN 1992  A D HENNINGS
C                        MODIFIED FROM  "AVANG" V3.0 SEP 1984  D OFFILER
C                  1.01  DEC 1997  W D N JACKSON
C                        Stips flags before using data
C-------------------------------------------------------------------------------
      REAL  FUNCTION CIRC_AVRG( ARR , NUM)
CDEC$ IDENT 'V1.00'

      IMPLICIT NONE
      INTEGER NUM,NM1,I,ITSTFLG,ICOUNT,IFLAG
      REAL ARR(NUM)
      REAL TARR(1000),DIF


      DO I=1,NUM
      TARR (I) = ARR(I)                           !Move values to temporary array
      CALL ISETFLG(TARR(I),0)                     !Strip flag
      END DO                                      !as they may be altered later.

C Alter angles to same sign .

      IF ( NUM .GT. 2 ) THEN
         NM1 = NUM - 1
         DO I = 1 , NM1
            DIF = TARR(I) - TARR(I+1)
            IF ( ABS ( DIF ) .GT. 180.0 ) THEN
                     TARR(I+1) = TARR(I+1) + SIGN (360.0 , DIF )
            ENDIF
         ENDDO
      ENDIF

C  Sum the good points.

      CIRC_AVRG= 0.0
      ICOUNT= 0
```

(continues on next page)

(continued from previous page)

```
      DO I = 1 , NUM
         IF (ITSTFLG (ARR(I)) .LE. 1) THEN          !Do check on original array
            CIRC_AVRG = CIRC_AVRG + TARR(I)         !..but use changed data
            ICOUNT =ICOUNT+1
         ENDIF
      ENDDO

C Calculate average.

      IF (ICOUNT .GT. 0 )THEN
          CIRC_AVRG = CIRC_AVRG / FLOAT (ICOUNT )
          IF (ICOUNT .GT. NUM/2 ) THEN          !More than half rejected, then
              IFLAG = 0                         !flag as reduced quality data.
          ELSE
              IFLAG = 1
          ENDIF
      ELSE
          CIRC_AVRG  = 0.
          IFLAG  = 3
      ENDIF

      IF ( CIRC_AVRG .LT.   0.0 ) CIRC_AVRG = CIRC_AVRG + 360.0
      IF ( CIRC_AVRG .GE. 360.0 ) CIRC_AVRG = CIRC_AVRG - 360.0

C Set the flag in the returned value

      CALL ISETFLG(CIRC_AVRG,IFLAG)

      END
```

8.6 c_gsun.for

```
C
C ROUTINE           C_SUN         SUBROUTINE FORTVAX  C_SUN.FOR
C
C PURPOSE           PUT SOLAR ZENITH AND AZIMUTH ANGLES IN MFD
C
C DESCRIPTION       Given date, time and location on the earth's
C                   surface this routine puts a solar zenith and
C                   azimuth angle in the array of derived parameters.
C                   It computes a value once every second. The
C                   angles are only obtained if all the flags are
C                   set to less than 3 and the date, time and location
C                   are all within sensible limits. Any flags set on input
C                   are also set in the solar angles derived. If
C                   the input is in error or the flags are set to 3
C                   a value of -99. is returned for ZEN and AZIM.
C                   To test the routine:
C                   $ FOR C_SUN
C                   $ FOR TEST_C_SUN
C                   $ LINK TEST_C_SUN,C_SUN
C                   Ensure contents of files RCONST.DAT and TEST_C_SUN.DAT
C                   contain simulated data you require to test the routine
```

(continues on next page)

(continued from previous page)

```
C                  with.
C
C VERSION          1.02  1st May 1992   J.A.Smith
C
C ARGUMENTS        RDER(1,515)  R*4 IN Time GMT (seconds from midnight)
C                  RDER(1,550)  R*4 IN Omega latitude degrees (north +ve)
C                  RDER(1,551)  R*4 IN Omega longitude degrees (east +ve)
C               or RDER(1,541)  R*4 IN INU latitude degrees (north +ve)
C               or RDER(1,542)  R*4 IN INU longitude degrees (east +ve)
C                  RCONST(1)    R*4 IN Day in month (1-31)
C                  RCONST(2)    R*4 IN Month in year (1-12)
C                  RCONST(3)    R*4 IN Year (eg 1984)
C                  RDER(1,642)  R*4 OUT Solar azimuth in degrees
C                  RDER(1,643)  R*4 OUT Solar zenith in degrees
C
C SUBPROGRAMS      S_SUN , ITSTFLG, ISETFLG
C
C CHANGES          01 Range checks for input data now done in S_SUN
C                     RWS 30/10/90
C                1.02 Check added if time RSECS has reached midnight and
C                     if so to reduce RSECS to less than 86400 s and increase
C                     the date.  JAS 1/5/92
C                1.03 Following the demise of the Omega, now uses INU position
C                     for flights after 30/09/97.  Note that this routine is
C                     now always called by CALIBRATE, even if neither Omega or
C                     INU were available.  WDNJ 20/10/97
C                1.04 Now strips flags from data before use.  WDNJ 22/12/97
C                1.05 Can take GIN input 05/09/07
C                1.06 Changes made how lon/lat input is derived AxW 29/03/10
C#######################################################################
      SUBROUTINE C_GSUN ( IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.05'
C
      INTEGER*4 IRAW(64,512), IFRQ(512), IFLAG(6)
      INTEGER*4 DAYM(12)/31,29,31,30,31,30,31,31,30,31,30,31/
      INTEGER*4 IMIDNIGHTS           ! added for v1.02
      REAL*4    RCONST(64), RDER(64,1024)
      LOGICAL   BAD_INPUT
C
      RSECS = RDER(1,515)            ! Seconds elapsed since midnight GMT
      IDAY = INT(RCONST(1))          ! Date in month
      IMON = INT(RCONST(2))          ! Month in Year
      IYR = INT(RCONST(3))           ! Year
!      IF((IYR.EQ.1997.AND.IMON.GE.10).OR.IYR.GT.1997) THEN
!        IF(ITSTFLG(RDER(1,541)).EQ.0)RLAT = RDER(1,541)           ! INU latitude
!        IF(ITSTFLG(RDER(1,542)).EQ.0)RLON = RDER(1,542)           ! INU longitude
!        IF(ITSTFLG(RDER(1,610)).EQ.0)RLAT=RDER(1,610)
!        IF(ITSTFLG(RDER(1,611)).EQ.0)RLON=RDER(1,611)
!        print *,ITSTFLG(RDER(1,610)),ITSTFLG(RDER(1,541))
!      ELSE
!        RLAT = RDER(1,550)           ! Omega latitude
!        RLON = RDER(1,551)           ! Omega longitude
!      END IF

!Changed on 31/03/2010 after suggestion from Dave Tiddeman
      IF((IYR.EQ.1997.AND.IMON.GE.10).OR.IYR.GT.1997) THEN
        RLAT = RDER(1,541)           ! INU latitude
```

(continues on next page)

(continued from previous page)

```
        RLON = RDER(1,542)           ! INU longitude
          IF(ITSTFLG(RDER(1,610)).LT.3)RLAT=RDER(1,610) !GIN latitude
          IF(ITSTFLG(RDER(1,611)).LT.3)RLON=RDER(1,611) !GIN longitude
      ELSE
        RLAT = RDER(1,550)           ! Omega latitude
        RLON = RDER(1,551)           ! Omega longitude
      END IF
C
      BAD_INPUT = .FALSE.
C
C   Check flags and only proceed if all less than 3
C
      DO I = 1 , 3
      IFLAG(I) = ITSTFLG(RCONST(I))
      ENDDO
      IFLAG(4) = ITSTFLG(RSECS)
      IFLAG(5) = ITSTFLG(RLAT)
      IFLAG(6) = ITSTFLG(RLON)
      CALL ISETFLG(RSECS,0)
      CALL ISETFLG(RLAT,0)
      CALL ISETFLG(RLON,0)
C
      IMAXFL = 0
      DO I = 1 , 6
      IMAXFL = MAX ( IMAXFL , IFLAG(I) )       ! Get highest flag value
      IF (IFLAG(I) .GE. 3)THEN
        BAD_INPUT = .TRUE.
        CALL ISETFLG ( AZIM , 3 )             ! Set invalid data flags
        CALL ISETFLG ( ZEN  , 3 )
      ENDIF
      ENDDO
C
C If input parameters OK proceed
C
      IF ( .NOT. BAD_INPUT )THEN
C.......................................................................
C v1.02 If time has run over midnight reduce RSECS to less than 24 hours of
C        seconds, ( 86400 ). The day of the month IDAY is then increased by
C        the number of midnights passed over.
C        If this gives too many days for the month then IDAY is set to the
C        first day and IMON to the next month.
C        If the data has crossed into a New Year then IMON is set to January
C        and the year is incremented.
C
      IF (RSECS.GE.86400.) THEN
          IMIDNIGHTS = (NINT(RSECS))/86400
          RSECS = RSECS - REAL(IMIDNIGHTS*86400)
          IDAY = IDAY + IMIDNIGHTS
          IF (MOD(IYR,4).NE.0) DAYM(2)=28  !reduce February if not a leap year
          IF (IDAY.GT.DAYM(IMON)) THEN
              IDAY = IDAY - DAYM(IMON)
              IMON = IMON + 1
              IF (IMON.EQ.13) THEN
                  IMON = 1
                  IYR = IYR + 1
              ENDIF
          ENDIF
```

(continues on next page)

(continued from previous page)

```
      ENDIF
C.........................................................................
C
C  Now compute solar zenith and azimuth angle
C
      CALL S_SUN(IDAY,IMON,IYR,RSECS,RLAT,RLON,AZIM,ZEN)

C
C  Flag values with highest input flag value
C
C  If azimuth or zenith angle not computed in S_SUN set flags to 3
C
      IF (AZIM.EQ.-99) THEN
        CALL ISETFLG(AZIM,3)
      ELSE
        CALL ISETFLG(AZIM,IMAXFL)
      ENDIF

      IF (ZEN.EQ.-99) THEN
        CALL ISETFLG(ZEN,3)
      ELSE
        CALL ISETFLG(ZEN,IMAXFL)
      ENDIF
C
      ELSE
      BAD_INPUT = .TRUE.
      AZIM = -99.0
      ZEN = -99.0
      CALL ISETFLG ( AZIM , 3 )              ! Set invalid data flags
      CALL ISETFLG ( ZEN  , 3 )
C
      ENDIF
C
C  Transfer to output array
C
      RDER(1,642) = AZIM
      RDER(1,643) = ZEN
C
      RETURN
      END
```

8.7 c_gwinds.for

```
C
C ROUTINE           C_GWINDS SUBROUTINE FORTVAX
C
C PURPOSE           Computes raw winds from TAS, vanes, and INS data
C
C DESCRIPTION       Computes values of the three wind components, using true
C                   airspeed, angle of attack and sideslip, and INS velocity,
C                   attitude, and attitude rate information. Note that at this
C                   stage the INS data have not been corrected for drift, so
C                   these are 'raw' winds, which will normally be corrected
C                   later as part of the interactive renavigation processing.
```

(continues on next page)

(continued from previous page)

```
C                 Once errors have been evaluated for the three INS velocity
C                 components, they can be applied directly to the three wind
C                 components; the wind components do not need to be recomputed
C                 from scratch.  To show that the winds are 'raw' all values
C                 of U, V and W are increased by 1000 m/s by this routine.
C                 This makes it easy to see that normal (flagged 0 or 1) data
C                 are 'raw', but it may not be enough to say unabiguously
C                 whether data that are already bad (flagged 2 or 3) are 'raw'
C                 or 'corrected'.
C
C                 The processing will handle the case that the INS is mounted
C                 off the boom axis, provided its position is specified in
C                 the flight constants file, using the INSPOSN keyword.  If
C                 the INS position is not specified then it is assumed to be
C                 in the nose bay, 7.06m behind the vanes, but on the axis of
C                 the boom.  All data is assumed to be at 32 Hz.
C
C                 This routine will not be called if there is no True
C                 Airspeed, or no INS information (with the exception of roll
C                 rate).  If there is no information from the angle of attack
C                 and sideslip vanes, winds will be computed using values of
C                 zero for these angles flagged with
C                 1's.  If there is no roll rate available (this wasn't
C                 recorded for the Ferranti 1012 INS), a value of 0 is used.
C                 This doesn't matter if the INS is located on the boom axis,
C                 since in this case roll rate has no effect on winds.
C
C                 The output vertical wind takes the worst flag present on the
C                 AOA, VZ, TAS and pitch data.  The output horizontal wind
C                 components take the worst flag present on the AOSS, VN, VE,
C                 TAS, and heading data.  This is suitable when the
C                 aircraft is not banking and reflects the fact that good
C                 horizontal winds can be found even when the vertical
C                 velocity is bad.  However this flagging scheme fails to
C                 reflect coupling between the vertical and horizontal
C                 measurement when the aircraft is banking.
C                 In addition horizontal wind components greater
C                 than 100 m/s and vertical components greater than 25 m/s
C                 are flagged with 2's, and if the change between adjacent
C                 samples (at 32 Hz) is greater than 1 m/s a flag of 2 is
C                 also applied.
C
C                 Input parameters (all at 32 Hz except 515):
C
C                 Para 515   Time, secs
C                 Para 779   Turb.probe dry true airspeed, m s-1
C                 Para 548   Angle of attack, deg
C                 Para 549   Angle of side slip, deg
C                 Para 558   INS velocity north, m s-1
C                 Para 559   INS velocity east, m s-1
C                 Para 557   INS vertical velocity, m s-1
C                 Para 560   INS roll, deg
C                 Para 561   INS pitch, deg
C                 Para 562   INS heading, deg
C                 Para 567   INS roll rate, deg s-1 (optional)
C                 Para 565   INS pitch rate, deg s-1
C                 Para 566   INS yaw rate, deg s-1
```

(continues on next page)

(continued from previous page)

```
C
C                   Constants:
C
C                   RCONST(1)  Distance of vanes ahead of INS, m (optional)
C                   RCONST(2)  Distance of vanes to port of INS, m (optional)
C                   RCONST(3)  Distance of vanes above INS, m (optional)
C
C                   Output parameters (all at 32 Hz):
C
C                   Para 714   Northward wind component + 1000, m s-1
C                   Para 715   Eastward wind component + 1000, m s-1
C                   Para 716   Vertical wind component + 1000, m s-1
C
C VERSION           1.00  10-5-93  W.D.N.JACKSON
C
C ARGUMENTS         IRAW(64,512) I*4 IN  Up to 64 samples for up to 512 DRS pars
C                   IFRQ(512)    I*4 IN  Sample rate of each DRS par (0-64)
C                   RCONST(64)   R*4 IN  Inputs constants
C                   RDER(64,1024)R*4 OUT Output array of up to 64 samples for
C                                        each of 1024 parameters
C
C CHANGES           1.01  20-04-98 W.D.N.JACKSON
C                   Error in computation of airspeed corrected.
C                   1.02  14-06-2004 Phil Brown
C                   AoA and AoSS now compulsory input parameters to ensure
C                   this routine gets called after C_TURB
C                   1.03  09/07/04 Phil Brown
C                   Input TAS parameter is now 779 (Turb.probe dry TAS)
C                   1.04  25/08/04 Phil Brown
C                   Temporary. Suspend rate-of-change checking on winds.
C                   1.05  29/11/04 Phil Brown
C                   Temporary. Check flagging of RU,RV,RW when returned to try
C                   to suppress FLTINV errors.
C                   1.06  05/09/07 Dave Tiddeman
C                   Will use GIN inputs if available rather than INU
C
*******************************************************************************
      SUBROUTINE C_GWINDS(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.04'
      INTEGER*4 IRAW(64,512)      !Raw data array
      INTEGER*4 IFRQ(512)         !Raw data frequency
      REAL*4    RCONST(64)        !Constants array
      REAL*4    RDER(64,1024)     !Derived data array
      INTEGER*4 VN,VE,VZ,ROL,PIT,HDG,ROLR,PITR,YAWR
C
C This routine uses the following parameters (note that the absence of AOA,
C AOSS or roll rate will not stop C_WINDS from being called).  All parameters,
C except time, are at 32 Hz:
C
      PARAMETER GMT=515                !Time, secs
      PARAMETER TAS=779                !True airspeed, m s-1
      PARAMETER AOA=548                !Angle of attack, deg
      PARAMETER AOS=549                !Angle of side slip, deg
C
C This routine takes three constants from the RCONST array.  They are
C all optional and if not specified will be defaulted to the position of the
C H423 INU on the 146 Core Console (16.002,-0.8128,-0.4390 m).
```

(continues on next page)

(continued from previous page)

```
C
      PARAMETER PL=1                     !Const dist of vanes ahead of INS
      PARAMETER PM=2                     !Const dist of vanes to port of INS
      PARAMETER PN=3                     !Const dist of vanes above INS
C
C This routine computes the following parameters, all at 32 Hz:
C Note that TARDIS conventially labels parameter 714, Northerly component, as V
C and parameter 715, Easterly component, as U.
C
      PARAMETER U=714                    !Northward wind component, m s-1
      PARAMETER V=715                    !Eastward wind component, m s-1
      PARAMETER W=716                    !Vertical wind component, m s-1
C
C Set LFLAG to false if you want to treat all data as unflagged.
C
      DATA LFLAG   /.TRUE./             !Set false if want to ignore flagging

      DATA RLSTSEC /-2.0/                !Initial dummy value for last sec processed

      DATA VN/558/
      DATA VE/559/
      DATA VZ/557/
      DATA ROL/560/
      DATA PIT/561/
      DATA HDG/562/
      DATA ROLR/567/
      DATA PITR/565/
      DATA YAWR/566/


      SAVE
      IF(ITSTFLG(RDER(1,613)).EQ.0)THEN
        VN=613                   !GIN velocity north, m s-1
        VE=614                   !GIN velocity east, m s-1
        VZ=615                   !GIN vertical velocity, m s-1
        ROL=616                  !GIN roll, deg
        PIT=617                  !GIN pitch, deg
        HDG=618                  !GIN heading, deg
        ROLR=622                 !GIN roll rate, deg s-1 (optional)
        PITR=623                 !GIN pitch rate, deg s-1
        YAWR=624                 !GIN yaw rate, deg s-1
      ENDIF

      RDEFAOA=0.0                        !If not specified AOA is 0.0 flagged 1
      CALL ISETFLG(RDEFAOA,1)
      RDEFAOS=RDEFAOA                    !If not specified AOSS is 0.0 flagged 1

      IF(.NOT.LFLAG) THEN                !Ignore flagging
        DO I=1,32                        !For each sample in second
          CALL C_WINDS_UVW(RDER(I,TAS),RDER(I,AOA),RDER(I,AOS),
     -          RDER(I,VN),RDER(I,VE),RDER(I,VZ),
     -          RDER(I,HDG),RDER(I,PIT),RDER(I,ROL),
     -          RCONST(PL),RCONST(PM),RCONST(PN),
     -          RDER(I,YAWR),RDER(I,PITR),RDER(I,ROLR),
     -          RDER(I,U),RDER(I,V),RDER(I,W))
        END DO
      ELSE                               !Apply flags
```

(continues on next page)

(continued from previous page)

```
        RL=RCONST(PL)                   !Get the INS position offsets
        RM=RCONST(PM)
        RN=RCONST(PN)
        IF(ITSTFLG(RL).GE.2) RL=16.002 !Use default values if not available
        IF(ITSTFLG(RM).GE.2) RM=-0.8128
        IF(ITSTFLG(RN).GE.2) RN=-0.4390
        LCONSEQ=.FALSE.                 !Will set true if this is next second
        IF(RDER(1,GMT).EQ.RLSTSEC+1.0) LCONSEQ=.TRUE.
        RLSTSEC=RDER(1,GMT)             !Save current time

        DO I=1,32                       !For each sample in second
          RTAS=RDER(I,TAS)              !Get the input values
          RAOA=RDER(I,AOA)
          RAOS=RDER(I,AOS)
          RVN=RDER(I,VN)
          RVE=RDER(I,VE)
          RVZ=RDER(I,VZ)
          RHDG=RDER(I,HDG)
          RPIT=RDER(I,PIT)
          RROL=RDER(I,ROL)
          RYAWR=RDER(I,YAWR)
          RPITR=RDER(I,PITR)
          RROLR=RDER(I,ROLR)
          IF(ITSTFLG(RAOA).GE.2) RAOA=RDEFAOA !Set AOA to 0 if missing
          IF(ITSTFLG(RAOS).GE.2) RAOS=RDEFAOS !Set AOSS to 0 if missing
          IF(ITSTFLG(RROLR).GE.2) RROLR=0.0   !Set roll rate to 0 if missing
          IHFLAG=MAX(ITSTFLG(RTAS),ITSTFLG(RAOS), !Compute worst horiz flag
     -        ITSTFLG(RVN),ITSTFLG(RVE),ITSTFLG(RHDG))
          IWFLAG=MAX(ITSTFLG(RTAS),ITSTFLG(RAOA), !Compute worst vert flag
     -        ITSTFLG(RVZ),ITSTFLG(RPIT))
          CALL ISETFLG(RTAS,0)          !Clear any flags before computation
          CALL ISETFLG(RAOA,0)
          CALL ISETFLG(RAOS,0)
          CALL ISETFLG(RVN,0)
          CALL ISETFLG(RVE,0)
          CALL ISETFLG(RVZ,0)
          IF(VN.NE.558)RVZ=-RVZ
          CALL ISETFLG(RHDG,0)
          CALL ISETFLG(RPIT,0)
          CALL ISETFLG(RROL,0)
          CALL ISETFLG(RYAWR,0)
          CALL ISETFLG(RPITR,0)
          CALL ISETFLG(RROLR,0)
          CALL C_WINDS_UVW(RTAS,RAOA,RAOS,RVN,RVE,RVZ,RHDG,RPIT,RROL,
     -        RL,RM,RN,RYAWR,RPITR,RROLR,RU,RV,RW) !Compute wind components
          IUFLAG=IHFLAG                 !Propagate worst case flag for each comp
          IVFLAG=IHFLAG
          IF(ABS(RU).GT.100.0) IUFLAG=MAX(IUFLAG,2) !Flag if out of range
          IF(ABS(RV).GT.100.0) IVFLAG=MAX(IVFLAG,2)
          IF(ABS(RW).GT.25.0) IWFLAG=MAX(IWFLAG,2)
          CALL ISETFLG(RU, 0)           ! ensure winds have zero flag
          CALL ISETFLG(RV, 0)
          CALL ISETFLG(RW, 0)
          IF(VN.EQ.558)THEN
           RU=RU+1000.                   !Add offset to show winds are 'raw'
           RV=RV+1000.
           RW=RW+1000.
```

(continues on next page)

(continued from previous page)

```
          ENDIF

C suspend rate-of-change checks.
C          IF(ITSTFLG(RLSTU).EQ.0.AND.LCONSEQ.AND.ABS(RLSTU-RU).GT.1.0)
C     -        IUFLAG=MAX(IUFLAG,2)      !Flag if rate of change too high
C          IF(ITSTFLG(RLSTV).EQ.0.AND.LCONSEQ.AND.ABS(RLSTV-RV).GT.1.0)
C     -        IVFLAG=MAX(IVFLAG,2)
C          IF(ITSTFLG(RLSTW).EQ.0.AND.LCONSEQ.AND.ABS(RLSTW-RW).GT.1.0)
C     -        IWFLAG=MAX(IWFLAG,2)

          CALL ISETFLG(RU,IUFLAG)       !Apply flags to result
          CALL ISETFLG(RV,IVFLAG)
          CALL ISETFLG(RW,IWFLAG)
          RDER(I,U)=RU                  !Transfer results to output array
          RDER(I,V)=RV
          RDER(I,W)=RW
          RLSTU=RU                      !Save latest values
          RLSTV=RV
          RLSTW=RW
          LCONSEQ=.TRUE.                !Further samples in second are consequetve
        END DO
      END IF
      RETURN
      END
```

8.8 c_heiman.for

```
C-----------------------------------------------------------------------------
C
C ROUTINE          C_HEIMAN   SUBROUTINE FORTVAX
C
C PURPOSE          To derive uncorrected Heimann temperatures
C
C DESCRIPTION      Converts rawdata input from the Heimann radiometer and
C                  black body source into uncorrected surface tempratures,
C                  parameter 537.
C
C                  The Heimmann is recorded by para 141,
C                  the blackbody reference temperature by para 142,
C                  and bit 0 of the signal register (para 27) indicates whether
C                  the Heimann is set to calibrate.
C
C ARGUMENTS        IRAW   input raw data
C                  IFRQ   raw data frequencies
C                  RCONST flight constants corresponding to PRTCCAL and HEIMCAL
C                  RDER   output data
C
C SUBPROGRAMS
C
C REFERENCES
C
C VERSION          1.00 D.R.Lauchlan
C
C CHANGES
```

(continues on next page)

(continued from previous page)

```
C DESCRIPTION      Converts the two input parameters 141 (raw Heimann) and
C                  142 (black body reference temperature) into one, the
C                  uncorrected HEIMAN temp (para 537).
C
C                  The Heimann Radiometer data is converted using a quadratic
C                  fit :
C                  Surface temp =  RCONST(4) + RCONST(5)*x + RCONST(6)*x**2
C                  RCONST(4 to 6) correspond to the constants with the keyword
C                  HEIMCAL in the flight constants file.
C
C                  The black body signal (para 142) is converted using
C                  a quadratic fit :
C
C                   BB = a + b*x + c*x**2
C
C                  where constants a, b and c correspond to RCONST(1 to 3)
C                  from the keyword PRTCCAL in the flight constant file.
C
C                  Signal Register (para 27) bit 0 indicates the position of
C                  the black body;  0 = b/b out of FoV, 1 = b/b in FoV.
C
C                  If signal register bit 0 is set to 1 and black body
C                  reference temprature has been steady for the previous
C                  3 seconds (mean of each second differs by no more than
C                  0.1K), the b/b reference temperature is output.
C                  Otherwise the HEIMAN temprature is output. An offset
C                  is assigned accordingly:
C                                                para 27
C                                                 bit 0
C                   233.26 to  313.06   Heimann-    0     (o/s = 0)
C                  1233.26 to 1313.06   Ref/BB -    0     (o/s = 1000)
C                  2233.26 to 2313.06   Heimann-    1     (o/s = 2000)
C                  3233.26 to 3313.06   Ref/BB -    1     (o/s = 2000 + 1000)
C
C                  (NOTE: an offset of 1000 is never assigned under
C                         this scheme)
C
C
C                  Heimann data is output for the time that the
C                  reference temperature is output. This is done in
C                  4 second bursts imediately after the reference
C                  sequence and overwrites the ramping sections.
C                  Non reference or dwelling calibration temeratures
C                  are flagged as 2.
C                  Dwell Heimann data is output for the corresponding
C                  calibration reference period after the instrument has
C                  switched back to measure, ie para 27 bit 0 becomes 0.
C
C                  Bits 16 and 17 of the output temperature RDER(x,537) are
C                  used to flag certain data conditions as follows :
C                    Bits 16 and 17 = 00  - Good data, MEASURE/HEIMANN
C                                   = 01  - Good data, but Heimann on CALIBRATE
C                                           and outputing DWELL temp
C                                           or looking at the Black Body temps
C                                           or BB moved out of field of view
C                                           and data are last Heimann dwell
C                                           data.
```

(continues on next page)

(continued from previous page)

```
C                                  = 02  - Suspect or absent signal register
C                                          data, non-reference calibration
C                                          temperatures and non-dwelling
C                                          calibration temperatures.
C                                  = 03  - Absent data, passed through from
C                                          IRAW(x,141)
C
C ARGUMENTS         IRAW(f,141)   Raw Heimann data
C                   IRAW(f,27)    Raw signal register data
C                   IRAW(f,142)   Raw black body data
C                   IFRQ(141)     Recorded frequency of Heimann Radiometer
C                   IFRQ(27)      Recorded frequency of signal register
C                   IFRQ(142)     Recorded frequency of black body
C                   RCONST(1-6)   Constants for quadratic fit
C                   RDER(f,537)   Uncorrected Heimann temps (deg K)
C
C SUBPROGRAMS       ITSTFLG   -  Returns the value of bits 16 & 17    - SCILIB
C                   ISETFLG   -  Sets the value of bits 16 & 17       - SCILIB
C                   IBITS     -  Extracts selected bits from input    - FORTRAN
C                   BTEST     -  Tests value of selected single bit   - FORTRAN
C         C_HEIMAN_LTST_BB  -  Checks array elemenets are within
C                              +/- 86                                  - LOCAL
C
C REFERENCES
C
C VERSION           1.00   09-11-94 D.R.Lauchlan
C                   Based on C_BARNES V2.00 by D.P. Briggs
C
C CHANGES           V1.01  10/02/99  W.D.N.JACKSON
C                   Bug in flag checking of raw data fixed.
C
C                   V1.02  27/09/02  W.D.N.JACKSON
C                   Changed to include handling of 16 bit data from the new
C                   DRS.  Also now expects calibrator temp cal to be in deg C.
C
C                   V1.03 11/11/04 J.P. TAYLOR
C                   Changed to account for 16bit representation of DRS
C                   parameters.  Allowable range of BB ref means changed from
C                   +/- 6 to +/- 86 this is equivalent to 0.1K with the new
C                   DRS 16bit data stream.
C-----------------------------------------------------------------------------
      SUBROUTINE C_HEIMAN(IRAW,IFRQ,RCONST,RDER)


CDEC$ IDENT 'V1.03'
C
      INTEGER*4 IRAW(64,512),IFRQ(512)
      REAL*4    RCONST(64),RDER(64,1024)

      REAL*4    R_DWELL(5,64)      !buffer for dwelling Heimann

      INTEGER   L_LASTREG,         !last signal register
     +          L_PRESREG,         !present signal register
     +          I_COUNT,           !loop counter
     +          I_COUNT_1,         !loop counter
     +          I_COUNT_2,         !loop counter
     +          I_COUNT_3,         !loop counter
```

(continues on next page)

(continued from previous page)

```
     +           I_DWELL_COUNT     !Heimann dwell count
      INTEGER*4 I_BBREF_MEAN(3),  !last three seconds of BB mean temps
     +           I_BB_F,           !black body frequency
     +           I_SIGREG_F,       !signal register frequency
     +           IFLAG,            !output data flag
     +           ISIGFLAG          !signal register flag
C functions:
      INTEGER   C_HEIMAN_LTST_BB,ITSTFLG
      DATA L_LASTREG/.FALSE./

      SAVE
c      REAL      C_HEIMAN_RBB_CONV
C
      I_BB_F       = IFRQ(142)
      I_SIGREG_F   = IFRQ(27)

      DO I_COUNT_1 = 1,I_BB_F
        IFLAG = 0
        I_COUNT_2 = (1 + I_COUNT_1) / I_SIGREG_F
C
C if signal register set
        L_PRESREG = BTEST(IRAW(I_COUNT_2,27),0)
        IF (L_PRESREG) THEN

C  if last signal register not set
          IF (.NOT.L_LASTREG)  THEN
C    init BB ref means
            DO I_COUNT = 1,3
              I_BBREF_MEAN(I_COUNT) = I_COUNT * 100
            ENDDO
C    init dwell counter & buffer
            DO I_DWELL_COUNT = 1,5
              DO I_COUNT = 1,I_BB_F
                 R_DWELL(I_DWELL_COUNT,I_COUNT) = 2000.0
              ENDDO
            ENDDO
            I_DWELL_COUNT = 0
          ENDIF

C  if last 3 BB ref means are within +/-86 (equivalent to approx 0.1K)
          IF (C_HEIMAN_LTST_BB(I_BBREF_MEAN)) THEN !output BB ref temp
            RDER(I_COUNT_1,537) = 3000 +
     -           273.16 +
     -           RCONST(1) +
     -           RCONST(2) * FLOAT(IRAW(I_COUNT_1,142)) +
     -           RCONST(3) * FLOAT(IRAW(I_COUNT_1,142)) ** 2
C       set flag to 2 if data is out of expected range (-20 to +40 degC)
            IF (RDER(I_COUNT_1,537) .GT. 3313.16 .OR.
     +          RDER(I_COUNT_1,537) .LT. 3253.16)  IFLAG = 2

C       store Heimann for corresponding seconds (max 5)
            IF (I_COUNT_1 .EQ. 1) THEN
              I_DWELL_COUNT = I_DWELL_COUNT + 1
              IF (I_DWELL_COUNT .GT. 5) THEN
                DO I_COUNT = 5,2,-1
                  DO I_COUNT_3 = 1,I_BB_F
                    R_DWELL(I_COUNT-1,I_COUNT_3) =
```

(continues on next page)

(continued from previous page)

```
     +                           R_DWELL(I_COUNT,I_COUNT_3)
                  ENDDO
                ENDDO
                I_DWELL_COUNT = 5
              ENDIF
            ENDIF

            R_DWELL(I_DWELL_COUNT,I_COUNT_1) = 2000 +
     -           273.16 +
     -           RCONST(4) +
     -           RCONST(5) * FLOAT(IRAW(I_COUNT_1,141)) +
     -           RCONST(6) * FLOAT(IRAW(I_COUNT_1,141)) ** 2
C          write(*,*)-999.999

          ELSE      !output HEIMANN temp
C        write dwell Heimann data if any
            IF (I_DWELL_COUNT.NE.0) THEN
              IF (I_DWELL_COUNT.EQ.5)
     +            I_DWELL_COUNT = 4
              RDER(I_COUNT_1,537) =
     +           R_DWELL(I_DWELL_COUNT,I_COUNT_1)
              IF(I_COUNT_1 .EQ. I_BB_F)
     +           I_DWELL_COUNT = I_DWELL_COUNT - 1
            ELSE
              RDER(I_COUNT_1,537) =  2000 +
     -           273.16 +
     -           RCONST(4) +
     -           RCONST(5) * FLOAT(IRAW(I_COUNT_1,141)) +
     -           RCONST(6) * FLOAT(IRAW(I_COUNT_1,141)) ** 2
              IF (IFLAG .LT. 2)
     +            IFLAG = 2
            ENDIF
          ENDIF

C    roll on BB means
          IF (I_COUNT_1 .EQ. 4) THEN
            DO I_COUNT = 2,1,-1
              I_BBREF_MEAN(I_COUNT+1) = I_BBREF_MEAN(I_COUNT)
            ENDDO

            I_BBREF_MEAN(1) = 0
            DO I_COUNT = 1,IFRQ(142)
              I_BBREF_MEAN(1) = I_BBREF_MEAN(1) +
     +            IRAW(I_COUNT,142)
            ENDDO
            I_BBREF_MEAN(1) = I_BBREF_MEAN(1) / I_BB_F
          ENDIF

        ELSE        !output Heimann temp
C        write dwell HEIMAN data if any
          IF (I_DWELL_COUNT.NE.0) THEN
            IF (I_DWELL_COUNT.EQ.5)
     +          I_DWELL_COUNT = 4
            RDER(I_COUNT_1,537) =
     +         R_DWELL(I_DWELL_COUNT,I_COUNT_1)
            IF(I_COUNT_1 .EQ. I_BB_F)
     +         I_DWELL_COUNT = I_DWELL_COUNT - 1
```

(continues on next page)

(continued from previous page)

```
          ELSE
            RDER(I_COUNT__1,537) =
     -           273.16 +
     -           RCONST(4) +
     -           RCONST(5) * FLOAT(IRAW(I_COUNT_1,141)) +
     -           RCONST(6) * FLOAT(IRAW(I_COUNT_1,141)) ** 2

          ENDIF

        ENDIF

C      Get flag value
        ISIGFLAG =0
        IF(IRAW(I_COUNT_2,27).EQ.'FFFF'X) ISIGFLAG=3
        IF ((RDER(I_COUNT_1,537) .GE. 1000.0 .AND.
     +       IFLAG .EQ. 0) .OR.
     +       ISIGFLAG .EQ. 1) THEN
          IFLAG = 1
        ELSE IF (ISIGFLAG .GT. IFLAG) THEN
          IFLAG = 2
        ENDIF
C      Set flag
        CALL ISETFLG(RDER(I_COUNT_1,537),IFLAG)
C
C     roll on sig reg
        L_LASTREG = L_PRESREG

      ENDDO
C
      RETURN
      END
C-----------------------------------------------------------------------------
C
C ROUTINE          C_HEIMAN_LTST_BB
C
C PURPOSE          Checks array elemenets are within +/- 86
C
C DESCRIPTION      Returns TRUE if the deviation from the mean value of the
C                  passed array is no greater than +/- 86 otherwise FALSE.
C
C VERSION          1.00   17-02-94  D.P.Briggs
C                  1.01   11-11-04  J.P.Taylor
C                  Array now allowed to be within +/- 86  (was +/- 6)
C                  Change due to new 16bit representation on  DRS
C                  86 is equivalent to 0.1K.
C
      INTEGER FUNCTION C_HEIMAN_LTST_BB(I_MEANS)
CDEC$ IDENT 'V1.00'

      INTEGER    I_COUNT
      INTEGER*4  I_MEANS(3), I_MEAN

      C_HEIMAN_LTST_BB = .TRUE.

      I_MEAN = (I_MEANS(1) + I_MEANS(2) + I_MEANS(3)) / 3

      DO I_COUNT = 1 , 3
```

(continues on next page)

(continued from previous page)

```
        IF ( ABS(I_MEAN - I_MEANS(I_COUNT)) .GT. 86)
     +     C_HEIMAN_LTST_BB = .FALSE.
      ENDDO

      RETURN
      END
```

8.9 c_ins1.for

```
C
C ROUTINE          C_INS1 SUBROUTINE FORTVAX
C
C PURPOSE          Calibrates H-423 velocities, attitudes, and attitude rates
C
C DESCRIPTION      Handles the demultiplexing, calibration, interpolation and
C                  quality control of data from the Honeywell H-423 SNU 84
C                  Inertial Navigation Unit, to produce the three aircraft
C                  velocity components (VN, VE and VZ), the three aircraft
C                  attitudes (Roll, Pitch and True Heading), and the three
C                  aircraft attitude rates (Roll rate, Pitch rate, and Yaw
C                  rate).  All INU parameters are 32 Hz, but for ease of
C                  computation there is a 1/32 s lag in the data - ie the
C                  second sample in each second in fact describes the
C                  aircraft state at the start of the second.
C
C                  The three aircraft accelerations, in the aircraft body frame,
C                  together with INU latitude, longitude and altitude are
C                  produced at 1 Hz.
C
C                  The INU interface sends 7 16-bit parameters at 32 Hz to the
C                  DRS.  The INU interface unit requests the I01 message from the
C                  INU 32 times a second.  The whole of the first I01 message
C                  received in a second is recorded in parameter 163.  The
C                  time tags from the 2nd to 32nd messages are recorded in
C                  parameter 164, samples 2 to 32.  The least 8 bits of the
C                  velocities, attitudes, and attitude rates are packed into
C                  the remaining DRS parameters.  Because the information that
C                  would go into the first sample of parameters 164 to 169 is
C                  already available in parameter 163, the first sample in the
C                  second of each of these parameters is used to record status
C                  information as follows:
C
C                  1st sample of para 164 - IIU status word (see below)
C                  1st sample of para 165 - INU message I14, word 01
C                  1st sample of para 166 - INU message I14, word 04
C                  1st sample of para 167 - All 0's
C                  1st sample of para 168 - All 1's
C                  1st sample of para 169 - Unused (all 0's)
C
C                  The information in the above 6 words is sampled at the
C                  beginning of each second; therefore if an error is indicated
C                  some of the data in the previous second may also be bad, and
C                  not necessarily all the data in the current second may be bad.
C
```

(continues on next page)

(continued from previous page)

```
C                 IIU status word:
C
C                 Bit 15  1 if ASMA link broken, IIU or SIMON off, else 0
C                     14  1 if IIU can get no response from INU, else 0
C                     13  1 if IIU has no valid baro information, else 0
C                     12  1 if any bit set in the IIU 1553 chip sts word, else 0
C                     11-3 Unused, set to 0
C                     2-0 IIU software version
C
C                 Input parameters are:
C
C                 Para 163 I01   32 Hz  This contains the full 32 word I01
C                                       message, sampled once a second
C                      164 TTAG  32 Hz  Time tags taken from I01 messages
C                      165 VXVY  32 Hz  Bits 14-21 of VX (0-7) and VY (8-15)
C                      166 VZTH  32 Hz  Bits 14-21 of VZ (0-7) and bits 0-7 of
C                                       THDG (8-15)
C                      167 RORR  32 Hz  Bits 0-7 of ROLL (0-7) and ROLR (8-15)
C                      168 PIPR  32 Hz  Bits 0-7 of PITC (0-7) and PITR (8-15)
C                      169 PAYR  32 Hz  Bits 0-7 of PAZI (0-7) and YAWR (8-15)
C
C                 The least significant bits of the information recorded by the
C                 DRS are (H-423 manual, Table 3-1A, p3-10 onwards):
C
C                   Time tags      2**6   micro-sec
C                   Altitude       2**2   foot
C                   CNEXZ          2**-30 dimensionless
C                   Longitude      2**-31 pirads (1 pirad = 180 deg)
C                   Velocities     2**-18 foot/sec
C                   Accelerations  2**-5  foot/sec/sec
C                   Attitudes      2**-15 pirads
C                   Attitude rates 2**-13 pirads/sec
C
C                 The INU I01 message contains the following 32 16-bit words
C                 (not all used by this module):
C
C                   01                     INU mode word
C                   02                     INU time tag
C                   03/04, 05/06, 07/08    VX, VY, VZ
C                   09, 10, 11             Platform Azimuth, Roll, Pitch
C                   12, 13                 True Heading, Magnetic Heading
C                   14, 15, 16             X, Y, Z accelerations
C                   17/18, 19/20, 21/22    CNEXX, CNEXY, CNEXZ direction cosines
C                   23/24, 25              Longitude, Inertial Altitude
C                   26, 27, 28             GC steering err, X & Y residual tilts
C                   29                     INU mode word 2 - current mode
C                   30, 31, 32             Roll, Pitch and Yaw rates
C
C                 Constants:
C
C                 The following constants are used by the module to compensate
C                 for the INU not being accurately aligned with the aircraft
C                 axes, they are the values that need to be added to the INU
C                 attitudes to obtain the aircraft attitude:
C
C                   RCONST(1) Roll offset (deg)  +ve Aircraft right bank wrt INU
C                   RCONST(2) Pitch offset (deg) +ve Aircraft pitched up wrt INU
```

(continues on next page)

(continued from previous page)

```
C                   RCONST(3) Yaw offset (deg)   +ve Aircraft yawed CW wrt INU
C
C                 These are defined in the flight constants file using the
C                 INSLEVL keyword.
C
C                 Output parameters are:
C
C                 Para 538 IACF   1 Hz  m/s/s +ve Forwards
C                      539 IACS   1 Hz  m/s/s +ve Starboard
C                      540 IACU   1 Hz  m/s/s +ve Upwards
C                      541 ILAT   1 Hz  deg   -90 to +90
C                      542 ILNG   1 Hz  deg   -180 to +180
C                      543 IALT   1 Hz  m     +ve Upwards
C                      558 VN    32 Hz  m/s   +ve Northwards
C                      559 VE    32 Hz  m/s   +ve Eastwards
C                      557 VZ    32 Hz  m/s   +ve Upwards
C                      560 ROLL  32 Hz  deg   +ve Right bank
C                      561 PITC  32 Hz  deg   +ve Nose up
C                      562 THDG  32 Hz  deg   +ve CW wrt True North (0-360 deg)
C                      567 ROLR  32 Hz  deg/s +ve Banking right
C                      565 PITR  32 Hz  deg/s +ve Pitching up
C                      566 YAWR  32 Hz  deg/s +ve Yawing CW wrt North
C                      563 IGS   32 Hz  m/s   +ve Always
C                      564 IDA   32 Hz  deg   +ve Track to right of heading
C
C                 Velocities are computed in the Earth-centred, Earth-fixed
C                 frame and expressed in local geodetic coordinates.
C                 Accelerations are computed in the aircraft frame.  Positions
C                 are uncorrected and based on whatever initial positions were
C                 loaded when the INU was aligned.
C
C                 The attitude angles are the Euler angles used to transform
C                 between local geodetic and aircraft body co-ordinates.
C                 The local geodetic axes are rotated in the counterclockwise
C                 direction about the downward axis by the true heading, with
C                 the yaw rate directed along this axis.  These new axes are
C                 then rotated counterclockwise about the rotated eastward
C                 axis by the pitch angle, with the pitch rate directed along
C                 this intermediate axis.  Finally, these new axes are rotated
C                 counterclockwise about the rotated northward axis (which
C                 becomes the forward axis in the aircraft body frame) by the
C                 roll angle, with the roll rate directed along this axis.
C
C                 Since only the least eight significant bits of velocity,
C                 attitude, and attitude rate are recorded at 32 Hz, the true
C                 values have to be reconstituted.  This is done by computing
C                 the expected value and assuming that the actual value will
C                 always be within +-127 bits of the expected value.  The first
C                 expected velocity in a second is computed using the
C                 accelerations available in the I01 word.  Subsequent values
C                 are based on the changes between the previous samples.
C                 Expected attitudes are computed in the same way, using the
C                 attitude rates available in the I01 word.  Expected attitude
C                 rates are computed using the current attitude rates.
C
C                 Note that the attitude data is corrected for any INU
C                 misalignment with the aircraft, provided that the INSLEVL
```

(continues on next page)

(continued from previous page)

```
C                  constants are entered in the Flight Constants file for the
C                  flight.  Because there is a variable delay between the INU
C                  sampling the aircraft attitude and velocity and having the
C                  result ready to be read by the DRS, together with a further
C                  variable delay between the data being ready and it actually
C                  being read by the DRS, this routine linearly interpolates the
C                  INU measurements, which are made available with their actual
C                  measurement times, onto the equispaced 32 Hz sampling
C                  intervals of the DRS.
C
C                  Most of the computation is quite straight forward, but the
C                  VN and VE velocities have to be derived from the VX and VY
C                  velocities since the INU does not maintain its 'platform'
C                  aligned with True North, but lets it wander at a normally
C                  fairly slow rate.  The wander angle, a, is the difference
C                  between the INU platform azimuth and True North.  VN and VE
C                  are then derived using:
C
C                    VN =   cos(a).VX - sin(a).VY
C                    VE = - sin(a).VX - cos(a).VY
C
C                  Accelerations are converted from platform to aircraft frame
C                  by applying a suitable transformation matrix.
C
C                  INU Groundspeed, IGS, and Drift Angle, IDA, are derived from
C                  VN, VE, and THDG using:
C
C                    IGS = sqrt(VE**2 + VN**2)
C                    IDA = artan(VE/VN) - THDG
C
C                  Flagging:
C
C                  Output data are flagged with 2 if they exceed the following
C                  max, min, rate of change limits:
C
C                    VN, VE      +-250  m/s,   +-20   m/s/s
C                    VZ          +-30.5 m/s,   +-20   m/s/s
C                    ROLL, PITC  +-60   deg,   +-20   deg/s
C                    THDG        0-360  deg,   +-15   deg/s
C                    ROLR, PITR  +-20   deg/s, +-20   deg/s/s
C                    YAWR        +-15   deg/s, +-20   deg/s/s
C                    IGS         0-250  m/s,   +-20   m/s/s
C                    IDA         +-45   deg,   +-10   deg/s
C                    IACF        +-5    m/s/s, +-4    m/s/s/s
C                    IACS        +-20   m/s/s, +-4    m/s/s/s
C                    IACU      2.5-18   m/s/s, +-7    m/s/s/s
C                    ILAT        +-90   deg,   +-.015 deg/s
C                    ILNG        +-180  deg,   +-.015 deg/s
C                    IALT    -200-12000 m,     +-30.5 m/s
C
C                  Data are also flagged under any of the following
C                  circumstances:
C
C                  IIU sts bit 15 set (no ASMA link) - All data in sec flagged 3
C                  IIU sts bit 14 set (no INU link)  - All data in sec flagged 3
C                  IIU sts bit 13 set (no baro info) - All vert in sec flagged 2
C                  IIU sts bit 12 set (1553 chip err)- All data in sec flagged 2
```

(continues on next page)

(continued from previous page)

```
C                 I14/01 not zero                   - All data in sec flagged 2
C                 All zeros word not zero           - All data in sec flagged 2
C                 All ones word not FFFF            - All data in sec flagged 2
C                 I01/1 bit 1 set (Sensor fail)     - All data in sec flagged 2
C                 I01/1 bit 2 set (Nav data fail)   - All data in sec flagged 2
C                 I01/1 bit 3 set (Degraded nav)    - All data in sec flagged 2
C                 I01/1 bit 4 set (Nav data unav.)  - All data in sec flagged 2
C                 I01/1 bit 5 set (Att data fail)   - All data in sec flagged 2
C                 I01/1 bit 9 set (Baro invalid)    - All vert in sec flagged 2
C                 I01/1 bit 10 set (BIT)            - All data in sec flagged 2
C                 I01/29 any bits except 9 (NAV) set- All data in sec flagged 2
C                 I01/29 more than one bit set      - All data in sec flagged 2
C                 Time tag has a value of FFFE   - All data in sample flagged 3
C
C                 In all cases the data take the worst of all possible flags,
C                 and if the flag is three the data are set to zero.
C
C VERSION         1.00  10-01-94  W.D.N.JACKSON
C
C ARGUMENTS       IRAW(2,64,512) I*2  IN  Up to 64 samples for up to 512 DRS
C                                         parameters
C                 IFRQ(512)      I*4  IN  Sample rate of each DRS par (0-64)
C                 RCONST(64)     R*4  IN  Inputs constants
C                 RDER(64,1024)  R*4  OUT Output array of up to 64 samples for
C                                         each of 1024 parameters
C
C REFERENCES      SNU 84-1 Rev D INU specification
C                 Honeywell H-423 system description
C                 MRF Technical Note 15
C
C CHANGES         V1.01  03-02-94  W.D.N.JACKSON
C                 No longer checks I14-04 when setting flags
C
C                 V1.02  11-05-94  W.D.N.JACKSON
C                 Now produces valid data, but without interpolation, if the
C                 IIU synching of the INU time tag clock has failed.
C
C                 V1.03  25-06-94  W.D.N.JACKSON
C                 Problems with retrieving platform azimuth and true heading
C                 when crossing +-180 degrees fixed.
C
C                 V1.04  24-07-95  W.D.N.JACKSON
C                 Now retrieves accelerations and positions at 1 Hz.
C
C                 V1.05  22-01-97  W.D.N.JACKSON
C                 Bug fixed which sometimes stopped interpolation.
C
C                 V1.06  09-07-98  W.D.N.JACKSON
C                 Bug fixed which caused incorrect attitude rate calculations.
C
C                 V1.07  06-08-98  G.W. Inverarity
C                 Convert feet to m assuming they're US standard feet (WGS-84).
C
C                 V1.08  13-12-02  G.W. Inverarity
C                 1. Convert feet to m as international feet (Honeywell).
C                 2. Added RPMIN array of minimum value limits and changed
C                    RPROC(15) to 0.015, consistent with the values under
```

(continues on next page)

(continued from previous page)

```
C                          "Flagging" above.
C                       3. Replaced 4. and RTTOMS*2*16 by RTMAX (= RTTOMS*2**16)
C                          when computing time differences.
C                       4. Added extra time tag checks when deciding whether
C                          or not to interpolate 32 Hz data.
C                       5. Extrapolate 1 Hz positions by integrating the equations
C                          for the rates of change of latitude, longitude and
C                          altitude using Euler's method, working in double
C                          precision to minimise rounding error, which can be of
C                          the order of 1 metre.
C                       6. Drift angle error when northward velocity zero corrected.
C                       7. Simplified the true heading and drift angle calculations
C                          using the MOD function.
C                       8. C_INS1_TRANS_BRATE rewritten.
C*******************************************************************************
      SUBROUTINE C_INS1(IRAW,IFRQ,RCONST,RDER)

CDEC$ IDENT     'V1.08'

      INTEGER*2 IRAW(2,64,512)         !Raw data array
      INTEGER*4 IFRQ(512)              !Raw data frequency (not used)
      REAL*4    RCONST(64)             !Constants array
      REAL*4    RDER(64,1024)          !Derived data array

      INTEGER*2 IVX,IVY,IVZ,IPA,IRO,IPI,ITH,IRR,IPR,IYR
      INTEGER*2 IXVX,IXVY,IXVZ,IXPA,IXRO,IXPI,IXTH,IXRR,IXPR,IXYR
      INTEGER*2 ITEMP
      INTEGER*4 ITEMP4,ITEMP4B
      REAL*4    RVX(32),RVY(32),RPA(32),RTDIF(32)
      REAL*4    RTT(0:32),RVN(0:32),RVE(0:32),RVZ(0:32),RROLL(0:32),
     -          RPITC(0:32),RTHDG(0:32),RROLR(0:32),RPITR(0:32),
     -          RYAWR(0:32)
      INTEGER*4 IPARA(17),IFLG(17)
      REAL*4    RPMIN(17),RPMAX(17),RPROC(17),RLSTVAL(17)

      REAL*8    DT         ! Time interval by which to extrapolate 1 Hz
                           ! positions.
      REAL*8    DL         ! Rate of change of latitude with time (rad/s)
      REAL*8    DLAMBDA    ! Rate of change of longitude with time (rad/s)
      REAL*8    RL         ! Meridional radius of curvature (m)
      REAL*8    RLAMBDA    ! Azimuthal radius of curvature (m) / COS(LAT)

      PARAMETER I01=163,TTAG=164,VXVY=165,VZTH=166 !Raw parameters
      PARAMETER RORR=167,PIPR=168,PAYR=169
      PARAMETER GMT=515,VN=558,VE=559,VZ=557       !Derived parameters
      PARAMETER ROLL=560,PITC=561,THDG=562
      PARAMETER ROLR=567,PITR=565,YAWR=566
      PARAMETER IGS=563,IDA=564
      PARAMETER IACF=538,IACS=539,IACU=540,ILAT=541,ILNG=542,IALT=543

      REAL*4    RFT2MTR ! International foot to metre conversion factor
      PARAMETER(RFT2MTR=0.3048)
      REAL*8    PI       ! Pi
      PARAMETER(PI=3.1415926535897932D0)
      REAL*8    RAD2DEG ! Radians to degrees
      PARAMETER(RAD2DEG=180.0D0/PI)
```

(continues on next page)

(continued from previous page)

```
      DATA LFLAG /.TRUE./               !Set to false if don't want data flagging
      DATA LINTER /.TRUE./              !Set to false if don't want interpolation
      DATA RLSTSEC /-2.0/               !Initial value for last sec processed
      DATA IPARA /VN,VE,VZ,ROLL,PITC,THDG,ROLR,PITR,YAWR,IGS,IDA,
     -    IACF,IACS,IACU,ILAT,ILNG,IALT/ !Derived paras
      DATA RPMIN /2*-250.,-30.5,2*-60.,0.,2*-20.,-15.,0.,-45.,
     -    -5.,-20.,2.5,-90.,-180.,-200./ ! Min values
      DATA RPMAX /2*250.,30.5,2*60.,360.,2*20.,15.,250.,45.,
     -    5.,20.,18.,90.,180.,12000./ !Max values
      DATA RPROC /3*20.,2*20.,15.,3*20.,20.,10.,
     -    4.,4.,7.,0.015,0.015,30.5/ !Max rates of change/s

      SAVE
C
      RTTOMS=2.**6/1.0E6                !Converts time tags to seconds
      RTMAX=RTTOMS*2.**16               !Maximum time tag
      RVTMPS=RFT2MTR/2.**4              !Converts velocities to m/s
      RRATOD=180./2.**13                !Converts attitude rates to deg/s
      RATTOD=RRATOD/4.0                 !Converts attitudes to degrees
      RATMSS=RVTMPS/2.0                 !Converts accelerations to m/s/s
      RTSHFT=1.0/32.0                   !Data time shift in secs
      RSINT=RTSHFT                      !DRS data sample interval
C
C Set flag false if this second is not immediately after previous one.
C
      LNXTSEC=.TRUE.
      IF(RDER(1,GMT).NE.RLSTSEC+1.0) LNXTSEC=.FALSE.
C
C Retrieve time tags - if INU is getting synched by IIU then time tags will
C always be in range 0 to 1s.  If INU 1553 clock not being reset by IIU then
C time tags will be in range 0 to RTMAX = 4.194304 s (2**22 microsec).  Set
C interpolation flag to false if the first tag of the second is not in the
C known range, thus indicating that the IIU is not synching the INU.
C
      ITT=JZEXT(IRAW(1,2,I01))
      RTT(1)=FLOAT(ITT)*RTTOMS
      DO I=2,32
        ITT=JZEXT(IRAW(1,I,TTAG))
        RTT(I)=FLOAT(ITT)*RTTOMS
      END DO
      DO I=2,32
        RTDIF(I)=RTT(I)-RTT(I-1)
        IF(RTDIF(I).LE.-RTMAX) RTDIF(I)=RTDIF(I)+RTMAX
        IF(RTDIF(I).LE.0.) RTDIF(I)=RTDIF(I)+1.0
      END DO
      LINTERP=LINTER
      IF(RTT(1).LT.0.970.OR.RTT(1).GT.0.991) LINTERP=.FALSE.
C
C Compute platform accelerations for use in retrieving VX, VY and VZ.
C
      RAX=FLOAT(IRAW(1,14,I01))*RATMSS
      RAY=FLOAT(IRAW(1,15,I01))*RATMSS
      RAZ=FLOAT(IRAW(1,16,I01))*RATMSS-9.75 !Remove gravitational acceleration
C
C Retrieve VX
C
      CALL MVBITS(IRAW(1,3,I01),0,14,IVX,2)
```

(continues on next page)

(continued from previous page)

```
      CALL MVBITS(IRAW(1,4,I01),14,2,IVX,0)
      RVX(1)=FLOAT(IVX)*RVTMPS
      DO I=2,32
        IF(I.EQ.2) RXVX=RVX(1)+RAX*RTDIF(2)
        IF(I.GT.2) RXVX=RVX(I-1)+(RVX(I-1)-RVX(I-2))/RTDIF(I-1)*RTDIF(I)
        RXVX=RXVX/RVTMPS
        RXVX=MIN(RXVX,32767.)
        RXVX=MAX(RXVX,-32768.)
        IXVX=NINT(RXVX)
        CALL C_INS1_MERGE(IXVX,IRAW(1,I,VXVY),0,IVX)
        RVX(I)=FLOAT(IVX)*RVTMPS
      END DO
C
C Retrieve VY
C
      CALL MVBITS(IRAW(1,5,I01),0,14,IVY,2)
      CALL MVBITS(IRAW(1,6,I01),14,2,IVY,0)
      RVY(1)=FLOAT(IVY)*RVTMPS
      DO I=2,32
        IF(I.EQ.2) RXVY=RVY(1)+RAY*RTDIF(2)
        IF(I.GT.2) RXVY=RVY(I-1)+(RVY(I-1)-RVY(I-2))/RTDIF(I-1)*RTDIF(I)
        RXVY=RXVY/RVTMPS
        RXVY=MIN(RXVY,32767.)
        RXVY=MAX(RXVY,-32768.)
        IXVY=NINT(RXVY)
        CALL C_INS1_MERGE(IXVY,IRAW(1,I,VXVY),8,IVY)
        RVY(I)=FLOAT(IVY)*RVTMPS
      END DO
C
C Retrieve VZ
C
      CALL MVBITS(IRAW(1,7,I01),0,14,IVZ,2)
      CALL MVBITS(IRAW(1,8,I01),14,2,IVZ,0)
      RVZ(1)=FLOAT(IVZ)*RVTMPS
      DO I=2,32
        IF(I.EQ.2) RXVZ=RVZ(1)+RAZ*RTDIF(2)
        IF(I.GT.2) RXVZ=RVZ(I-1)+(RVZ(I-1)-RVZ(I-2))/RTDIF(I-1)*RTDIF(I)
        RXVZ=RXVZ/RVTMPS
        RXVZ=MIN(RXVZ,32767.)
        RXVZ=MAX(RXVZ,-32768.)
        IXVZ=NINT(RXVZ)
        CALL C_INS1_MERGE(IXVZ,IRAW(1,I,VZTH),0,IVZ)
        RVZ(I)=FLOAT(IVZ)*RVTMPS
      END DO
C
C Retrieve lat, long and altitude
C
      ITEMP4=IRAW(1,21,i01)
      CALL MVBITS(ITEMP4,0,16,ITEMP4B,16) !Latitude
      ITEMP4=IRAW(1,22,I01)
      CALL MVBITS(ITEMP4,0,16,ITEMP4B,0)
      RCNEXZ=FLOAT(ITEMP4B)/2.**30
      IF(RCNEXZ.GE.-1.AND.RCNEXZ.LE.1) RLAT=ASIND(RCNEXZ)
      IF (RLAT.GT.89.9)
     &  print *,'Latitude close to 90 could cause problems'
      IF (RLAT.LT.-89.9)
     &  print *,'Latitude close to -90 could cause problems'
```

(continues on next page)

(continued from previous page)

```
      ITEMP4=IRAW(1,23,I01)
      CALL MVBITS(ITEMP4,0,16,ITEMP4B,16) !Longitude
      ITEMP4=IRAW(1,24,I01)
      CALL MVBITS(ITEMP4,0,16,ITEMP4B,0)
      RLNG=FLOAT(ITEMP4B)*180./2.**31
      RALT=FLOAT(IRAW(1,25,I01))*4.*RFT2MTR !Height in metres
C
C Compute basic attitude and attitude rates at start of second, and transform
C attitude rates from aircraft body co-ordinates to rates of change of
C Euler angles.
C
      RPA(1)=FLOAT(IRAW(1,9,I01))*RATTOD
      RROLL(1)=FLOAT(IRAW(1,10,I01))*RATTOD
      RPITC(1)=FLOAT(IRAW(1,11,I01))*RATTOD
      RTHDG(1)=FLOAT(IRAW(1,12,I01))*RATTOD
      RROLR(1)=FLOAT(IRAW(1,30,I01))*RRATOD
      RPITR(1)=FLOAT(IRAW(1,31,I01))*RRATOD
      RYAWR(1)=FLOAT(IRAW(1,32,I01))*RRATOD
C Compute wander angle and transform VX and VY to VN and VE
      RWA=RPA(1)-RTHDG(1)
      RVN(1)=COSD(RWA)*RVX(1)-SIND(RWA)*RVY(1)
      RVE(1)=-SIND(RWA)*RVX(1)-COSD(RWA)*RVY(1)
      CALL C_INS1_TRANS_BRATE(RLAT,RALT,RVN(1),RVE(1),RROLR(1),RPITR(1),
     &     RYAWR(1),RROLL(1),RPITC(1),RTHDG(1),RRR1,RPR1,RYR1)
      RROLR(1)=RRR1
      RPITR(1)=RPR1
      RYAWR(1)=RYR1
C
C Retrieve Roll
C
      DO I=2,32
        IF(I.EQ.2) RXRO=RROLL(1)+RRR1*RTDIF(2)
        IF(I.GT.2) RXRO=RROLL(I-1)+(RROLL(I-1)-RROLL(I-2))/RTDIF(I-1)
     -       *RTDIF(I)
        RXRO=RXRO/RATTOD
        RXRO=MIN(RXRO,32767.)
        RXRO=MAX(RXRO,-32768.)
        IXRO=NINT(RXRO)
        CALL C_INS1_MERGE(IXRO,IRAW(1,I,RORR),0,IRO)
        RROLL(I)=FLOAT(IRO)*RATTOD
      END DO
C
C Retrieve Pitch
C
      DO I=2,32
        IF(I.EQ.2) RXPI=RPITC(1)+RPR1*RTDIF(2)
        IF(I.GT.2) RXPI=RPITC(I-1)+(RPITC(I-1)-RPITC(I-2))/RTDIF(I-1)
     -       *RTDIF(I)
        RXPI=RXPI/RATTOD
        RXPI=MIN(RXPI,32767.)
        RXPI=MAX(RXPI,-32768.)
        IXPI=NINT(RXPI)
        CALL C_INS1_MERGE(IXPI,IRAW(1,I,PIPR),0,IPI)
        RPITC(I)=FLOAT(IPI)*RATTOD
      END DO
C
C Retrieve Platform azimuth
```

(continues on next page)

(continued from previous page)

```
C
      DO I=2,32
        IF(I.EQ.2) RXPA=RPA(1)+RYR1*RTDIF(2)
        IF(I.GT.2) THEN
          RDIFF=MOD(MOD(RPA(I-1)-RPA(I-2),360.)+360.,360.)
          IF(RDIFF.GT.180.) RDIFF=RDIFF-360.
          RXPA=RPA(I-1)+RDIFF/RTDIF(I-1)*RTDIF(I)
        END IF
        RXPA=RXPA/RATTOD
        IF(RXPA.GT.32767.) RXPA=RXPA-65536.
        IF(RXPA.LT.-32768.) RXPA=RXPA+65536.
        RXPA=MIN(RXPA,32767.)
        RXPA=MAX(RXPA,-32768.)
        IXPA=NINT(RXPA)
        CALL C_INS1_MERGE(IXPA,IRAW(1,I,PAYR),0,IPA)
        RPA(I)=FLOAT(IPA)*RATTOD
      END DO
C
C Retrieve Heading
C
      DO I=2,32
        IF(I.EQ.2) RXTH=RTHDG(1)+RYR1*RTDIF(2)
        IF(I.GT.2) THEN
          RDIFF=MOD(MOD(RTHDG(I-1)-RTHDG(I-2),360.)+360.,360.)
          IF(RDIFF.GT.180.) RDIFF=RDIFF-360.
          RXTH=RTHDG(I-1)+RDIFF/RTDIF(I-1)*RTDIF(I)
        END IF
        RXTH=RXTH/RATTOD
        IF(RXTH.GT.32767.) RXTH=RXTH-65536.
        IF(RXTH.LT.-32768.) RXTH=RXTH+65536.
        RXTH=MIN(RXTH,32767.)
        RXTH=MAX(RXTH,-32768.)
        IXTH=NINT(RXTH)
        CALL C_INS1_MERGE(IXTH,IRAW(1,I,VZTH),8,ITH)
        RTHDG(I)=FLOAT(ITH)*RATTOD
      END DO
C
C Retrieve Roll rate
C
      IRR=IRAW(1,30,I01)
      DO I=2,32
        IXRR=IRR
        CALL C_INS1_MERGE(IXRR,IRAW(1,I,RORR),8,IRR)
        RROLR(I)=FLOAT(IRR)*RRATOD
      END DO
C
C Retrieve Pitch rate
C
      IPR=IRAW(1,31,I01)
      DO I=2,32
        IXPR=IPR
        CALL C_INS1_MERGE(IXPR,IRAW(1,I,PIPR),8,IPR)
        RPITR(I)=FLOAT(IPR)*RRATOD
      END DO
C
C Retrieve Yaw rate
C
```

(continues on next page)

(continued from previous page)

```
      IYR=IRAW(1,32,I01)
      DO I=2,32
        IXYR=IYR
        CALL C_INS1_MERGE(IXYR,IRAW(1,I,PAYR),8,IYR)
        RYAWR(I)=FLOAT(IYR)*RRATOD
      END DO
C
C Compute wander angle and transform VX and VY to VN and VE
C
      DO I=2,32
        RWA=RPA(I)-RTHDG(I)
        RVN(I)=COSD(RWA)*RVX(I)-SIND(RWA)*RVY(I)
        RVE(I)=-SIND(RWA)*RVX(I)-COSD(RWA)*RVY(I)
      END DO
C
C Transform the roll, pitch and yaw rates just recovered from aircraft body
C co-ordinates to rates of change of Euler angles.
C
      DO I=2,32
        CALL C_INS1_TRANS_BRATE(RLAT,RALT,RVN(I),RVE(I),RROLR(I),
     &        RPITR(I),RYAWR(I),RROLL(I),RPITC(I),RTHDG(I),RRR1,
     &        RPR1,RYR1)
        RROLR(I)=RRR1
        RPITR(I)=RPR1
        RYAWR(I)=RYR1
      END DO
C
C Retrieve accelerations and transform from platform to aircraft co-ordinates.
C Units are m/s/s and the normal up value is about 9.85.
C
      RAX=FLOAT(IRAW(1,14,I01))*RATMSS
      RAY=FLOAT(IRAW(1,15,I01))*RATMSS
      RAZ=FLOAT(IRAW(1,16,I01))*RATMSS
      CALL C_INS1_TRANS_ACCL(RAX,RAY,RAZ,RROLL(1),RPITC(1),RPA(1),
     -    RACF,RACS,RACU)
C
C Interpolate data onto equispaced 32 Hz intervals, using data time tags
C Because (assuming) the first data sample of each DRS second contains data
C with a time tag that puts the data validity point near the end of the
C previous second, the data is shifted by one sample interval so that it can
C be accurately interpolated.  As a result there is a 1/32 s shift on all INU
C parameters output by this subroutine and, for example, the 2nd sample in
C each output second has a validity time which is exactly the start of the
C second, whereas the 1st sample should really be the 32 sample of the previous
C second's data.
C
C It is necessary to interpolate the data because of computation delays and
C differences between the DRS and the INU sampling frequencies.  However since
C all data have an accurate (to 64 us) validity time attached this (linear)
C interpolation is relatively easy to do.
C
      DO I=0,32                         !Time shift the time tags
        RTT(I)=RTT(I)+RTSHFT
      END DO
      DO I=31,0,-1                      !Make sure times increase
        IF(RTT(I).GT.RTT(I+1)) RTT(I)=RTT(I)-1.0
      END DO
```

(continues on next page)

(continued from previous page)

```
C
C If haven't got the last sample from the previous second then extrapolate
C back from the first and second samples of the current second.
C
      IF(.NOT.LNXTSEC) THEN
        RTT(0)=2*RTT(1)-RTT(2)
        RVN(0)=2*RVN(1)-RVN(2)
        RVE(0)=2*RVE(1)-RVE(2)
        RVZ(0)=2*RVZ(1)-RVZ(2)
        RROLL(0)=2*RROLL(1)-RROLL(2)
        RPITC(0)=2*RPITC(1)-RPITC(2)
        RTHDG(0)=2*RTHDG(1)-RTHDG(2)
        RROLR(0)=2*RROLR(1)-RROLR(2)
        RPITR(0)=2*RPITR(1)-RPITR(2)
        RYAWR(0)=2*RYAWR(1)-RYAWR(2)
      END IF
C
C Interpolate the data
C
      DO I=1,32
        IF(.NOT.LINTERP.OR.RTT(I-1).GE.RTT(I).OR.RTT(I-1).LT.-RTSHFT
     &    .OR.RTT(I-1).GT.1.0.OR.RTT(I).LT.-RTSHFT.OR.RTT(I).GT.1.0)
     &    THEN            !Can't interpolate
          RPROP=1.
        ELSE
          RPROP=(RSINT*(I-1)-RTT(I-1))/(RTT(I)-RTT(I-1)) !Interpolation proporti
        END IF
        RDER(I,VN)=RVN(I-1)+(RVN(I)-RVN(I-1))*RPROP
        RDER(I,VE)=RVE(I-1)+(RVE(I)-RVE(I-1))*RPROP
        RDER(I,VZ)=RVZ(I-1)+(RVZ(I)-RVZ(I-1))*RPROP
        RDER(I,ROLL)=RROLL(I-1)+(RROLL(I)-RROLL(I-1))*RPROP
        RDER(I,PITC)=RPITC(I-1)+(RPITC(I)-RPITC(I-1))*RPROP
        RDER(I,ROLR)=RROLR(I-1)+(RROLR(I)-RROLR(I-1))*RPROP
        RDER(I,PITR)=RPITR(I-1)+(RPITR(I)-RPITR(I-1))*RPROP
        RDER(I,YAWR)=RYAWR(I-1)+(RYAWR(I)-RYAWR(I-1))*RPROP
        RDIFF=MOD(MOD(RTHDG(I)-RTHDG(I-1),360.)+360.,360.) !Heading needs special␣
↪treatment
        IF(RDIFF.GT.180.) RDIFF=RDIFF-360.
        RDER(I,THDG)=MOD(MOD(RTHDG(I-1)+RDIFF*RPROP,360.)+360.,360.)
      END DO
C
C Save last samples of second for use in next second.
C
      RTT(0)=RTT(32)-RTSHFT            !Restore original time tag
      RVN(0)=RVN(32)
      RVE(0)=RVE(32)
      RVZ(0)=RVZ(32)
      RROLL(0)=RROLL(32)
      RPITC(0)=RPITC(32)
      RTHDG(0)=RTHDG(32)
      RROLR(0)=RROLR(32)
      RPITR(0)=RPITR(32)
      RYAWR(0)=RYAWR(32)
      RLSTSEC=RDER(1,GMT)
C
C Correct attitudes for INU levelling errors.
C
```

(continues on next page)

(continued from previous page)

```
      DO I=1,32
        RDER(I,ROLL)=RDER(I,ROLL)+RCONST(1)
        RDER(I,PITC)=RDER(I,PITC)+RCONST(2)
        RDER(I,THDG)=MOD(MOD(RDER(I,THDG)+RCONST(3),360.)+360.,360.)
      END DO
C
C Compute ground speed and drift angle
C
      DO I=1,32
        RDER(I,IGS)=SQRT(RVN(I)**2+RVE(I)**2)
        IF(RDER(I,VE).EQ.0..AND.RDER(I,VN).EQ.0.) THEN
          RTRK=0.
        ELSE
          RTRK=ATAN2D(RDER(I,VE),RDER(I,VN))
        END IF
        RIDA=MOD(MOD(RTRK-RDER(I,THDG),360.)+360.,360.)
        IF(RIDA.GT.180.) RIDA=RIDA-360.
        RDER(I,IDA)=RIDA
      END DO
C
C Transfer 1Hz data to output buffer.
C
      RDER(1,IACF)=RACF
      RDER(1,IACS)=RACS
      RDER(1,IACU)=RACU
C
C Extrapolate the positions forward to the true start of the second by
C using Euler's method to integrate the equations for the rates of
C change of latitude, longitude and altitude.  Use double precision to
C inhibit further amplification of rounding error, which corresponds to
C position errors on the order of a metre.
C
      IF(LINTERP) THEN
         CALL RLATLONG(DBLE(RLAT),RALT,RVN(1),RVE(1),RL,RLAMBDA,DL,
     &         DLAMBDA)
         DT=DBLE(RTSHFT)-DBLE(RTT(1))
         IF(DT.LT.0.0D0) DT=DT+1.0D0
         RDER(1,ILAT)=REAL(DBLE(RLAT)+DT*DL*RAD2DEG)
         RDER(1,ILNG)=REAL(DBLE(RLNG)+DT*DLAMBDA*RAD2DEG)
         RDER(1,IALT)=REAL(DBLE(RALT)+DT*DBLE(RVZ(1)))
      ELSE !Can't interpolate
         RDER(1,ILAT)=RLAT
         RDER(1,ILNG)=RLNG
         RDER(1,IALT)=RALT
      END IF
C
C Flag the data if required:
C
C   IIU sts bit 15 set (no ASMA link) - All data in sec flagged 3
C   IIU sts bit 14 set (no INU link)  - All data in sec flagged 3
C   IIU sts bit 13 set (no baro info) - All vert in sec flagged 2
C   IIU sts bit 12 set (1553 chip err)- All data in sec flagged 2
C   I14/01 not zero                   - All data in sec flagged 2
C   All zeros word not zero           - All data in sec flagged 2
C   All ones word not FFFF            - All data in sec flagged 2
C   I01/1 bit 1 set (Sensor fail)     - All data in sec flagged 2
C   I01/1 bit 2 set (Nav data fail)   - All data in sec flagged 2
```

(continues on next page)

(continued from previous page)

```
C   I01/1 bit 3 set (Degraded nav)    - All data in sec flagged 2
C   I01/1 bit 4 set (Nav data unav.)  - All data in sec flagged 2
C   I01/1 bit 5 set (Att data fail)   - All data in sec flagged 2
C   I01/1 bit 9 set (Baro invalid)    - All vert in sec flagged 2
C   I01/1 bit 10 set (BIT)            - All data in sec flagged 2
C   I01/29 any bits except 9 (NAV) set- All data in sec flagged 2
C   I01/29 more than one bit set      - All data in sec flagged 2
C   Time tag has a value of FFFE      - All data in sample flagged 3
C
C   Apply max/min/rate of change checks to output parameters.
C
      IF(LFLAG) THEN
        IALLFLG=0
        IVFLG=0
        IF(BTEST(IRAW(1,1,TTAG),15)) IALLFLG=3         !No link to IIU
        IF(BTEST(IRAW(1,1,TTAG),14)) IALLFLG=3         !No link to INU
        IF(BTEST(IRAW(1,1,TTAG),13)) IVFLG=2           !IIU has no baro data
        IF(BTEST(IRAW(1,1,TTAG),12)) IALLFLG=MAX(IALLFLG,2) !IIU 1553 chip error
        IF(IRAW(1,1,VXVY).NE.0) IALLFLG=MAX(IALLFLG,2) !I14/01 has a bit set
        IF(IRAW(1,1,RORR).NE.0) IALLFLG=MAX(IALLFLG,2) !Some 0s missing
        IF(IRAW(1,1,PIPR).NE.'FFFF'X) IALLFLG=MAX(IALLFLG,2) !Some 1s missing
        IF(BTEST(IRAW(1,1,I01),16-1)) IALLFLG=MAX(IALLFLG,2) !Sensor fail
        IF(BTEST(IRAW(1,1,I01),16-2)) IALLFLG=MAX(IALLFLG,2) !Nav data fail
        IF(BTEST(IRAW(1,1,I01),16-3)) IALLFLG=MAX(IALLFLG,2) !Degraded nav
        IF(BTEST(IRAW(1,1,I01),16-4)) IALLFLG=MAX(IALLFLG,2) !Nav data unavail
        IF(BTEST(IRAW(1,1,I01),16-5)) IALLFLG=MAX(IALLFLG,2) !Attitude data fail
        IF(BTEST(IRAW(1,1,I01),16-9)) IVFLG=2                !Baro invalid
        IF(BTEST(IRAW(1,1,I01),16-10)) IALLFLG=MAX(IALLFLG,2) !Doing BIT
        IF(.NOT.BTEST(IRAW(1,29,I01),16-9)) IALLFLG=MAX(IALLFLG,2) !Not NAVIGATE
        ISET=0
        DO I=0,15
          IF(BTEST(IRAW(1,29,I01),I)) ISET=ISET+1
        END DO
        IF(ISET.GT.1) IALLFLG=MAX(IALLFLG,2)
        DO J=1,17
          IFLG(J)=IALLFLG
          IF(J.EQ.3.OR.J.EQ.14.OR.J.EQ.17)
     -        IFLG(J)=MAX(IALLFLG,IVFLG) !Flag vertical measurements separately
          IF(J.EQ.10) IFLG(J)=MAX(IALLFLG,IFLG(1),IFLG(2)) !G/S
          IF(J.EQ.11) IFLG(J)=MAX(IALLFLG,IFLG(1),IFLG(2)) !D/A
        END DO
        DO I=1,32                      !Flag 32Hz parameters
          IFL=0
          IF(IRAW(1,I,TTAG).EQ.'FFFE'X) IFL=3
          DO J=1,11
            IFLAG=MAX(IFL,IFLG(J))
            IP=IPARA(J)
            IF(RDER(I,IP).LT.RPMIN(J).OR.RDER(I,IP).GT.RPMAX(J))
     &           IFLAG=MAX(2,IFLAG) ! Max/min checks
            IF((LNXTSEC.OR.I.GT.1).AND.ITSTFLG(RLSTVAL(J)).LT.2) THEN !ROC chks
              RDIFF=RDER(I,IP)-RLSTVAL(J)
              IF (J.EQ.6) THEN
                 RDIFF=MOD(MOD(RDIFF,360.)+360.,360.)
                 IF(RDIFF.GT.180.) RDIFF=RDIFF-360.
              END IF
              IF(ABS(RDIFF)*32.GT.RPROC(J)) IFLAG=MAX(2,IFLAG)
            END IF
```

(continues on next page)

(continued from previous page)

```
            IF(IFLAG.EQ.3) RDER(I,IP)=0.
            CALL ISETFLG(RDER(I,IP),IFLAG)
            RLSTVAL(J)=RDER(I,IP)       !Save last value
          END DO
        END DO
        DO J=12,17                      !Flag 1Hz parameters
          IFLAG=MAX(IFL,IFLG(J))
          IP=IPARA(J)
          IF(RDER(1,IP).LT.RPMIN(J).OR.RDER(1,IP).GT.RPMAX(J))
     &         IFLAG=MAX(2,IFLAG) ! Max/min checks
          IF(LNXTSEC.AND.ITSTFLG(RLSTVAL(J)).LT.2) THEN !ROC chks
            RDIFF=RDER(1,IP)-RLSTVAL(J)
            IF(ABS(RDIFF).GT.RPROC(J)) IFLAG=MAX(2,IFLAG)
          END IF
          IF(IFLAG.EQ.3) RDER(1,IP)=0.
          CALL ISETFLG(RDER(1,IP),IFLAG)
          RLSTVAL(J)=RDER(1,IP)         !Save last value of second
        END DO
      END IF
C
      RETURN
      END
C***********************************************************************
      SUBROUTINE C_INS1_MERGE(IEXPVAL,IPART,ISTART,INEWVAL)
CDEC$ IDENT 'V1.00'
C
C Reconstitutes a full sixteen bit word, using the expected 16 bit value, and
C the lowest 8 bits of the new 16 bit value.  Can only work when the expected
C value is correct to within +- 127 bits.
C
C Arguments:  IEXPVAL I*2 In  Expected new value (-32768 to 32767)
C             IPART   I*2 In  Contains new lowest 8 bits, in top or bottom byte
C             ISTART  I*4 In  0 if 8 bits in low bytes, else 8 bits in top byte
C             INEWVAL I*2 Out New value (-32768 to 32767)
C
      INTEGER*2 IEXPVAL,IPART,INEWVAL,ITEMP1,ITEMP2
      INTEGER*4 ISTART
      BYTE      BTEMP1(2),BTEMP2(2)
      EQUIVALENCE (ITEMP1,BTEMP1),(ITEMP2,BTEMP2)

      ITEMP1=IEXPVAL
      ITEMP2=IPART
      IF(ISTART.EQ.0) BTEMP1(1)=BTEMP2(1)
      IF(ISTART.NE.0) BTEMP1(1)=BTEMP2(2)
      IF(ITEMP1-IEXPVAL.GE.128) THEN
        IF(ITEMP1.GE.-32512) THEN
          ITEMP1=ITEMP1-256
        ELSE
          BTEMP1(2)='7F'X               !Heading/azimuth -180 to +180 change
        END IF
      ELSE IF(ITEMP1-IEXPVAL.LE.-128) THEN
        IF(ITEMP1.LE.32511) THEN
          ITEMP1=ITEMP1+256
        ELSE
          BTEMP1(2)='80'X               !Heading/azimuth +180 to -180 change
        END IF
      END IF
```

(continues on next page)

(continued from previous page)

```
      INEWVAL=ITEMP1
      RETURN
      END
C*****************************************************************************
!+    Convert the attitude rate vector into Euler angle rates of change.
!
!     *********************** COPYRIGHT ********************
!     Crown Copyright 2002, Met Office. All rights reserved.
!     *********************** COPYRIGHT ********************
!
!     Subroutine Interface:

      SUBROUTINE C_INS1_TRANS_BRATE(LAT,ALT,VN,VE,RP,RQ,RR,RROL,RPIT,
     &     RHDG,RRR,RPR,RYR)

CDEC$ IDENT 'V2.00'

      IMPLICIT NONE

!     Description:
!     Transforms attitude rate vector (p,q,r) of the aircraft with respect
!     to the inertial frame expressed in aircraft body co-ordinates to
!     the true yaw, pitch and roll rates (psi', theta' and phi',
!     respectively) about the local geodetic frame's downward axis, an
!     intermediate rotated horizontal axis and the aircraft body frame's
!     forward axis, respectively.
!
!     Method:
!     The angular frequency vector of the aircraft with respect to the
!     inertial frame has components p, q and r with respect to the
!     aircraft's forward, starboard and downward axes, respectively.  To
!     obtain the angular frequency of the aircraft with respect to the
!     local geodetic frame it is necessary to subtract the sum of the
!     angular frequency of the local geodetic frame with respect to the
!     Earth frame and of the Earth frame with respect to the inertial
!     frame: Omega + rho = (Omega + d lambda / dt) Xhat + d L / dt Yhat,
!     where Xhat and Yhat are, respectively, the unit vectors directed
!     from the Earth's centre towards the north pole and the point in
!     the equatorial plane at 90 degrees west.  The rates of change of
!     latitude L and longitude lambda with respect to time are given by
!     routine RLATLONG in this file.  This vector is now transformed
!     into aircraft body co-ordinates by first transforming it into
!     local geodetic co-ordinates by pre-multiplying by the
!     transformation matrix
!
!      ( cos(L)   0  -sin(L) )
!      (    0     1     0    )
!      ( sin(L)   0   cos(L) )
!
!     to obtain Omega + rho = (Omega + d lambda / d t) cos(L) nhat + d L
!     / d t what + (Omega + d lambda / d t) sin(L) zhat, where nhat,
!     what and zhat are the unit vectors in the northward, westward and
!     upward directions.  Next, this is transformed into aircraft body
!     co-ordinates by pre-multiplying by the direction cosine matrix
!     (DCM) of C_INS1_TRANS_ACCL using the true heading rather than the
!     platform azimuth.  This is itself obtained from a sequence of
!     three Euler angle rotations.  The DCM for a counterclockwise
```

(continues on next page)

(continued from previous page)

```
!     rotation of the true heading psi about the downward geodetic axis
!     is
!
!                      ( cos(psi)  sin(psi)   0 )
!      A3(psi) =       (-sin(psi)  cos(psi)   0 ).
!                      (   0          0       1 )
!
!     Next, the angle about which the pitch angle is defined is obtained
!     by rotating the east axis in the local geodetic frame
!     counterclockwise about the downward geodetic axis by the heading
!     angle psi.  The DCM for a counterclockwise rotation of the pitch
!     angle theta about this intermediate horizontal axis is
!
!                      ( cos(theta)  0  -sin(theta) )
!      A2(theta) =     (     0       1       0      ).
!                      ( sin(theta)  0   cos(theta) )
!
!     The DCM for a counterclockwise rotation of the roll angle phi
!     about the aircraft forward axis is
!
!                      ( 1      0          0    )
!      A1(phi) =       ( 0   cos(phi)  sin(phi) ).
!                      ( 0  -sin(phi)  cos(phi) )
!
!     Finally, the transformation matrix required to flip a
!     north/west/up co-ordinate system into a north/east/down system is
!
!                      ( 1  0  0 )
!     Oflip =          ( 0 -1  0 ).
!                      ( 0  0 -1 )
!
!     Putting these together, the DCM for transforming from the
!     local geodetic frame to the aircraft body frame is
!     CGB =  A1(phi) A2(theta) A3(psi) Oflip.
!
!     Having subtracted CGB (Omega + rho) from (p, q, r) to obtain the
!     attitude rate vector of the aircraft with respect to the local
!     geodetic frame in aircraft body co-ordinates (p', q', r'), this
!     can be written in terms of the roll rate phi', pitch rate theta'
!     and yaw rate psi' as
!
!     ( p' )    ( phi')           (   0    )                      (  0  )
!     ( q' ) = (   0   ) + A1(phi)  ( theta') + A1(phi) A2(theta) (  0  ),
!     ( r' )    (   0   )           (   0    )                      ( psi')
!
!     giving
!
!     p' =  phi' - psi' sin(theta)
!     q' =  theta' cos(phi) + psi' cos(theta) sin(phi)
!     r' = -theta' sin(phi) + psi' cos(theta) cos(phi)
!
!     which inverts to give (when theta != +/- 90 degrees)
!
!     phi' = p' + (q' sin(phi) + r' cos(phi)) tan(theta)
!     theta' = q' cos(phi) - r' sin(phi)
!     psi' = (q' sin(phi) + r' cos(phi)) / cos(theta).
!
```

(continues on next page)

(continued from previous page)

```
!     When theta = +/- 90 degrees, so that the aircraft is pointing
!     straight up or down, the heading and roll angles are no longer
!     unique so the transformation fails.  This condition is trapped to
!     prevent the code crashing in this unlikely eventuality, with roll
!     and yaw rates of 0.0 deg/s returned instead.  There is no need to
!     return a warning flag, however, as pitch angles this steep already
!     trip the max/min checks in the main body of code, leading to
!     computed values having flags of 2.
!
!     Current Code Owner: W.D.N. Jackson
!
!     History:
!     Version  Date      Comment
!
!       1.00   10/01/94  Original code. (W.D.N. Jackson)
!       1.01   09/07/98  Volatile statement added to fix bug which caused
!                        wrong values to be returned when the input and output
!                        arguments were the same (W.D.N. Jackson)
!       2.00   08/10/02  Transformations corrected to yield the true yaw,
!                        pitch and roll rates of the aircraft with respect to
!                        the local geodetic frame. (G.W. Inverarity)
!
!     Code Description:
!     FORTRAN 77 with extensions recommended in the Met Office F77
!     Standard.
!
!     Scalar arguments with INTENT(IN):
!
      REAL*4 LAT         ! Latitude (deg)
      REAL*4 ALT         ! Altitude (m)
      REAL*4 VN          ! Northward velocity component (m/s)
      REAL*4 VE          ! Eastward velocity component (m/s)
      REAL*4 RP          ! Attitude rate component forward axis (deg/s)
      REAL*4 RQ          ! Attitude rate component starboard axis (deg/s)
      REAL*4 RR          ! Attitude rate component downward axis (deg/s)
      REAL*4 RROL        ! Roll angle phi (deg)
      REAL*4 RPIT        ! Pitch angle theta (deg)
      REAL*4 RHDG        ! Heading angle psi (deg)

!     Scalar arguments with INTENT(OUT):

      REAL*4 RRR         ! Roll rate (deg/s)
      REAL*4 RPR         ! Pitch rate (deg/s)
      REAL*4 RYR         ! Yaw rate (deg/s)

!     Local Parameters:

      REAL*8    OMEGA        ! WGS-84 Earth angular frequency (rad/s)
      PARAMETER (OMEGA=7.292115D-5)

!     Local Scalars:

      REAL*4 CR          ! cos(phi)
      REAL*4 SR          ! sin(phi)
      REAL*4 OMEGAF      ! Angular frequency component forward axis
      REAL*4 OMEGAN      ! Angular frequency component northward axis
      REAL*4 OMEGAS      ! Angular frequency component starboard axis
```

(continues on next page)

(continued from previous page)

```
      REAL*4 OMEGAW     ! Angular frequency component westward axis
      REAL*4 OMEGAU     ! Angular frequency component body upward axis
      REAL*4 OMEGAZ     ! Angular frequency component geodetic upward axis
      REAL*4 RP1        ! Corrected angular frequency component forward axis
      REAL*4 RQ1        ! Corrected angular frequency component starboard axis
      REAL*4 RR1        ! Corrected angular frequency component downward axis
      REAL*4 TERM       ! q' sin(phi) + r' cos(phi)
      REAL*8 DL         ! Rate of change of latitude (rad/s)
      REAL*8 DLAMBDA    ! Rate of change of longitude (rad/s)
      REAL*8 DTERM      ! Omega + d lambda / d t
      REAL*8 RL         ! Meridional radius of curvature (m)
      REAL*8 RLAMBDA    ! Azimuthal radius of curvature (m) / COS(LAT)

!     Functions and subroutines used:

      EXTERNAL C_INS1_TRANS_ACCL ! In this file
      EXTERNAL RLATLONG ! In this file

!-    End of header

! Sum of Earth rate and platform rate angular frequency vectors in
! local geodetic co-ordinates.

      CALL RLATLONG(DBLE(LAT), ALT, VN, VE, RL, RLAMBDA, DL, DLAMBDA)
      DTERM = OMEGA + DLAMBDA
      OMEGAN = REAL(DTERM * COSD(DBLE(LAT)))
      OMEGAW = REAL(DL)
      OMEGAZ = REAL(DTERM * SIND(DBLE(LAT)))

! Transform into forward/starboard/upward aircraft body co-ordinates.

      CALL C_INS1_TRANS_ACCL(OMEGAN, OMEGAW, OMEGAZ, RROL, RPIT, RHDG,
     &      OMEGAF, OMEGAS, OMEGAU)

! Subtract this angular frequency vector from that reported by the INU
! in forward/starboard/downward aircraft body co-ordinates.

      RP1 = RP - OMEGAF
      RQ1 = RQ - OMEGAS
      RR1 = RR + OMEGAU

! Error in the roll rate and yaw rate if the pitch angle is within 2^(-22)
! of +/- 90 degrees.

      IF (ABS(ABS(RPIT) - 90.0) .LT. 2.3841858E-7) THEN
         RRR = 0.0
         RYR = 0.0
      ELSE
         CR = COSD(RROL)
         SR = SIND(RROL)
         TERM = RQ1 * SR + RR1 * CR

         RRR = RP1 + TERM * TAND(RPIT)
         RYR = TERM / COSD(RPIT)
      END IF
      RPR = RQ1 * CR - RR1 * SR
      RETURN
```

(continues on next page)

(continued from previous page)

```
      END
C*****************************************************************************
      SUBROUTINE C_INS1_TRANS_ACCL(RAX,RAY,RAZ,RROL,RPIT,RHDG,RAF,RAS,
     -    RAU)
C
C Transforms accelerations from the platform (navigation) frame to the
C aircraft frame.  Uses the transpose of the Direction Cosine Matrix defined
C in SNU 84-1 Rev D, section 6.5.2.
C
C CHANGES          V1.01  24-07-98  G.W. Inverarity
C                  Entries RT(2,1), RT(3,1) corrected.
C
CDEC$ IDENT 'V1.01'
      REAL*4 RT(3,3)                    !Full transformation matrix

      SA=SIND(RHDG)                     !Compute sines and cosines
      CA=COSD(RHDG)
      SP=SIND(RPIT)
      CP=COSD(RPIT)
      SR=SIND(RROL)
      CR=COSD(RROL)
      RT(1,1)=CA*CP                     !Load matrix
      RT(2,1)=CA*SP*SR-SA*CR
      RT(3,1)=CA*SP*CR+SA*SR
      RT(1,2)=-SA*CP
      RT(2,2)=-SA*SP*SR-CA*CR
      RT(3,2)=CA*SR-SA*SP*CR
      RT(1,3)=SP
      RT(2,3)=-CP*SR
      RT(3,3)=-CP*CR
      RAF=RT(1,1)*RAX+RT(1,2)*RAY+RT(1,3)*RAZ !Transform accelerations
      RAS=RT(2,1)*RAX+RT(2,2)*RAY+RT(2,3)*RAZ
      RAD=RT(3,1)*RAX+RT(3,2)*RAY+RT(3,3)*RAZ
      RAU=-RAD                          !Convert down to up
      RETURN
      END

!+    Rates of change of latitude and longitude with time.
!
!     *********************** COPYRIGHT *************************
!     Crown Copyright 2002, Met Office. All rights reserved.
!     *********************** COPYRIGHT *************************
!
!     Subroutine interface:

      SUBROUTINE RLATLONG(LAT, ALT, VN, VE, RL, RLAMBDA, RLAT, RLONG)

CDEC$ IDENT 'V1.00'

      IMPLICIT NONE

!     Description:
!     Computes the rates of change of latitude and longitude with time.
!
!     Method:
!     The rate of change of latitude L and longitude lambda with time are,
!     respectively,
```

(continues on next page)

(continued from previous page)

```
!
!     d L / d t = vn / (rL + h),
!     d lambda / d t = ve / (rlambda + h) / cos(L),
!
!     where
!
!     rL = R0 (1 - e^2) / (1 - e^2 sin(L)^2)^(3/2),
!     rlambda = R0 / sqrt(1 - e^2 sin(L)^2),
!
!     and vn and ve are the components of the aircraft's velocity in the
!     Earth frame in the northward and eastward directions,
!     respectively, h is the aircraft's altitude above the WGS-84
!     ellipsoid, whose radius of curvature and eccentricity are,
!     respectively, R0 and e.
!
!     Note that LAT is REAL*8 and the radii of curvature are returned
!     to allow this routine to be used by INTTEXT:KF_EXTR.FOR as well.
!
!     Current Code Owner: W.D.N. Jackson
!
!     History:
!     Version    Date     Comment
!
!        1.00    23/09/02 Original code. (G.W. Inverarity)
!
!     Code Description:
!     FORTRAN 77 with extensions recommended in the Met Office F77
!     Standard.
!
!     Scalar arguments with INTENT(IN):

      REAL*8      LAT                  ! Latitude (deg)
      REAL*4      ALT                  ! Altitude (m)
      REAL*4      VN                   ! Northward velocity component (m/s)
      REAL*4      VE                   ! Eastward velocity component (m/s)

!     Scalar arguments with INTENT(OUT):

      REAL*8      RL                   ! Meridional radius of curvature (m)
      REAL*8      RLAMBDA              ! Azimuthal radius of curvature (m)
                                       ! / COS(LAT)
      REAL*8      RLAT                 ! Latitude rate of change (rad/s)
      REAL*8      RLONG                ! Longitude rate of change (rad/s)

!     Local Parameters:

      REAL*8     F                     ! WGS-84 flattening
      PARAMETER(F=1.0D0/298.257223563D0)
      REAL*8     EPSQ                  ! WGS-84 eccentricity squared
      PARAMETER(EPSQ=F*(2.0D0-F))
      REAL*8     R0                    ! WGS-84 equatorial radius (m)
      PARAMETER(R0=6.378137D6)

!     Local Scalars:

      REAL*8      TERM                 ! 1 - EPSQ SIN(LAT)^2.
```

(continues on next page)

(continued from previous page)

```
!-    End of header

      TERM = 1.0D0 - EPSQ * SIND(LAT) ** 2
      RL = R0 * (1.0D0 - EPSQ) / TERM ** 1.5D0
      RLAMBDA = R0 / SQRT(TERM)
      RLAT = DBLE(VN) / (RL + DBLE(ALT))
      RLONG = DBLE(VE) / (RLAMBDA + DBLE(ALT)) / COSD(LAT)
      RETURN
      END
```

8.10 c_lwc.for

```
C
C ROUTINE             C_LWC   SUBROUTINE FORT VAX   [C_LWC.FOR]
C
C PURPOSE             To calibrate DRS parm 42 to tardis parm 535 (LWC)
C
C DESCRIPTION              The Liquid Water Content (LWC) is a four hertz
C                     parameter. It requires the True Air Speed (Parm 517),
C                     True De_iced Temperature (parm 520) and Static
C                     Pressure (parm 576). All these derived parameters
C                     (517, 520, 576) are at 32 Hertz. So for each quarter
C                     point of the LWC requires a sample of eight of
C                     the derived paramters to be averaged. This is done using
C                     only good data points. If there are not eight samples but
C                     more than one then the flag for the derived LWC is set to 1.
C                     If the frequency of the DRS parm (42) is not equal to 4
C                     then no values are calculated and all four points of the
C                     LWC are set to -9999.0, with a flag of 3. If a point cannot
C                     be calculated then the value of it is set to -9999.0 with
C                     a flag value of 3. If the instrument is saturated then the
C                     flag value is 1. If the derived value for the LWC falls out
C                     of the bounds of -10 to 10 then the flag is set to 2.
C
C VERSION              1.02 17-01-96 D Lauchlan
C
C ARGUMENTS          IRAW(64,512) I*4   IN   Raw data for the parameters
C                     IFRQ(512)    I*4   IN   Frequencies of the data
C                     RCONST(64)   R*4   IN   Constants required by routine,(1-28)
C                     RDER(64,1024)R*4   OUT  Tardis parameters
C
C COMMON             None.
C
C SUBPROGRAMS             ISETFLG (linked automatically)
C
C FILES                     None.
C
C REFERENCES            MRF2 Specification for Total Water Hygrometer 4 Dec 1989
C                     Ouldridge Feb 1982
C                    Johnson 1979
C
C CHANGES            110190 Documentational changes only         M.Glover
C                    v 1.02 17-01-96 D Lauchlan
C                    Unused parameters removed
```

(continues on next page)

(continued from previous page)

```
C
C                   V1.03  27/09/02  W.D.N.JACKSON
C                   Changed to include handling of 16 bit data from the new
C                   DRS.
C################################################################################

        SUBROUTINE C_LWC(IRAW, IFRQ, RCONST, RDER)
CDEC$ IDENT 'V1.03'
        INTEGER*4 IRAW(64,512), IFRQ(512)

        REAL*4 RCONST(64), RDER(64, 1024)

C        The frequencies of the derived parameters passed into this module
C        may change. That is why IFRQ_*** has been set up. Here is a table of
C        what values of it corresponds to what frequency;
C
C                        Frq        IFRQ_***
C                         4           0
C                        16         1
C                        32         7
C                        64        15
C
        DATA IFRQ_TAS/7/, IFRQ_TDT/7/, IFRQ_PRESSURE/7/

C        Calibrate the Johnson_Williams Liquid Water Content Probe - DRS
C        parameter 42, sample rate 4 Hz. This is to be put into g kg-1.
C        This uses the elements of RCONST from 1 to 2.

        IF (IFRQ(42).EQ.4) THEN  ! check the frequency.

                N_TAS=1
                N_TDT=1
                N_PRE=1

                DO IS=1, IFRQ(42)  ! for each sample
                        IRAW_FLAG=0
                        IF(IRAW(IS,42).EQ.0.OR.IRAW(IS,42)
     &                       .EQ.'FFFF'X) IRAW_FLAG=3
                                ICHECK=1
                                ICHECK_2=1
                        TAS=0.0
                        P=0.0
                        TDT=0.0

C                        See if all the const are there,if not set the flag to 3
                              DO I=1,2,1
                               IF (ITSTFLG(RCONST(I)).GT.2) THEN
                                       ICHECK=ICHECK+1
                               END IF
                        END DO

                        SUM=0  ! Reset the sum.
                        ICOUNT=0

C                        Find the average of the TAS.
                         DO INC=N_TAS, N_TAS+IFRQ_TAS
                               IFLAG=ITSTFLG(RDER(INC, 517))
```

(continues on next page)

(continued from previous page)

```
                                IF (IFLAG.LT.1) THEN
                                           SUM=SUM+RDER(INC, 517)
                                        ICOUNT=ICOUNT+1
                                END IF
                            END DO

C                        Reset the starter for the do loop to be the old start
C                        point plus the incremental for the TAS frequency plus
C                        one.
                        N_TAS=N_TAS+IFRQ_TAS+1

                        IF (ICOUNT.EQ.0) THEN
C                                No good points.
                                ICHECK=-9999
                        ELSE
                                IF (ICOUNT.NE.IFRQ_TAS+1) THEN
                                        ICHECK_2=99
                                END IF
                                       TAS=SUM/FLOAT(ICOUNT)
                        END IF

                        SUM=0.0  ! Reset the sum.
                        ICOUNT=0

C                        Find the average of the true de_iced temp.
                        DO INC=N_TDT, N_TDT+IFRQ_TDT
                                IFLAG=ITSTFLG(RDER(INC, 520))
                                IF (IFLAG.LT.1) THEN
                                               SUM=SUM+RDER(INC, 520)
                                        ICOUNT=ICOUNT+1
                                END IF
                        END DO

                        N_TDT=N_TDT+IFRQ_TDT+1
                        IF (ICOUNT.EQ.0) THEN
                                ICHECK=-9999
                        ELSE
                                IF (ICOUNT.NE.IFRQ_TDT+1) THEN
                                        ICHECK_2=99
                                END IF
                                       TDT=SUM/FLOAT(ICOUNT)
                        END IF

                        SUM=0  ! Reset the sum.
                        ICOUNT=0

C                        Find the static pressure average.
                        DO INC=N_PRE, N_PRE+IFRQ_PRESSURE
                                IFLAG=ITSTFLG(RDER(INC, 576))

C                                Only use good data, namley of flag zero.
                                IF (IFLAG.LT.1) THEN
                                        SUM=SUM+RDER(INC, 576)
                                        ICOUNT=ICOUNT+1
                                END IF
                        END DO
```

(continues on next page)

(continued from previous page)

```
                        N_PRE=N_PRE+IFRQ_PRESSURE+1

                        IF (ICOUNT.EQ.0) THEN
                                ICHECK=-9999
                        ELSE
                                IF (ICOUNT.NE.IFRQ_PRESSURE+1) THEN
                                        ICHECK_2=99
                                END IF
                                P=SUM/ICOUNT
                        END IF

                        IF (TDT.LT.10.0) THEN
                                ICHECK=-9999
                        ELSE
                                RHO=(0.3484*P)/TDT
                        END IF

C                        Make sure that division by one does not happen.
                        IF ((TAS.LT.1).OR.(RHO.LT.1E-08)) THEN
                                ICHECK=-9999
                        END IF

C                        ICHECK will be more than one if any of the constants
C                        are missing, or the true air speed is zero, or rho
C                        is zero.
                        IF (ICHECK.EQ.1.and.icheck_2.eq.1) THEN
                                IF(IRAW(IS,42).EQ.0.OR.IRAW(IS,42)
     &                                 .EQ.'FFFF'X) IFLAG=3
C                        ICHECK_2 will be diffrent than 1 if there are not eight
C                        samples for the true de-iced temp or pressure.
                        ELSE IF (ICHECK_2.EQ.99.and.icheck.eq.1) THEN
                                IFLAG=1
                        ELSE
                                IFLAG=3
                        END IF

C                        If the flag of the raw data is less than three, then
C                        convert the raw data into derived data. This is done
C                        using ;
C
C                               LWC=  (A+Bx)*77.2
C                                     ------------
C                                        TAS*RHO
C
C
C                               RHO=0.3484*STATIC_PRESSURE
C                                   ----------------------
C                                       TRUE_DE_ICED_TEMP
C
C

                                     IF (IFLAG.LT.3) THEN
                                RAW=FLOAT(IRAW(IS,42))
                                RDER(IS, 535)=((RCONST(1)+RCONST(2)*RAW)
     1                                                  *77.2)/(TAS*RHO)
                             ELSE
```

(continues on next page)

(continued from previous page)

```
C                                If the flag is three or above, set the
C                                derived data to -9999.0.
                                RDER(IS, 535)=-9999.0
                            END IF

C                        If the derived data is outside the bounds but not
C                        -9999.0, then set the flag to two.
                        IF (((RDER(IS, 535).LT.-8.0).OR.
     1                                        (RDER(IS, 535).GT.8.0)).AND.
     2                                            (RDER(IS, 535).GT.-9000.0)) THEN
                                CALL ISETFLG(RDER(IS, 535), 2)

C                        If J/W > 300/TAS  set J/W FLAG=1
C                        300/tas = instrument saturation value.
C                        Ref Ouldridge Feb 1982, Johnson 1979

                        ELSE IF        ((RDER(IS, 535).GE.-8.0).AND.
     1                                        (RDER(IS, 535).LE.8.0).AND.
     3                                                 (RDER(IS, 535).GT.(300/TAS)))␣
↪THEN
                                    CALL ISETFLG(RDER(IS, 535), 1)

                        ELSE

C                                The derived data is within the limits then
C                                set the flag to that of the raw data. If the
C                                data is -9999.0 the flag will be three.
                                 CALL ISETFLG(RDER(IS, 535), IFLAG)
                            ENDIF

                        IF(IRAW_FLAG.GT.ITSTFLG(RDER(IS, 535))) THEN
                                CALL ISETFLG(RDER(IS, 535), IRAW_FLAG)
                        END IF

                 END DO    ! For the frequency of the LWC.
        ELSE

C                The data has not got the right frequency.
                DO IS=1,4
                        RDER(IS, 535)=-9999.0
                        CALL ISETFLG(RDER(IS,535),3)
                END DO
        ENDIF


        RETURN

        END
```

8.11 c_nephl1.for

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!         ROUTINE      C_NEPHL1
```

(continues on next page)

(continued from previous page)

```
!
!         PURPOSE      Calibrates the parameters from the TSI 3 wavelength
!                      Nephelometer.
!
!         DESCRIPTION  The nephelometer output are subject to calibrations
!                      internal to the instrument plus the normal MILLIE/DRS
!                      calibrations.
!
!                      All raw parameters are digital conversions of the input
!                      voltage. The digital values are converted using a
!                      linear fit then the nephelometer internal cals are
!                      applied to obtain the output derived values.
!                      Parameters 175, 176 & 183 are linear thus
!
!                           output = a + b * v /vfs
!
!                      where a and b are nephelometer internal constants,
!                      v the derived voltage and vfs the full scale voltage.
!                      Scattering parameters 177 through 182 are logarithmic
!                      so
!
!                          output = 10**((v/b) - a) - c
!
!                      a, b and c are nephelometer internal constants, v
!                      the derived voltage.
!
!                      Parameter 184, nephelometer status, is the analog output
!                      of a 4 bit D to A converter. The LSB corresponds to
!                      0.625V.
!
!         VERSION      1.00  D.P.Briggs
!
!         SUBROUTINES  ISETFLG
!
!         FILES        NONE
!
!         REFERENCES   Nephelometer Instruction Manual
!                      Nephelometer internal technical note
!
!         PARAMETERS   RAW      DERIVED    FREQ   RANGE   UNITS
!  NEPH PRESSURE       175        760      1Hz    0-10V    mb
!  NEPH TEMPERTURE     176        761      1Hz    0-10V    K
!  NEPH BLUE SP        177        762      1Hz    0-10V    /m
!  NEPH GREEN SP       178        763      1HZ    0-10V    /m
!  NEPH RED SP         179        764      1Hz    0-10V    /m
!  NEPH BLUE BSP       180        765      1Hz    0-10V    /m
!  NEPH GREEN BSP      182        766      1Hz    0-10V    /m
!  NEPH RED BSP        181        767      1Hz    0-10V    /m
!  NEPH HUMIDITY       183        768      1Hz    0-5V     %
!  NEPH STATUS         184        769      1Hz    0-10V    bits
!
!         NEPHELOMETER CONSTANT KEYWORDS
!   CALNPRS   I J A B
!   CALNTMP   I J A B
!   CALNBTS   I J A B C    NOTE : I & J are multiplexer calibrations.
!   CALNGTS   I J A B C           A, B & C are instrument internal cals.
!   CALNRTS   I J A B C
```

(continues on next page)

(continued from previous page)

```
!   CALNBBS  I J A B C
!   CALNGBS  I J A B C
!   CALNRBS  I J A B C
!   CALNHUM  I J A B
!   CALNSTS  I J
!
!         CHANGES
!         V1.01  12/05/97  W.D.N.JACKSON
!                Miscellaneous bug fixes, the most serious being the incorrect
!                calculation of the red backscattering coefficient, and a
!                tendency to crash with floating overflows.
!         V1.02  29/05/97  W.D.N.JACKSON
!                Changed to reflect fact that Red BS comes in on 181 and green
!                on para 182.
!         V1.03  19/06/98  W.D.N.JACKSON
!                Changed to reflect fact that bit 3 of the status word is 0
!                (not 1) when on calibrate.
!         V1.04  01/09/02  W.D.N.JACKSON
!                Small changes to handle new 16 bit DRS.  Also status parameter
!                is now calibrated.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      SUBROUTINE C_NEPHL1(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.04'
      IMPLICIT NONE
      INTEGER   IFLAG,IPARM,ISTAT
      REAL      RVOLT
      INTEGER*4 IRAW(64,512),IFRQ(512)
      REAL*4    RCONST(64),RDER(64,1024)
!
! initialise output values to 0.0 flagged 3
!
      IFLAG = 3
      DO IPARM = 760,769
        RDER(1,IPARM)=0.0
        CALL ISETFLG(RDER(1,IPARM),IFLAG)
      ENDDO
!
! Check all data valid.  Return if not.
!
      DO IPARM = 175,184
        IF(IRAW(1,IPARM).EQ.0) RETURN
        IF(IRAW(1,IPARM).EQ.'FFFF'X) RETURN
      ENDDO
!
!   184/769 1Hz   NEPH STATUS
! input is voltage proportional to a 4 bit counter at 0.625V (256 bits) per unit
! Status word bits have the following meanings (it is not clear if the status
! word is valid when the lamp is not working):
!   Bit 0 set - Backscatter on
!   Bit 1 set - Chopper on
!   Bit 2 set - Lamp off
!   Bit 3 set - Not on calibrate
!
      IFLAG=0
      RVOLT=FLOAT(IRAW(1,184))*RCONST(44)+RCONST(43)
      ISTAT=NINT(RVOLT/0.625)
      IF(ISTAT.LT.0.OR.ISTAT.GT.15) RETURN
```

(continues on next page)

(continued from previous page)

```
      RDER(1,769) = FLOAT(ISTAT)
      CALL ISETFLG(RDER(1,769),IFLAG)
!
!   175/760 1Hz   NEPH PRESSURE
      RVOLT = RCONST(1) + RCONST(2) * FLOAT(IRAW(1,175))
      RDER(1,760) = RCONST(4) * RVOLT/10.0 + RCONST(3)
      CALL ISETFLG(RDER(1,760),IFLAG)
!
!   176/761 1Hz   NEPH TEMPERATURE
      RVOLT = RCONST(5) + RCONST(6) * FLOAT(IRAW(1,176))
      RDER(1,761) = RCONST(8) * RVOLT/10.0 + RCONST(7)
      CALL ISETFLG(RDER(1,761),IFLAG)
!
!   183/768 1Hz   NEPH HUMIDITY
      RVOLT = RCONST(39) + RCONST(40) * FLOAT(IRAW(1,183))
      RDER(1,768) = RCONST(42) * RVOLT/10.0 + RCONST(41)
      CALL ISETFLG(RDER(1,768),IFLAG)
!
! set flags for scattering coeffs:
!  lamp or reference chopper off, flag 3.
!  valve position in calibrate mode flag 2.
!  if not backscatter shutter on flag backscatter coeffs with 3
      IF (BTEST(ISTAT,1) .AND. .NOT.BTEST(ISTAT,2)) THEN
        IF (.NOT.BTEST(ISTAT,3)) IFLAG = 2
!
!   177/762 1Hz   NEPH BLUE SP
        RVOLT = RCONST(9) + RCONST(10) * FLOAT(IRAW(1,177))
        RDER(1,762) = 10 ** ((RVOLT/RCONST(12)) - RCONST(11)) -
     &               RCONST(13)
        CALL ISETFLG(RDER(1,762),IFLAG)
!
!   178/763 1HZ   NEPH GREEN SP
        RVOLT = RCONST(14) + RCONST(15) * FLOAT(IRAW(1,178))
        RDER(1,763) = 10 ** ((RVOLT/RCONST(17)) - RCONST(16)) -
     &               RCONST(18)
        CALL ISETFLG(RDER(1,763),IFLAG)
!
!   179/764 1Hz   NEPH RED SP
        RVOLT = RCONST(19) + RCONST(20) * FLOAT(IRAW(1,179))
        RDER(1,764) = 10 ** ((RVOLT/RCONST(22)) - RCONST(21)) -
     &               RCONST(23)
        CALL ISETFLG(RDER(1,764),IFLAG)
!
        IF (BTEST(ISTAT,0)) THEN  !backscatter shutter on
!   180/765 1Hz   NEPH BLUE BSP
          RVOLT = RCONST(24) + RCONST(25) * FLOAT(IRAW(1,180))
          RDER(1,765) = 10 ** ((RVOLT/RCONST(27)) - RCONST(26)) -
     &                 RCONST(28)
          CALL ISETFLG(RDER(1,765),IFLAG)
!
!   182/766 1Hz   NEPH GREEN BSP
          RVOLT = RCONST(29) + RCONST(30) * FLOAT(IRAW(1,182))
          RDER(1,766) = 10 ** ((RVOLT/RCONST(32)) - RCONST(31)) -
     &                 RCONST(33)
          CALL ISETFLG(RDER(1,766),IFLAG)
!
!   181/767 1Hz   NEPH RED BSP
```

(continues on next page)

(continued from previous page)

```
          RVOLT = RCONST(34) + RCONST(35) * FLOAT(IRAW(1,181))
          RDER(1,767) = 10 ** ((RVOLT/RCONST(37)) - RCONST(36)) -
     &                  RCONST(38)
          CALL ISETFLG(RDER(1,767),IFLAG)
        ENDIF
      ENDIF
!
      RETURN
      END
```

8.12 c_nevz.for

```
C
C ROUTINE          C_NEVZ SUBROUTINE FORTVAX
C
C PURPOSE          Produces calibrated Nevzorov parameters
C
C DESCRIPTION      Calculates liquid and total water values for the Nevzorov
C                  together with reference and collector voltages, using the
C                  equations supplied with the unit, namely:
C
C                  Water content = V**2/U/L/SR
C
C                  where V is the output voltage (V)
C                        U is the True air speed (m/s)
C                        L is the energy expended in heating and evaporating
C                             the water, for which a value of 2589 J/g is used
C                        SR is the product of the sensor area and the resistanc
C                           of the collector sensor at the chosen temperature.
C
C                  Flagging:
C
C                  If on Calibrate, as indicated by bit 1 or 2 in the signal
C                  register being set then the data is flagged with a 2.
C                  Otherwise the data carries the flag of the True Airspeed
C                  parameter.
C
C VERSION          1.00  18/01/99  W.D.N.JACKSON
C
C ARGUMENTS
C                  Constants:
C                  RCONST(1)   CALNVLW X0
C                  RCONST(2)   CALNVLW X1
C                  RCONST(3)   CALNVLR X0
C                  RCONST(4)   CALNVLR X1
C                  RCONST(5)   CALNVLC X0
C                  RCONST(6)   CALNVLC X1
C                  RCONST(7)   CALNVTW X0
C                  RCONST(8)   CALNVTW X1
C                  RCONST(9)   CALNVTR X0
C                  RCONST(10)  CALNVTR X1
C                  RCONST(11)  CALNVTC X0
C                  RCONST(12)  CALNVTC X1
C                  RCONST(13)  CALRSL X0     RS Value at T0
```

(continues on next page)

(continued from previous page)

```
C                   RCONST(14)  CALRSL T0
C                   RCONST(15)  CALRST X0     RS value at T0
C                   RCONST(16)  CALRST T0
C
C                   Inputs:
C                   NVLW Nevzorov Liquid Water                  Para 208  8 Hz
C                   NVLR Nevzorov Liquid Reference              Para 209  8 Hz
C                   NVLC Nevzorov Liquid Collector              Para 210  8 Hz
C                   NVTW Nevzorov Total Water                   Para 211  8 Hz
C                   NVTR Nevzorov Total Reference               Para 212  8 Hz
C                   NVTC Nevzorov Total Collector               Para 213  8 Hz
C                   SREG Signal register [bits 1-2]             Para  27  2 Hz
C                   TAS  True airspeed                          Para 517 32 Hz
C
C                   Outputs:
C                   NVLW Nevzorov Liquid Water     [gm-3]        Para 602 8 Hz
C                   NVLR Nevzorov Liquid Reference [V]           Para 603 8 Hz
C                   NVLC Nevzorov Liquid Collector [V]           Para 604 8 Hz
C                   NVTW Nevzorov Total Water      [gm-3]        Para 605 8 Hz
C                   NVTR Nevzorov Total Reference  [V]           Para 606 8 Hz
C                   NVTC Nevzorov Total Collector  [V]           Para 607 8 Hz
C
C SUBPROGRAMS       ITSTFLG          Examines bits 16,17 for flags
C                   ISETFLG          Sets flag bits 16,17 = 0 --> 3
C
C REFERENCES
C
C CHANGES            V1.01  27/09/02  W.D.N.JACKSON
C                    Changed to include handling of 16 bit data from the new
C                    DRS.
C                    V1.02  13/11/06
C                    Signal register bits 1 and 2 swapped to be the correct way
C                    round 1=TW, 2=LW
C                    V1.03  12/12/06
C                    Voltage flags explicitly set to zero
C
C*******************************************************************************
      SUBROUTINE C_NEVZ(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.03'
      INTEGER*4 IRAW(64,512),IFRQ(512)
      REAL*4    RCONST(64),RDER(64,1024)

      RERR=0.                           !Default error return
      CALL ISETFLG(RERR,3)              ! is 0 flagged 3
      DO I=1,8                          !For each sample
        RTAS=RDER((I-1)*4+1,517)
        ITASFLG=ITSTFLG(RTAS)           !Note TAS flag
        CALL ISETFLG(RTAS,0)            !Clear TAS flag
        IF(RTAS.LE.0) ITASFLG=3         !Invalid TAS
        RDER(I,602)=RERR                !Default return
        RV=RCONST(1)+RCONST(2)*IRAW(I,208) !Compute voltage
        IF(ITASFLG.LT.3) THEN           !If TAS valid
          RDER(I,602)=RV*RV/RTAS/2589/RCONST(13) !Compute liquid water
          CALL ISETFLG(RDER(I,602),0)
          IF(.NOT.BTEST(IRAW(1,27),2)) CALL ISETFLG(RDER(I,602),2) !Flag 2 if on cal
        END IF
        RDER(I,603)=RCONST(3)+RCONST(4)*IRAW(I,209) !Calibrate voltages
```

(continues on next page)

(continued from previous page)

```
        RDER(I,604)=RCONST(5)+RCONST(6)*IRAW(I,210)
        CALL ISETFLG(RDER(I,603),0)
        CALL ISETFLG(RDER(I,604),0)
        RDER(I,605)=RERR               !Repeat the processing for total water
        RV=RCONST(7)+RCONST(8)*IRAW(I,211)
        IF(ITASFLG.LT.3) THEN
          RDER(I,605)=RV*RV/RTAS/2589/RCONST(15)
          CALL ISETFLG(RDER(I,605),0)
          IF(.NOT.BTEST(IRAW(1,27),1)) CALL ISETFLG(RDER(I,605),2)
        END IF
        RDER(I,606)=RCONST(9)+RCONST(10)*IRAW(I,212)
        RDER(I,607)=RCONST(11)+RCONST(12)*IRAW(I,213)
        CALL ISETFLG(RDER(I,606),0)
        CALL ISETFLG(RDER(I,607),0)
      END DO
C
      RETURN
      END
```

8.13 c_nox.for

```
C
C ROUTINE        C_NOX SUBROUTINE FORTVAX
C
C PURPOSE        A subroutine to calculate Nitrogen monoxide, Nitrogen dioxide
C                and NOx measured by the TECO 42 NOx analyser.
C
C DESCRIPTION        The NOx analyser outputs three measurements, NO, NO2 and NOx.
C                These are input to the program as DRS bits, and converted
C                into PPB by multiplying the DRS bits by a calibration factor.
C
C
C TO COMPILE        $FORT C_NOX
C
C VERSION        1.00  28 Sept. 1998          I. Hawke
C               1.01  23 June. 1999     I. Hawke     5ppb Offset included
C               1.02  07 Mar   2000     I. Hawke     Offset removed
C               1.03  07 Mar   2000     I. Hawke     5ppb offset included
C               1.04  29 Mar   2000     I. Hawke     New Conversion Algorithm
C               1.05  30 Mar   2000     I. Hawke     Flow Testing Added
C               1.06  02 Oct   2002     N. Jackson   Modified for 16 bit DRS
C               1.07  27 Oct   2005     D.Tiddeman   New constants for flagging
C                                                    low airspeeds or out of
C                                                    range values.
C               1.08  05 Dec  2005      D.Tiddeman   No flag 2 for negative
C
C ARGUMENTS        IRAW(1,203) - on entry contains the raw NO signal
C                IRAW(1,204) - on entry contains the raw NO2 signal
C                IRAW(1,205) - on entry contains the raw NOx signal
C               IRAW(1,114) - on entry contains ozone flow signal
C               RCONST(1,2,3,4) XO and X1 voltage cal for NO, v to ppb, ppb offs
C               RCONST(5,6,7,8) same for NO2
C               RCONST(9,10,11,12) same for NOx
C               RCONST(13,14) X0 and X1 voltage cal for Ozone flow
```

(continues on next page)

(continued from previous page)

```
C                RDER(1,770) - on exit contains the derived NO signal
C                RDER(1,771) - on exit contains the derived NO2 signal
C                RDER(1,772) - on exit contains the derived NOx signal
C
C*******************************************************************************
      SUBROUTINE C_NOX(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.08'
      IMPLICIT NONE
      INTEGER*4 IRAW(64,1024),IFRQ(512)
      INTEGER   IFLG,IFLG1,ITSTFLG,IS
      REAL*4 NO,NO2,NOX,OZF,RERR
      REAL*4 RCONST(64),RDER(64,1024)
C
C
C Set default values for output
C
      RERR=0.
      CALL ISETFLG(RERR,3)
      RDER(1,770)=RERR
      RDER(1,771)=RERR
      RDER(1,772)=RERR

C     Copy across raw signals
C
      NO=FLOAT(IRAW(1,203))
      NO2=FLOAT(IRAW(1,204))
      NOX=FLOAT(IRAW(1,205))
      OZF=FLOAT(IRAW(1,114))
C
C     Convert TECO NOX DRS signals first to voltage, than apply voltage to
C     ppb conversion, then subtract instrument offset which ensures signal
C     voltage doesn't go negative.
C
      NO=(RCONST(1)+NO*RCONST(2))*RCONST(3)-RCONST(4)
      NO2=(RCONST(5)+NO2*RCONST(6))*RCONST(7)-RCONST(8)
      NOX=(RCONST(9)+NOX*RCONST(10))*RCONST(11)-RCONST(12)
C
C Convert ozone flow to voltage
C
      OZF=RCONST(13)+OZF*RCONST(14)
C
C Do flagging
C
      IF(IRAW(1,114).EQ.0) RETURN
      IF(IRAW(1,114).EQ.'FFFF'X) RETURN
      IF(OZF.LE.0.) RETURN             !Reject data if pump off or not recorded
C
      IFLG1=0
      IF(ITSTFLG(RCONST(17)).EQ.0)THEN
        DO IS=1,32
          IF(IRAW(IS,223).LT.62*RCONST(17)) IFLG1=1
        ENDDO
      ENDIF
      IFLG=IFLG1
      IF(IRAW(1,203).EQ.0) IFLG=3
      IF(IRAW(1,203).EQ.'FFFF'X) IFLG=3
      IF(ITSTFLG(RCONST(15)).EQ.0.AND.ITSTFLG(RCONST(16)).EQ.0)THEN
```

(continues on next page)

(continued from previous page)

```
        IF(NO.LT.RCONST(15).OR.NO.GT.RCONST(16))IFLG=3
      ENDIF
      CALL ISETFLG(NO,IFLG)
      RDER(1,770)=NO
C
      IFLG=IFLG1
      IF(IRAW(1,204).EQ.0) IFLG=3
      IF(IRAW(1,204).EQ.'FFFF'X) IFLG=3
      IF(ITSTFLG(RCONST(15)).EQ.0.AND.ITSTFLG(RCONST(16)).EQ.0)THEN
        IF(NO2.LT.RCONST(15).OR.NO2.GT.RCONST(16))IFLG=3
      ENDIF
      CALL ISETFLG(NO2,IFLG)
      RDER(1,771)=NO2
C
      IFLG=IFLG1
      IF(IRAW(1,205).EQ.0) IFLG=3
      IF(IRAW(1,205).EQ.'FFFF'X) IFLG=3
      IF(ITSTFLG(RCONST(15)).EQ.0.AND.ITSTFLG(RCONST(16)).EQ.0)THEN
        IF(NOX.LT.RCONST(15).OR.NOX.GT.RCONST(16))IFLG=3
      ENDIF
      CALL ISETFLG(NOX,IFLG)
      RDER(1,772)=NOX
C

      RETURN
      END
```

8.14 c_ozone1.for

```
C
C ROUTINE        C_OZONE1 SUBROUTINE FORTVAX
C
C PURPOSE        A subroutine to calculate the ozone mixing ratio.
C
C DESCRIPTION   Calibration routine for TECO OZONE
C               Fourth order fit for temperature transducer
C               Linear fit for pressure transducer
C               Signal 10V =1000 ppb ozone corrected by pressure and temperature
C
C TO COMPILE        $FORT C_OZONE1
C
C VERSION            1.00  30th Aug 1996  D.Tiddeman   Module Written
C               1.01  12th Sep 1999  I. Hawke     O3 Smoothing Algorithm
C               1.02  2 Oct 2002     N. Jackson   Convert to work on new DRS
C               1.03  11 Feb 2005   D.Tiddeman   No longer does P+T correction
C                1.04  Unknown       D.Tiddeman   RVSM, min and max flags addeds
C               1.05  31 Jan 2007   R Purvis     Rewritten to take out P, T
C                                                corrections
C                                                 Flow commented out
C
C ARGUMENTS         IRAW(1,100) - on entry contains the raw ozone signal
C                IRAW(1,114) - on entry contains the raw ozone flow
C                IRAW(1,223) - on entry contains the raw rvsm airspeed
C                RCONST(X)
```

(continues on next page)

(continued from previous page)

```
C                RDER(1,574) - on exit contains the derived ozone mixing ratio
C
C
C*******************************************************************************
      SUBROUTINE C_OZONE1(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.05'
      IMPLICIT NONE
      INTEGER*4 IRAW(64,512),IFRQ(512)
      INTEGER   IFLG,ITSTFLG,POINT,IS
      REAL*4 OZSIG
      REAL*4 RCONST(64),RDER(64,1024),FLOW
C
C        INITIALISE OZONE CHANNEL TO ZERO
C
      RDER(1,574)=0.0
C
C     SET UNUSED CHANNELS TO -9999 AND FLAG AS 3
C
C        RDER(1,691) = -9999
C      CALL ISETFLG(RDER(1,691),3)
C        RDER(1,692) = -9999
C      CALL ISETFLG(RDER(1,692),3)
C      RDER(1,693) = -9999
C      CALL ISETFLG(RDER(1,693),3)

C     SKIP PROCESSING IF ANY RAW DATA INVALID
C
      IFLG=0
      IF((IRAW(1,100).EQ.0).OR.(IRAW(1,100).EQ.'FFFF'X)) IFLG=3
C     IF(IRAW(1,114).EQ.0) IFLG=3
C     IF(IRAW(1,114).EQ.'FFFF'X) IFLG=3
C     FLOW=RCONST(16)+IRAW(1,114)*RCONST(17) !Flow in volts
C     IF(FLOW.LE.0.) IFLG=3
      IF(IFLG.EQ.3) GOTO 100
C
C     CALCULATE OZONE MIXING RATIO
C
      ozsig=FLOAT(IRAW(1,100))            ! ozsig becomes ozone signal
      ozsig=rconst(1)+ozsig*rconst(2)    ! convert drs bits to volts
      OZSIG=OZSIG*RCONST(4)+RCONST(3)    ! IN PPB
      RDER(1,574) = OZSIG
C
C         INSERT FLAGS FOR ON GROUND USING RVSM(223)VALUE AND MIN AIR SPEED(21)
c         MIN VALUE (18), OUTSIDE CALIBRATION RANGE(19, 300PPBv) AND MAX VALUE (20,␣
↪500PPBv)
C
      IF(IFLG.EQ.0)THEN
      IF(ITSTFLG(RCONST(21)).EQ.0)THEN
        DO IS=1,32
          IF(IRAW(IS,223).LT.62*RCONST(21)) IFLG=1
        ENDDO
      ENDIF
      IF((ITSTFLG(RCONST(18)).EQ.0.AND.ITSTFLG(RCONST(19))).EQ.0
     &   .AND.ITSTFLG(RCONST(20)).EQ.0)THEN
        IF(OZSIG.GT.RCONST(19))IFLG=2
        IF(OZSIG.LT.RCONST(18).OR.OZSIG.GT.RCONST(20))IFLG=3
      ENDIF
```

(continues on next page)

(continued from previous page)

```
      ENDIF
C
C     FLAG OZONE SIGNAL
C
  100 CALL ISETFLG(RDER(1,574),IFLG)
      RETURN
      END
```

8.15 c_press1.for

```
!
! ROUTINE          C_PRESS1 SUBROUTINE FORTVAX
!
! PURPOSE          Calibrates the cabin pressure sensor and the S9 static port.
!
! DESCRIPTION      Apply calibration the combined transducer and DRS
!                  coefficients to DRS parameters 14 and 221 to obtain derived
!                  parameters 579 and 778.  Invalid data is flagged with 3, data
!                  outside limits is flagged with 2.
!
! METHOD           For each DRS parameter to be calibrated:
!                  1. If data is FFFF or FFFE then flag 3
!                  2. Apply the calibration constants
!                  3. Check the results for being within acceptable values.
!                  4. Set data flag bits (16+17) 0: Good data
!                                                1: Data of lower quality
!                                                2: Probably faulty, exceed lims
!                                                3: Data absent or invalid.
!
!                  Flagging - If a value can't be computed, due to missing data
!                  missing constants, divide be zeroes, etc, a value of 0 is
!                  used, flagged with a three.  If a value is outside its
!                  limits for range or rate of change, it is flagged with a two.
!                  If there are no problems with the data it is flagged with 0.
!
! VERSION          1.00  23/07/03  W.D.N.JACKSON
!
! ARGUMENTS        Inputs:
!                    DRS para  14 CABP  1 Hz Cabin pressure
!                        para 221 S9SP 32 Hz S9 static pressure
!
!                  Constants:
!                        RCONST(1 to 3) Para 14 cal constants X0 to X2
!                        RCONST(4 to 6) Para 221 cal constants X0 to X2
!
!                  Outputs:
!                    Derived para 579 CABP mb  1 Hz Cabin pressure
!                            para 778 S9SP mb 32 Hz S9 static pressure
!
!                  Flags:
!                    Missing/corrupt data output as 0 flagged 3.
!                    Out of range data flagged 2.
!
! SUBPROGRAMS      ISETFLG
```

(continues on next page)

(continued from previous page)

```
!
! REFERENCES
!
! CHANGES          V1.00 23/07/03  WDNJ Original version
!                  Note that V1.00 has no application of S9 position errors.
!
```

8.16 c_press.for

```
C
C ROUTINE          C_PRESS SUBROUTINE FORTVAX
C
C PURPOSE          Calibrates static, Pitot-static and cabin pressures.  Derives
C                  pressure height.
C
C DESCRIPTION      Apply calibration coefficients to DRS parameters 8, 9 and 14
C                  to obtain Static Pressure (Para 576), Pitot-static pressure
C                  (Para 577) and Cabin Pressure (Para 579) in mb; also derive
C                  Pressure height (Para 578) in metres.  Parameter 14 does not
C                  have to be present and if absent Para 579 is filled with 0's
C                  flagged with threes.
C
C METHOD           For each DRS parameter to be calibrated:
C                  1. Check all its required constants are present (FLAG <3)
C                  2. Mask the 12 data bits and float the result.
C                  3. Calibrate the value using the constants within a linear
C                     or quadratic equation.
C                  4. Check the result for being within acceptable values.
C                  5. Set data flag bits (16+17) 0: Good data
C                                                1: Data of lower quality
C                                                2: Probably faulty, exceed lims
C                                                3: Data absent or invalid.
C                  Note that parameter 14 (cabin pressure) is optional; this
C                  module will be called by CALIBRATE whether the DRS was
C                  recording cabin pressure or not.
C
C                  Flagging - If a value can't be computed, due to missing data
C                  missing constants, divide be zeroes, etc, a value of 0 is
C                  used, flagged with a three.  If a value is outside its
C                  limits for range or rate of change, it is flagged with a two.
C                  If there are no problems with the data it is flagged with 0.
C                  Any flags on input data are propagated through subsequent
C                  calculations.
C
C VERSION          1.01  280892   A.D.HENNINGS
C
C ARGUMENTS        For Static pressure:
C                    RCONST(1) - REAL*4 IN   Constant in quadratic calib eqn.
C                    RCONST(2) - REAL*4 IN   Coeff X  in quadratic calib eqn.
C                    RCONST(3) - REAL*4 IN   Coeff X2 in quadratic calib eqn.
C                    IFRQ(8)   - INT*4  IN   Input frequency of sampling
C                    IRAW(IN,8)- INT*4  IN   Raw DRS static pressure sensor o/p
C                                             (samples IF = 1,IFRQ(8))
C                    RDER(OP,576) REAL*4 OUT Derived static pressure in mb.
```

(continues on next page)

(continued from previous page)

```
C                                    (samples OP = 1,32)
C                  For Pitot-static pressure:
C                    RCONST(4) -  REAL*4 IN  Constant in linear calib eqn.
C                    RCONST(5) -  REAL*4 IN  Coeff X  in linear calib eqn.
C                    IFRQ(9)   -  INT*4  IN  Input frequency of sampling
C                    IRAW(IN,9)-  INT*4  IN  Raw DRS Pitot-static press sens o/p
C                                            (samples IF = 1,IFRQ(9))
C                    RDER(OP,577) REAL*4 OUT Derived Pitot-static pressure in mb
C                                            (samples OP = 1,32)
C                  For Pressure height:
C                    RDER(IN,576) REAL*4 IN  Derived static pressure in mb.
C                                            (samples IN = 1,32)
C                    RDER(OP,578) REAL*4 OUT Derived Pressure height in metres
C                                            (samples OP = 1,32)
C                    n.b. computed by S_PHGT, valid limits -206.0m to 11.0km
C
C                  For Cabin pressure:
C                    RCONST(6) -  REAL*4 IN  Constant in quadratic calib eqn.
C                    RCONST(7) -  REAL*4 IN  Coeff X  in quadratic calib eqn.
C                    RCONST(8) -  REAL*4 IN  Coeff X2 in quadratic calib eqn.
C                    IFRQ(14) -   INT*4  IN  Input frequency of sampling
C                    IRAW(IN,14)  INT*4  IN  Raw DRS Cabin pressure sensor o/p
C                                            (samples IF = 1,IFRQ(14)) 1HZ
C                    RDER(OP,579) REAL*4 OUT Derived Cabin pressure in mb
C                                            (samples OP = 1)
C
C SUBPROGRAMS      S_PHGT, S_QCPT, ITSTFLG, ISETFLG
C
C REFERENCES        Range limits taken from MRF1/MRF2 and match the sensor range
C                  Rates of change estimated using a max rate of descent
C                  of approx 1000m /minute give Static pressure r.o.c. of
C                  0.05 mb between 32Hz samples, which is less than the
C                  resolution of 0.25 mb per DRS unit. Therefore the r.o.c.
C                  limit for static pressure is set to 1mb between samples,
C                  as the typical noise at low level is less than 3 units
C                  (=0.75 mb).
C                   The Pitot-static system is much noisier at low level
C                  and samples may be up to 50 DRS units different. Therefore
C                  the r.o.c. limit is set to 1.5 mb (37 DRSU) to capture
C                  only the extreme noise and spikes.
C                   The Cabin pressure sensor is of the same type as is used
C                  in the Static pressure system. Limits on normal range are
C                  the same maximum pressure as Static, a minimum set of 650mb
C                  which is marginally lower than the altitude power cut-out
C                  switch's activating level at 10,000ft.  Rates of change can
C                  vary in excess of environmental rates due to manual control
C                  of cabin pressurisation.
C
C CHANGES          V1.00 23/01/90  ADH Original version
C                  V1.01 28/08/92  Includes calibration of CABIN pressure (ADH)
C                  V1.02 02/06/93  Pitot-static r.o.c. limit now set to 4.4mb
C                  (was 1.5mb) between samples.  This is based on analysis of
C                  the high turbulence A257 flight when there were meaningful
C                  changes of up to 4.0 mb between samples.  The limit on rate
C                  of change of cabin pressure has been adjusted to 2.5mb/s in
C                  the light of experience.  Data flagged 2 is now processed,
C                  and data flags propagate consistently.  (WDNJ)
```

(continues on next page)

(continued from previous page)

```
C
C*******************************************************************************
      SUBROUTINE C_PRESS(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.02'
C
      IMPLICIT  INTEGER*4 (I)
      IMPLICIT  REAL*4    (R)
      INTEGER*4 IRAW(64,512),IFRQ(512)
      REAL*4    RCONST(64),RDER(64,1024)
      INTEGER*4 ICFLAG(8)
      DATA      R576ERCNT, R577ERCNT,R577ERCNT /3*1.0/ !S_QCPT error counts
      DATA      RLV576,RLV577,RLV579 /3*0./            !S_QCPT last good values
      DATA      RLT576,RLT577,RLT579 /3*0./            !S_QCPT last times
C
      PARAMETER R576MX=1050.           !Max static pressure (mb)
      PARAMETER R576MN=100.            !Min static pressure (mb)
      PARAMETER R576RG=1.              !Max static pressure change (32mb/s)
      PARAMETER R577MX=125.            !Max Pitot-static pressure (mb)
      PARAMETER R577MN=0.              !Min Pitot-static pressure (mb)
      PARAMETER R577RG=4.4             !Max Pitot-static pressure chnge (32mb/s)
      PARAMETER R579MX=1050.           !Max cabin pressure (mb)
      PARAMETER R579MN=650.            !Min cabin pressure (mb)
      PARAMETER R579RG=2.5             !Max cabin pressure change mb/s
C
C Note that if this routine does not compute a value for any reason then
C CALIBRATE will automatically use values of zero flagged with threes.
C
      SAVE
C      SAVE RLV576,RLV577,RLV579
C      SAVE RLT576,RLT577,RLT579
C      SAVE R576ERCNT,R577ERCNT,R579ERCNT

      DO IT=1,8
        ICFLAG(IT)=ITSTFLG(RCONST(IT)) !Note Constants flags
      END DO
      RSEC=RDER(1,515)                 !Time - seconds past midnight
C
C Derive static pressure and pressure height
C
      ICONFLG=MAX(ICFLAG(1),ICFLAG(2),ICFLAG(3)) !Check constants flags
      IF(ICONFLG.LT.3.AND.IFRQ(8).GT.0) THEN
        DO IS=1,IFRQ(8)                !For each data sample
          RVAL=FLOAT(IRAW(IS,8).AND.'FFF'X) !Just keep 12 DRS bits
          ISTPFLG= 0 !ITSTFLG(IRAW(IS,8))  !Check for GOULD flags
          IF(ISTPFLG.LT.3) THEN
            RDER(IS,576)=(RVAL*RCONST(3)+RCONST(2))*RVAL+RCONST(1) !Cal static
            CALL S_QCPT(RSEC,RLT576,RDER(IS,576),RLV576, !Quality control point
     -          R576MX,R576MN,R576RG,3.,R576ERCNT,IQFLAG)
            ISTPFLG=MAX(ISTPFLG,IQFLAG)
            CALL ISETFLG(RDER(IS,576),ISTPFLG) !Apply flag
            IF(ISTPFLG.LT.3) CALL S_PHGT(RDER(IS,576),RDER(IS,578)) !Press hght
          END IF
        END DO                         !Next sample
      END IF
C
C Derive Pitot-static pressure
C
```

(continues on next page)

(continued from previous page)

```
      ICONFLG=MAX(ICFLAG(4),ICFLAG(5)) !Check constants flags
      IF(ICONFLG.LT.3.AND.IFRQ(9).GT.0) THEN
        DO IS=1,IFRQ(9)                 !For each data sample
          RVAL=FLOAT(IRAW(IS,9).AND.'FFF'X) !Just keep 12 DRS bits
          IPSPFLG=0 !ITSTFLG(IRAW(IS,9))  !Check for GOULD flags
          IF(IPSPFLG.LT.3) THEN
            RDER(IS,577)=RVAL*RCONST(5)+RCONST(4) !Calibrate Pitot-static
            CALL S_QCPT(RSEC,RLT577,RDER(IS,577),RLV577, !Quality control point
     -          R577MX,R577MN,R577RG,3.,R577ERCNT,IQFLAG)
            IPSPFLG=MAX(IPSPFLG,IQFLAG)
            CALL ISETFLG(RDER(IS,577),IPSPFLG) !Apply flag
          END IF
        END DO                          !Next sample
      ENDIF
C
C Derive Cabin Pressure - note that this parameter was never processed on the
C GOULD computer so there is no need to check the raw data for flags.
C
      ICONFLG=MAX(ICFLAG(6),ICFLAG(7),ICFLAG(8)) !Check constants flags
      IF(ICONFLG.LT.3.AND.IFRQ(14).GT.0) THEN
        DO IS=1,IFRQ(14)                !For each data sample
          RVAL=FLOAT(IRAW(IS,14).AND.'FFF'X) !Just keep 12 DRS bits
          RDER(IS,579)=(RVAL*RCONST(8)+RCONST(7))*RVAL+RCONST(6) !Cal cabin pres
           CALL S_QCPT(RSEC,RLT579,RDER(IS,579),RLV579, !Quality control point
     -        R579MX,R579MN,R579RG,3.,R579ERCNT,IQFLAG)
         CALL ISETFLG(RDER(IS,579),IQFLAG) !Apply flag
        END DO                          !Next sample
      END IF
C
      RETURN
      END
```

8.17 c_psap.for

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!         ROUTINE      C_PSAP
!
!         PURPOSE      Calibrates the parameters from the Particle Soot
!                      Absorbtion Photometer (PSAP).
!
!         DESCRIPTION  All raw parameters are digital conversions of the input
!                      voltage. The digital values are converted using a
!                      linear fit then the instrument cals are
!                      applied to obtain the output derived values.
!                      Parameter 175 is linear thus
!
!                           output = v * 0.5E-5
!
!                      v the derived voltage and vfs the full scale voltage.
!                      Parameter 177 is logrithmic so
!
!                          ouput = 10**((v/2.0) - 7.0)
!
!
```

(continues on next page)

(continued from previous page)

```
!          VERSION      1.00  D.P.Briggs
!
!          SUBROUTINES  ISETFLG
!
!          FILES        NONE
!
!          PARAMETERS   RAW      DERIVED    FREQ   RANGE   UNITS
!   PSAP LIN ABS COEFF  185        648      1Hz    0-10V    /m
!   PSAP LOG ABS COEFF  186        649      1Hz    0-10V    /m
!   PSAP TRANSMITTANCE  187                 1Hz    0-10V
!   PSAP FLOW RATE      188                 1Hz    0-10V
!
!          PSAP CONSTANT KEYWORDS
!   CALPLIN    i j    NOTE : i & j are multiplexer calibrations.
!   CALPLOG    i j
!
!          CHANGES
!
!          V1.01  01/10/02  W.D.N.JACKSON
!          Adjusted for 16 bit data from the new DRS
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      SUBROUTINE C_PSAP(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.01'
      IMPLICIT NONE
      INTEGER*4 IRAW(64,512),IFRQ(512)
      REAL*4    RCONST(64),RDER(64,1024)
      REAL      RVOLT
      INTEGER*4 IFLAG,IPARM,IFREQ,IT,IF
!
! initialise output values to 0.0 flagged 3
      IFLAG = 3
      DO IPARM = 648,649
        DO IFREQ = 1,IFRQ(IPARM-463)
          RDER(IFREQ,IPARM)=0.0
          CALL ISETFLG(RDER(IFREQ,IPARM),IFLAG)
        ENDDO
      ENDDO
!
! Check for possible error in DRS data by checking for all 0s or 1s.
! Do not process futher if any parameter is found faulty.
!
      IFLAG = .FALSE.
      DO IPARM = 185,188
        DO IFREQ = 1,IFRQ(IPARM)
          IF(IRAW(IFREQ,IPARM).EQ.0) IFLAG=.TRUE.
          IF(IRAW(IFREQ,IPARM).EQ.'FFFF'X) IFLAG=.TRUE.
        ENDDO
      ENDDO
      IF (IFLAG) RETURN
!
! FLAGGING SCHEME
!    filter transmittance < 0.5 flag 1
!    filter transmittance > 1.0 flag 3
!    flow rate < 1.0 lpm flag 1
!    flow rate = 0.0 lpm flag 3
!   Bit values are taken from old DRS version and scaled between .1 and .9 *2**16
```

(continues on next page)

(continued from previous page)

```
!
      IT=IRAW(1,187)
      IF=IRAW(1,188)
      IF (IT .LT. 27546) THEN
        IFLAG = 1
      ELSE IF (IT .GT. 48538) THEN
        IFLAG = 3
      ELSE IF (IF .LT. 17050 .AND. IF .GE. 7194) THEN
        IFLAG = 1
      ELSE IF (IF .LT. 7194 ) THEN
        IFLAG = 3
      ELSE
        IFLAG = 0
      ENDIF
!
      IF (IFLAG .LT. 3) THEN
!   185/648 1Hz   PSAP LIN ABS COEFF
        RVOLT = RCONST(1) + RCONST(2) * FLOAT(IRAW(1,185))
        RDER(1,648) = RVOLT * 0.5E-5
!
!   186/649 1Hz   PSAP LOG ABS COEFF
        RVOLT = RCONST(3) + RCONST(4) * FLOAT(IRAW(1,186))
        RDER(1,649) = 10**((RVOLT/2.0) - 7.0)
      ENDIF
      CALL ISETFLG(RDER(1,648),IFLAG)
      CALL ISETFLG(RDER(1,649),IFLAG)
!
      RETURN
      END
```

8.18 c_radal1.for

```
C
C ROUTINE        C_RADAL1 SUBROUTINE FORTVAX
C
C PURPOSE        To subroutine to calculate the aircraft altitude from the radar
C                altimeter.
C
C DESCRIPTION        The raw radar altimeter data is provided as a 16 bit signed
C               number from the ARINC 429 data bus, with a least bit resolution
C               of 0.25 ft.
C
C                The derived data is quality controlled to ensure that:
C                (a) data outside the range 0 to 8191.75 ft are flagged 3
C                (b) more than two values the same are flagged 3
C                (c) more than 1000' change between values is flagged 3
C
C TO COMPILE        $FORT C_RADAL1
C
C VERSION        1.00  02/10/02  W.D.N.JACKSON
C
C ARGUMENTS        IRAW(X,37)  - where x=1 or 2, on entry this contains the raw
C                              radar height.
C                IFRQ(37)    - on entry contains 2, the frequency of the raw
```

(continues on next page)

(continued from previous page)

```
C                               radar height.
C                RDER(X,575) - where x= 1 or 2, on exit contains the derived
C                               radar height in meters.
C
C CHANGES        V1.01  WDNJ  05/11/04
C                Flagging criteria improved
C
!*******************************************************************************
      SUBROUTINE C_RADAL1(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.01'
      IMPLICIT  NONE
      INTEGER*4 IRAW(64,512),IFRQ(512),IFLG,IS,IV,ILSTVAL,IMATCH,
     &          ILSTFLG,IMATCHVAL
      REAL*4    RCONST(64),RDER(64,1024)
      DATA      ILSTVAL /-1/, IMATCH /0/, ILSTFLG /0/
!
! Convert raw data
! to metres and store in appropriate element of array RDER.
! Repeat this for all samples passed to the routine. Note that IRAW contains
! the 16 bit number zero extended.
!
      DO IS=1,IFRQ(37)
        RDER(IS,575)=0.0
        IFLG=0
        IV=IRAW(IS,37)
        IF(IV.EQ.'FFFF'X) IFLG=3       !No DLU
        IF(IV.EQ.'FFFE'X) IFLG=3       !No ARINC data
        IF(IV.LT.0.OR.IV.GE.'7FFF'X) IFLG=3 !Glitch or maxed out
        IF(IV.EQ.ILSTVAL) IMATCH=IMATCH+1 !Count data replications
        IF(IV.NE.ILSTVAL) IMATCH=0
        IMATCHVAL=1                    !More than two the same is prob error
        IF(ILSTFLG.NE.0) IMATCHVAL=0
        IF(IV.EQ.ILSTVAL.AND.IV.NE.0.AND.IMATCH.GT.IMATCHVAL)
     &    IFLG=-1                      !Keep data value rather than set to 0
        IF(ABS(IV-ILSTVAL).GT.4000) IFLG=3 !More than 1000ft in 0.5s is error
        ILSTVAL=IV
        IF(IFLG.LT.3) RDER(IS,575)=IV*0.25*0.3048
        IF(IFLG.EQ.-1) IFLG=3
        ILSTFLG=IFLG
        CALL ISETFLG(RDER(IS,575),IFLG)
      END DO
      RETURN
      END
```

8.19 c_rflux.for

```
C-------------------------------------------------------------------------------
C ROUTINE             C_RFLUX        SUBROUTINE FORTVAX       [C_RFLUX.FOR]
C
C PURPOSE            CORRECT RAW FLUXES FOR PYRANOMETERS AND PYRGEOMETERS
C
C DESCRIPTION      Flux corrections are performed for the six instruments
C                  which are normally configured:
C                  Upward-facing :-  Clear dome and Red dome pyranometers.
```

(continues on next page)

(continued from previous page)

```
C                                   Silver dome pyrgeometer.
C                     Downward-facing:- Clear dome and Red dome pyranometers.
C                                   Silver dome pyrgeometer.
C
C                     The actual configuration is specified by the preset array
C                     ICONF, which has six elements whose meaning interpreted as:
C                       1,4 : Clear dome pyranometer  (upper/lower)
C                       2,5 : red    "       "           "     "
C                       3,6 : Silver "   pyrgeometer     "     "
C                      (normally: ICONF(1-3) Upper instruments.
C                                 ICONF(4-6) Lower instruments.)
C
C                    Check that the normal configuration of instruments is to
C                    be used. Any changes are indicated by the presence of a large
C                    offset to the last calibration constant for any instrument
C                    (i.e. the obscurer indicator constant).
C                    If this is present the offset is interpreted as a revised
C                    ICONF indicator for that instrument. See note below.]
C
C                     n.b. Lower instruments were fitted w.e.f. Flight H797
C                          Upper instruments were fitted w.e.f. Flight H842
C
C                     This value solely determines the control path through the
C                     routine for the processing of each instruments inputs.
C                     Should the configuration aboard the aircraft be changed
C                     the array ICONF should be adjusted accordingly.
C                     e.g. If ICONF(1) was modified := 2; it would imply that the
C                     'channel' contained raw flux, zero-offset and thermistor
C                     values for a red dome - rather than clear - pyranometer.
C                     The value of ICONF(1) i.e. 2 would determine the processing
C                     path, the selection of the appropriate set of constants
C                     to apply for correction and the range checking.
C
C NOTE                CHANGES FROM STANDARD CONFIGURATION.
C                     Should the configuration of BBR instruments aboard the
C                     aircraft be changed e.g. swapping a red dome for clear dome,
C                     the array ICONF is  adjusted accordingly. The mechanism used
C                     is to add an offset to the sixth constant in the calibration
C                     constants file (i.e. the obscurer) for that instrument.
C                     Example: If the second 'channel' (inputs 674,677,680) which
C                     in the standard configuration is a red dome pyranometer,
C                     was replaced with a second clear dome instrument, the sixth
C                     constant for the second line of the constants for C_RFLUX
C                     would be changed from 1.0000E+0 to 21.0000E+0, the offset
C                     decodes to "2" when detected by this program.
C                     This is assigned to ICONF(2) and would imply that the
C                     'channel' inputs contain raw flux, zero-offset and thermistor
C                     values for a red dome - rather than clear dome - pyranometer,
C                     and should be range-checked for that type of output only.
C
C                     Corrections applied:
C                     --------------------
C                     Pyranometers (Clear and Red dome) are corrected for:
C                     - Subtraction of a zero offset (mean over past 10 seconds)
C                     - Attitude (pitch and roll) -Upper instruments only.
C                       test if flux is above a critical limit indicating a direct
C                       solar beam component.
```

(continues on next page)

(continued from previous page)

```
C                       If not direct, assume diffuse and apply no attitude corr.
C                       If DIRECT, a geometric correction is used to "level"
C                       the instrument to produce the equivalent hemispheric
C                       downward flux through a horizontal surface (without
C                       inclusion of diffuse component).
C                       The ratio of the Direct:Direct+Diffuse components is
C                       assumed to be 0.95 at present. This value could be
C                       optimised for a particular atmosphere depending on the
C                       turbidity as a function of height.
C
C                       Correct for COSINE effect. (MRF Technical note No.7).
C                       [Pitch and roll offsets of the instrument thermopiles
C                       relative to the aircraft INS platform are derived in
C                       flight by flying a box pattern in cloud-free skies -
C                       These offsets are then used in addition to the INS pitch
C                       and roll (meaned over two seconds). (See MRF Technical
C                       note No 4.) and these values are supplied as arguments
C                       four and five in each set of CONSTANTS below.
C                   - Time constant of thermopile relative to INS. The mean of
C                     last two seconds of INS pitch/roll angles are used in the
C                     attitude correction, giving an effective difference of
C                     0.5 seconds.
C                   - Correct flux output for proportion of hemispheric dome
C                     obscured by indicated obscurer pillar. (Rawlins 1986).
C
C                   Pyrgeometers (IR) are corrected for:
C                   - Zero offset (mean over past 10 seconds)
C                   - Temperature sensitivity  (Coefficients in CONSTANTS below)
C                   - Linear dependence 0.2% per degree with sensitivity defined
C                     as unity at zero C. applied to signal. (MRF Int note No 50)
C                   - Calculation of flux (sigma T^4 correction)
C                     Flux = signal +(sigma* Tsink^4)
C                     where sigma = Stefan-Boltzmann constant.
C                   _ Upper instrument is corrected for dome transmission
C                     effects (MRF Tech note 3)
C
C VERSION            1.14 17-01-96   D Lauchlan
C
C METHOD            1. First time routine is called, assign constants to named
C                      program variables/arrays.
C                      Decide on basis of input constants whether upper instr.
C                      data is available to be processed.
C                   2. Derive/convert any intermediate results used multiply
C                      within several code sections following.
C                   3. Derive running mean zero-offsets over the past 10 seconds
C                      for each instrument
C
C                   4. Calculate mean pitch and roll values for the current
C                      second and use them to derive running means for the past
C                      two seconds.
C                   5. Correct thermistor temperatures for non-linearity.
C                   6. Cycle through each of six instrument input channels.
C                      Use the control variable in ICONF() to select execution
C                      of appropriate code sections.
C                      In all cases; derive a signal zero-offset and reduce the
C                      signal flux by this amount.
C                      Apply temperature-dependance corrections to pyranometers.
```

(continues on next page)

(continued from previous page)

```
C                      For upward-facing pyranometers the 'critical' value to
C                      discriminate between diffuse and direct-sun conditions is
C                            FCRIT  = 920.*(COS(ZENRAD))**1.28
C                      where ZENRAD : solar zenith angle (in radians)
C                      [N.B. This approximates to the 'German' equation but is
C                       simpler, and does not produce negative values at low
C                       Sun elevations].
C                      Correct flux output for proportion of hemispheric dome
C                      obscured by indicated obscurer pillar. (Rawlins 1986).
C
C                   7. Range check flux output and set a flag accordingly.
C                      Apply flag values to resulting flux output dependent on
C                      relevant flag settings.
C
C ARGUMENTS         RCONST(1),( 7)..(31)  - REAL*4 IN Temperature Sens. coeff a
C                   RCONST(2),( 8)..(32)  - REAL*4 IN Temperature Sens. coeff b
C                   RCONST(3),( 9)..(33)  - REAL*4 IN Temperature Sens. coeff c
C                   RCONST(4),(10)..(34)  - REAL*4 IN Pitch offset of Instrument
C                   RCONST(5),(11)..(35)  - REAL*4 IN Roll  offset of Instrument
C                   RCONST(6),(12)..(36)  - REAL*4 IN Obscurer pillar type.
C
C                   RDER(1,par)             REAL*4  IN Six raw flux signals W/M-2
C                              (par=673-675,682-684)
C                   RDER(1,par)             REAL*4  IN six zero-offsets   (W/M-2)
C                              (par=676-678,685-687)
C                   RDER(1,par)             REAL*4  IN six instr. temperatures K
C                              (par=679-681,688-690)
C                   RDER(32,560)            REAL*4  IN INS Roll      (degrees)
C                   RDER(32,561)            REAL*4  IN INS Pitch     (degrees)
C                   RDER(32,562)            REAL*4  IN INS heading   (degrees)
C                   RDER(1,642)             REAL*4  IN Solar azimuth (degrees)
C                   RDER(1,643)             REAL*4  IN Solar zenith  (degrees)
C
C                                                          Pos. Dome  Units
C                   RDER(1,1019)            REAL*4 OUT Corrected Upp Clear W/m-2
C                   RDER(1,1020)            REAL*4 OUT flux.      "  Red dome "
C                   RDER(1,1021)            REAL*4 OUT            "  I/R      "
C                   RDER(1,1022)            REAL*4 OUT           Low Clear    "
C                   RDER(1,1023)            REAL*4 OUT            "  Red dome "
C                   RDER(1,1024)            REAL*4 OUT            "  I/R      "
C
C SUBPROGRAMS              ITSTFLG, ISETFLG, S_RUNM, CORR_THM, RMEANOF, CIRC_AVRG
C
C REFERENCES             MRF Internal note  4.
C                    "     "      "   12.
C                    "     "      "   31.
C                    "     "      "   50.
C                    "     "      "   56.
C                   MRF Technical note 3. Pyrgeometer Corrections due to Dome
C                                         Transmission.  February 1991 Kilsby
C                   MRF Technical note 7. Report of Broad-band radiative fluxes
C                                         working group. 17/12/91      Saunders
C                   MRF Technical note 8. Pyramometer calibrationsin Ascension
C                                         of Feb.1992.   4/6/92        Seymour
C                   RAWLINS  R        D/Met.O.(MRF)/13/1      1986.
C                   SAUNDERS R         "        "   "         21/3/90
C                   SAUNDERS R        M/MRF/13/5              22/7/92
```

(continues on next page)

(continued from previous page)

```
C
C CHANGES          10/01/91 A.D.Hennings.
C                    Ability to change ICONF to when reconfiguring instrument
C                    fit on A/C using the constants file.
C                 10/01/91 Pitch & Roll averaging changed from 3 to 2 seconds.
C                 25/01/91 Flags assessment changed; use of new flag IFLAG_SUN
C                 29/01/91 Roll limit checking:replace ROLBAR with ABS(ROLBAR).
C                          Flags assessment changed; IFLAG_OUTPUT being max of
C                          (signal,Pitch,Roll,Zenith) flags.
C                 30/07/91 FCRIT for Red dome now only used if no clear dome
C                 16/10/91 Corrected pyrgeometer temp sensitivity correction
C                 20/01/92 Use INS heading instead of obsolete Omega heading.
C                 03/02/92 New subroutine CIRC_AVRG to calc INS mean heading
C                 21/07/92 Levelling of upper pyranometers changed to use
C                          direct beam component, and cosine effect included.
C                          Recommendations of MRF Tech note 7.   (V1.13)
C                          references to Tech note 8. and M/MRF/13/5
C                 24/07/92 Pyrgeometer corrections for Dome transmission.
C                          (Downwelling) MRF Tech note 3.
C                 17/01/96 D Lauchlan
C                          Unused variables removed
C                 22/12/97 W D N Jackson, Flags cleared from all data before
C                          use.
C                 11/08/98 W D N Jackson, Upper pyranometer obscurer
C                          corrections changed to correct values.  The
C                          values have been incorrect in all previous versions
C                          of C_RFLUX. The error is only small. (Source
C                          P Hignett)
C-----------------------------------------------------------------------------
      SUBROUTINE C_RFLUX     (IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.16'
C
      IMPLICIT NONE
      INTEGER*4 IRAW(64,512), IFRQ(512)
      REAL*4    RCONST(64), RDER(64,1024)

      INTEGER   ITSTFLG
      REAL      CIRC_AVRG              !Function returning average of angles
C
C working input data and processed output arrays
C
      REAL*4 ZIN(6),                   !Zero offset samples
     &       RTHM(6),                  !Uncorrected thermistor samples
     &       RFLX(6),                  !Uncorrected flux samples
     &       THM(6),                   !Corrected thermistor samples
     &       FLX(6),                   !Corrected flux samples
     &       PITINS,ROLINS,            !Input pitch & roll (mean of 32hz) degs
     &       PITCH ,ROLL,              !Corrected pitch and roll (Rads)
     &       HDGINS,                   !Input INS heading (degrees)
     &       SOLAZM,SOLZEN,            !Input Solar Azimuth & zenith angle. Rads
     &       HDGRAD,                   !Convert Omega heading to radians
     &       ZENRAD,                   !Convert Solar Zenith ang to radians
     &       AZMRAD,                   !Convert Solar Zenith ang to radians
     &       SUNHDG                    !Sun Heading (Sol Azm-A/c Omega hdg.)Rads
C
C C0NSTANT information
C
```

(continues on next page)

(continued from previous page)

```
      REAL*8    TSA(6)                       !Temperature senstvty alph,beta gm
     -         ,TSB(6)                       !
     -         ,TSG(6)                       !
     -         ,PITOFF(6)                    !Angular offset   " Pitch.
     -         ,ROLOFF(6)                    !Angular offset   " Roll.
      INTEGER*4 IOBTYP(6)                    !Obscurer type (0: none 1:short
                                             !               2: tall)
C
C flags signifying validity of input arguments and derived values.
C
      INTEGER*4 IFLAG_ANG              !Test of sun angle too low
     -         ,IFLAG_ROLL             !INS Roll
     -         ,IFLAG_PIT              !INS Pitch
     -         ,IFLAG_AZM              !Solar azimuth angle
     -         ,IFLAG_ZEN              ! "    zenith   "
     -         ,IFLAG_INHDG            !INS Heading
     -         ,IFLAG_SHDG             !Sun hdg.  Max(IFLAG_AZM and IFLAG_INHDG)
     -         ,IFLAG_SUN              !Sun attitude Max(Pitch/Roll/Zen/Ang)
     -         ,IFLAG_FLX              !Raw flux input
     -         ,IFLAG_THM              !Corrected thermistor
     -         ,IFLAG_ZER              !Meaned zero-offset
     -         ,IFLAG_SIGNAL           !Max of (IFLAG_FLX and IFLAG_ZER)
     -         ,IFLAG_CORRN            !Max of (all correction flags relevant)
     -         ,IFLAG_OUTPUT           !Max of (IFLAG_SIGNAL and IFLAG_CORRN)
                                       ! and result of range tests on output.
     -         ,IDUM                   !Argument, return value of no interest.

C  arrays , counters and pointer arguments for Zero-offset mean derivation

      REAL*4    ZBAR(6)                  !Output means over past 10 seconds
      REAL*4    ZBUF(10,6), ZSUM(6)      !Buffer and total holder
      INTEGER*4 IZP(6),     IZCNT(6)     !Buffer pointer and counter of samples.
      DATA      IZP/6*1/,   IZCNT/6*0/   !Initialise ptrs, count of good samples
C
C  arrays , counters and pointer arguments for Pitch and Roll mean derivation
C
      REAL*4    PITBAR,ROLBAR            !Output means over past 2 seconds. degs
      REAL*4    PBUF(3),RBUF(3),PSUM,RSUM!Buffers and total holders
      INTEGER*4 IPPT,IRPT,IPCNT,IRCNT    !Buffer pointer and counter of samples.
      DATA      IPPT,IRPT/1,1/           !Initialise buffer pointers
      DATA      IPCNT,IRCNT/2*0/         !Initialise count of good samples


      LOGICAL   OFIRST/.TRUE./           !Indicator as to first time through rtn

      INTEGER*4 ICONF(6)                !6 input channels (instruments).
      DATA      ICONF/                  !Control variables- Currently set as:
     -                 1,               !Upper clear dome pyranometer in chan 1
     -                 2,               !      red   dome pyranometer in chan 2
     -                 3,               !      silverdome pyrgeometer in chan 3
     -                 4,               !Lower clear dome pyranometer in chan 4
     -                 5,               !      red   dome pyranometer in chan 5
     -                 6/               !      silverdome pyrgeometer in chan 6

      REAL*4   RMAXFLX(6),RMINFLX(6)    !Range limits on corrected flux.
      DATA     RMAXFLX/                 !Max. admissible corrected flux output
     -             1380.,               !Upward-facing  clear  dome pyranometer
```

(continues on next page)

(continued from previous page)

```
     -             700.,               !              red    dome pyranometer
     -             550.,               !              silver dome pyrgeometer
     -            1380.,               !Downward-facing clear dome pyranometer
     -             700.,               !               red   dome pyranometer
     -             750 /               !               silverdome pyrgeometer
      DATA     RMINFLX/                !Min. admissible corrected flux output
     -             -20.,               !Upward-facing  clear  dome pyranometer
     -             -20.,               !              red    dome pyranometer
     -             -20.,               !              silver dome pyrgeometer
     -             -20.,               !Downward-facing clear dome pyranometer
     -             -20.,               !               red   dome pyranometer
     -              50./               !               silverdome pyrgeometer

      REAL*4    THETA,RCOSTH             !Angle between Sun and Normal to Instr
      REAL*4    ROLLIM,THTMAX            !Roll max limit: Sun-angle max limit
      PARAMETER (ROLLIM=7.0, THTMAX=80.0) !in degrees.
C
C local variables.
C
      LOGICAL   UPPERS                  !Upper instruments fitted?
      INTEGER*4 IS,IE                   !First and last instrument 'channel'
C       SAVE IS,IE
      INTEGER*4 IN,I                    !Instrument (channel); loop index
      REAL*4    FCRIT,FCRITVAL          !Critical flux value (direct/diffuse)
      REAL*8    SIGMA,                  !Stefan-Boltzmann constant.
     -          FOBSC,                  !Obscurer value for any instrument
     -          TH,                     !Place holder for corrected thermistor
     -          FL                      !Place holder for corrected flux
      REAL*4    DEG2RD                  !Degrees to radians conversion factor
      REAL*4    RTEMP,                  !Temp vrb: used with ICONF changes.
     -          ROBTYP                  ! "    " : specify Obscurer type used.
      INTEGER*4 ITYPE,ISIG,ICOR         !Indices to data tables
C
C  levelling corrections
C                                      !Select INDX of solar zenith angle
      INTEGER*4 INDX                   !where  INDX = NINT(SOLZEN/10) + 1
                                       !INDX
                                       !1-3: (0 -29.9 deg)
                                       !4-6: (30-59.9 deg)
                                       !7-9: (60-89.9 deg)
                                       !10:  (  >89.9 deg)

      REAL*4    CEFF(10)/1.010, 1.005, 1.005, !Correction to pyranometers for
     &                   1.005, 1.000, 0.995, !COSINE effect dependant on solar
     &                   0.985, 0.970, 0.930, !zenith angle. Determined by expt
     &                   0.930/               !Ref: Tech note 8. Table 4

      REAL*4    FDIR(10)/.95,.95,.95,   !(Proportion of flux from direct source
     &                   .95,.95,.95,   !for varying solar zenith angles.)
     &                   .95,.95,.95,   !Addressed by INDX as above.
     &                   .95/           !Ref: M/MRF/13/5


C     table of proportion of hemispheric dome obscured by each pillar-type

      REAL*4    ROBSC(3,6)              !Obscurer corrections (Type,Up|Loc)
      DATA ((ROBSC(ITYPE,IN),IN=1,6),ITYPE=1,3)/    !Ref:RAWLINS 1986
```

(continues on next page)

(continued from previous page)

```
      !    Upper Instruments  |   Lower instruments
      !Port    Starbd  Centre  Port    Starbd  Centre
     & 00.000, 00.000, 00.000, 00.000, 00.000, 00.000,  ! No pillar (Ind=1)
     & 00.010, 00.010, 00.000, 00.000, 00.000, 00.000,  ! Short "   ( "  2)
     & 00.040, 00.040, 00.000, 00.000, 00.000, 00.000/  ! Tall  "   ( "  3)
! The following lines contain the incorrect upper pyranometer corrections which
! have been used in all previous versions of C_RFLUX (WDNJ 11/8/98).
!     & 00.016, 00.016, 00.000, 00.000, 00.000, 00.000,  ! Short "   ( "  2)
!     & 00.046, 00.046, 00.000, 00.000, 00.000, 00.000/  ! Tall  "   ( "  3)

C     logic table combining two group input flag conditions resulting in an
C     output flag.

      INTEGER*4 IFLAG_TABLE(0:3,0:3)
      DATA ((IFLAG_TABLE(ISIG,ICOR),ICOR=0,3),ISIG=0,3)/
      !        CORRECTION
      !     0   1   2   3
      !   -------------------                 See Saunders LM 1990 for
     -      0,  1,  3,  3,  ! 0               details of this table.
     -      1,  2,  3,  3,  ! 1  SIGNAL
     -      2,  2,  3,  3,  ! 2
     -      3,  3,  3,  3  /! 3

      PARAMETER (SIGMA  = 5.669E-08)
      PARAMETER (DEG2RD = 57.295776)
      SAVE
!-----------------------------------------------------------------------------
!+
!  1. First time routine is called, assign constants to named
!     program variables/arrays.


      IF (OFIRST) THEN
        OFIRST= .FALSE.
!
!       Prior to Flight H842 no upper radiometers were recorded in this form;
!       hence no data constants are passed to this routine. Check for condition.
!
        UPPERS = .FALSE.
        DO IN = 1 ,18                               !Any non-zero value indicates
        IF (RCONST(IN) .NE. 0.)  UPPERS = .TRUE.    !constants are being passed
        END DO                                      !for upper instruments too.
!
!       Set 'channel' limits accordingly.
!
        IF (UPPERS) THEN
          IS = 1                    !all six instrument present
          IE = 6
        ELSE
          IS = 4                    !only lower instruments fitted
          IE = 6
        ENDIF

!                        Put RCONST values into program variables.)
        DO IN = IS,IE
          TSA(IN)    = RCONST((IN-1)*6 +1) !Temperature sensitivity coefficents
          TSB(IN)    = RCONST((IN-1)*6 +2) ! Alpha, Beta, Gamma
```

(continues on next page)

(continued from previous page)

```
          TSG(IN)    = RCONST((IN-1)*6 +3) !
          PITOFF(IN) = RCONST((IN-1)*6 +4) !Pitch offset of instrument
          ROLOFF(IN) = RCONST((IN-1)*6 +5) !Pitch offset of instrument

!         Check whether the configuration has been modified by examining the
!         last constant for each instrument (=IOBTYP). If it is >10 an offset
!         has been added to it; identify this and restore correct constant.
!
          RTEMP = RCONST((IN-1)*6 +6)            !Get obscurer value (+offset?)
          IF (ABS(RTEMP) .GE. 10.0) THEN         !An offset has been added.
              RTEMP      = RTEMP/10.             !Bring the offset into the
              ICONF(IN)  = INT(RTEMP)            !truncate range |1 - 6|>ICONF()
              ROBTYP     = (RTEMP-ICONF(IN))*10. !Restore the Obscurer const.
              ICONF(IN)  = IABS(ICONF(IN))       !Config indicator must be +ve.
              IOBTYP(IN) = NINT(ROBTYP)          !assign Obscurer type in use
                                                 !(1: none, 2: short, 3: tall)

          ELSE                                   !use default ICONF values
              IOBTYP(IN) = NINT(RTEMP)           !Obscurer type in use
          ENDIF

        END DO                          !next instrument.
      ENDIF                             !of First-time-through actions.
!-

!+
!  2. Derive/convert any intermediate results used several times
!     within code sections following.
!
!     Put input data into arrays.

      IF (UPPERS) THEN
        DO IN = 1,3                              !Upper instruments
          RFLX (IN)  = RDER(1,673+IN -1)         !  Signal       w/m-2
          ZIN  (IN)  = RDER(1,676+IN -1)         !  zero         w/m-2
          RTHM (IN)  = RDER(1,679+IN -1)         !  thermistor   deg K
        END DO
      ENDIF

      DO IN = 1,3                                !Lower instruments
        RFLX (IN+3) = RDER(1,682+IN -1)          !  Signal       w/m-2
        ZIN  (IN+3) = RDER(1,685+IN -1)          !  zero         w/m-2
        RTHM (IN+3) = RDER(1,688+IN -1)          !  thermistor   deg K
      END DO

      HDGINS = CIRC_AVRG( RDER(1,562), 32)       !Mean of INS Heading samples
                                                 !(special for circular values)
      SOLAZM        = RDER(1,642)                !Solar azimuth angle
      SOLZEN        = RDER(1,643)                !Solar zenith   "
!-

!+    set flags for corrections

      IFLAG_INHDG = ITSTFLG (HDGINS)             !Flag of INS   heading
      CALL ISETFLG(HDGINS,0)                     !Strip flag
      IFLAG_ZEN   = ITSTFLG (SOLZEN)             !Flag of solar zenith angle
      CALL ISETFLG(SOLZEN,0)                     !Strip flag
```

(continues on next page)

(continued from previous page)

```
      IFLAG_AZM   = ITSTFLG (SOLAZM)             !Flag of solar azimuth angle
      CALL ISETFLG(SOLAZM,0)                     !Strip flag
      IFLAG_SHDG  = MAX(IFLAG_INHDG,IFLAG_AZM)   !Choose higher heading flag
!-

!+    Convert samples to radians measure.

      HDGRAD = HDGINS /DEG2RD            !Convert INS   heading to radians
      ZENRAD = SOLZEN/DEG2RD             !Convert Solar Zenith ang to radians
      AZMRAD = SOLAZM/DEG2RD             !Convert Solar Zenith ang to radians
      SUNHDG = AZMRAD - HDGRAD           !Sun Heading (Solar Az-A/C hdg (INS))
!-

      IF (SOLZEN .GT. 0. .AND. SOLZEN .LT.90.)THEN !Prevent exponentiation error
      FCRIT  = 920.*(COS(ZENRAD))**1.28  !Critical flux value (direct/diffuse)
      ENDIF

!+  3. Derive running mean of zero offsets for each instrument over ten seconds


      DO I=IS,IE
      CALL S_RUNM(ZBUF(1,I),IZP(I),IZCNT(I),10,ZIN(I),ZSUM(I),ZBAR(I))
      END DO

!-

!+  4. means of 32hz INS PITCH & ROLL arguments for one second.

      CALL RMEANOF(32 ,RDER(1,560), ROLINS, IDUM) !Mean of INS Roll samples.
      CALL RMEANOF(32 ,RDER(1,561), PITINS, IDUM) !Mean of  "  Pitch  "    .

!    then derive running mean of pitch and roll values. (meaned over two secs)

      CALL S_RUNM(RBUF,IRPT,IRCNT,2,ROLINS,RSUM,ROLBAR)         !Roll
      CALL S_RUNM(PBUF,IPPT,IPCNT,2,PITINS,PSUM,PITBAR)         !Pitch

!     Set Pitch flag, no acceptability test currently used.

      IFLAG_PIT = ITSTFLG (PITBAR)
      CALL ISETFLG(PITBAR,0)                    !Strip flag

!     Roll limit acceptable?

      IFLAG_ROLL= ITSTFLG (ROLBAR)              !Flag of meaned Roll.
      CALL ISETFLG(ROLBAR,0)                    !Strip flag
      IF ( ABS(ROLBAR) .GT. ROLLIM)             !Comparison in degrees
     -    IFLAG_ROLL= MAX(IFLAG_ROLL,1)         !Flag if Roll too great

!  5.  Correct thermistor values for linearity

      CALL CORR_THM (RTHM,THM)                  !Input temps deg K, output deg C

!-----------------------------------------------------------------------------

      DO IN = IS,IE                       !Cycle through available instruments

      FOBSC     = ROBSC(IOBTYP(IN),IN)     !select correction for obscurer
```

(continues on next page)

(continued from previous page)

```
      IFLAG_CORRN  = 0                     !Set corrections flag to valid
      IFLAG_FLX = ITSTFLG (RFLX(IN))       !Flag of raw flux input
      CALL ISETFLG(RFLX(IN),0)             !Strip flag
      IFLAG_ZER = ITSTFLG (ZBAR(IN))       !Flag of meaned zero-offset
      CALL ISETFLG(ZBAR(IN),0)             !Strip flag
      IFLAG_THM = ITSTFLG (THM (IN))       !Flag of corrected thermistor.
      CALL ISETFLG(THM (IN),0)             !Strip flag

      IFLAG_SIGNAL= MAX(IFLAG_FLX,IFLAG_ZER) !Obtain worst of (flx,zero) flag.

      IF (IFLAG_SIGNAL .EQ. 3) THEN         !**** Check Flux validity
         FLX(IN) =  -99.                    !Set output to 'failed' value.
         IFLAG_OUTPUT=  3                   !'Failed' flag.

      !------------------------------------------------------------------
      ELSE                                  ! OK to begin correcting flux.

        FLX(IN) = RFLX(IN) - ZBAR(IN)       !Subtract meaned zero-offset.

!       Perform temperature sensitivity correction.

          IF (IFLAG_THM .LT. 2) THEN                !Thermistor temperatures
            FL      = FLX(IN)                       !have been corrected and
            TH      = THM(IN)                       !converted  to C by CORR_THM.
            FLX(IN) = FL /
     -               (1.+ TH*(TSA(IN)
     -                  + TH*(TSB(IN)
     -                  + TH* TSG(IN) )))
          ENDIF

        !----------------------------------------------------------------------
        IF (ICONF(IN) .EQ. 3 .OR. ICONF(IN) .EQ. 6) THEN  !*** Pyrgeometers only
        !----------------------------------------------------------------------

!         Perform 'sigma* Tsink^4' correction

          IF (IFLAG_THM .LT. 2) THEN
            FL = FLX(IN)
            FLX(IN) =FL * (1.0/(1.0-FOBSC))+SIGMA*(TH+273.16)**4
          ENDIF
                                    !Correction to upper Pyrgeometer for
                                    !dome transmission of downwelling I/R.
          IF (ICONF(IN) .EQ. 3 )THEN
            FLX(IN) = FLX(IN) + (-6.0 + 0.0175* FLX(IN))!see Tech note 3. page 2
          ENDIF

          IFLAG_CORRN = IFLAG_THM                        !Relevant corrections
          IFLAG_OUTPUT = IFLAG_TABLE(IFLAG_SIGNAL,IFLAG_CORRN)

        !----------------------------------------------------------------------
        ELSE                         !Upper and Lower Pyranometer corrections
        !----------------------------------------------------------------------

          IF (ICONF(IN) .EQ. 4 .OR. ICONF(IN) .EQ. 5) THEN  !Lower pyranometers

             FLX(IN)= FLX(IN)*(1.0/(1.0- FOBSC))            !Obscurer corr'n.
                                                            !All corr'n complete
```

(continues on next page)

(continued from previous page)

```
             IFLAG_CORRN = 0                                !no relevant corrs
             IFLAG_OUTPUT = IFLAG_TABLE(IFLAG_SIGNAL,IFLAG_CORRN)

          ELSE                                              !Upper Pyranometers

!+          Compare incoming flux with Fcrit (Critical value) of expected flux.
!           IF Flux > Fcrit; treat irradiation as being DIRECT.
!           ELSE             assume it is DIFFUSE irradiation.
!             (n.b. for RED dome, Fcrit value used is 1/2 normal Fcrit.)

            FCRITVAL = FCRIT
            IF( ICONF(1)  .NE.  1) FCRITVAL = FCRIT * .5    !1/2 For RED dome.

            IF (FLX(1)  .GT. FCRITVAL) THEN                 !*Direct or Diffuse?
!-

!+            DIRECT is appropriate; check angle between Sun & normal-to-
!             instrument is not > 80 deg, before correction for platform level.

              PITCH=PITBAR + PITOFF(IN) !Combine  A/C mean and Inst offset Pitch
              PITCH=PITCH/DEG2RD        !.. and convert to radians
              ROLL =ROLBAR + ROLOFF(IN) !Combine A/C  mean and Inst offset Roll
              ROLL = ROLL/DEG2RD        !.. and convert to radians

!             Find angle between Solar zenith and normal-to-Instrument.
                                                        !Ref:Tech note 7 Page 10
                                                        !Derive cosine of angle.
              RCOSTH = SIN(ROLL)*SIN(ZENRAD)*SIN(SUNHDG)
     &              +  COS(ROLL)*COS(PITCH) *COS(ZENRAD)
     &              -  COS(ROLL)*SIN(PITCH) *SIN(ZENRAD)*COS(SUNHDG)
              THETA = ACOS(RCOSTH)             !Express angle in radians

!             Compare with maximum allowable angle. ( must be < 80 Deg)

              IF (THETA .GT. THTMAX/DEG2RD) THEN
                IFLAG_ANG =  2                 !Failed Low sun test; Flag value
              ELSE
                IFLAG_ANG =  0                 !Angle Sun/Instr acceptable.
              ENDIF

!             Apply levelling correction using combined pitch and roll, if
!             necessary conditions are met:-

              IFLAG_CORRN = MAX (IFLAG_PIT,  IFLAG_ROLL)  !A/c  Attitude flags.
              IFLAG_CORRN = MAX (IFLAG_CORRN,IFLAG_ANG)
              IFLAG_SUN   = MAX (IFLAG_SHDG ,IFLAG_ZEN)
              IFLAG_CORRN = MAX (IFLAG_CORRN,IFLAG_SUN)

              IFLAG_OUTPUT = IFLAG_TABLE(IFLAG_SIGNAL,IFLAG_CORRN)

              IF ( IFLAG_CORRN .LT. 2 .AND. RCOSTH .NE.0.) THEN

! *OLD VERSION* FLX(IN) = FLX(IN) * (COS(ZENRAD)/RCOSTH)   !levelling correction

!            Correct the flux for attitude of aircraft for direct component of
!            beam. Also include COSINE effect correction.    (Ref: M/MRF/13/5)
```

(continues on next page)

(continued from previous page)

```
                INDX = NINT(SOLZEN/10) + 1
                INDX = MIN (INDX,10)

                FLX (IN) =              FLX(IN)/
!                         --------------------------------------------
     &            (1.- FDIR(INDX)*(1.- CEFF(INDX)*(RCOSTH/COS(ZENRAD))))
              ENDIF

            ELSE                          !* Critical value, (flux less than.)
                                          !  Diffuse case;   make Obscurer
                                          !  correction if signal is valid.

              IFLAG_CORRN  = MAX(IFLAG_PIT,  IFLAG_ROLL)
              IFLAG_CORRN  = MAX(IFLAG_CORRN,IFLAG_ZEN)
              IFLAG_OUTPUT = IFLAG_TABLE (IFLAG_SIGNAL,IFLAG_CORRN)
              FLX(IN) = FLX(IN)*(1.0/(1.0- FOBSC))

            ENDIF                         !* Critical value for direct?

            IF ( IFLAG_SIGNAL .EQ. 3) THEN
              FLX(IN) = -99.                    !set  invalid flux to obvious
            ENDIF                               !known value.

          ENDIF                           !**  Upper or Lower pyranometers?
        ENDIF                             !*** pyranometer or pyrgeometer?

!       Perform range checks on valid output fluxes.

        IF (IFLAG_OUTPUT .LT. 3 ) THEN
            IF (FLX(IN) .GT. RMAXFLX(ICONF(IN)) .OR.
     -          FLX(IN) .LT. RMINFLX(ICONF(IN))     ) THEN
               IFLAG_OUTPUT = 2           !Failed, flag result as 'suspect'
            ENDIF
        ENDIF
      ENDIF                               !**** Flux signal validity?

!     Assign processed flux to output parameter

      RDER(1,1018 + IN) = FLX(IN)                     !Fill output argument
      CALL ISETFLG (RDER(1,1018 + IN), IFLAG_OUTPUT)  !Set output flag

      IFLAG_CORRN       = 0
      IFLAG_SIGNAL      = 0
      IFLAG_OUTPUT      = 0
      END DO                                          !(..Control value IN)

      RETURN
      END


C-----------------------------------------------------------------------------
C ROUTINE          CORR_THM     SUBROUTINE     FORTVAX            [C_RFLUX.FOR]
C
C PURPOSE          Correct thermistors for non-linearity using a quintic eqn.
C
C DESCRIPTION      The thermistors used in the pyrgeometer/pyranometers all
C                  have characteristic non-linear temperature dependence
```

(continues on next page)

(continued from previous page)

```
C                  due to the manufacturing process. If not corrected for,
C                  this can lead to errors in temperature of up to 1 deg C.
C                  The thermistor manufacturers provide a curve of the the
C                  correction needed to be applied for a range of
C                  temperatures.  A quintic equation has been fitted to this
C                  curve to give the best fit coefficients used by this routine.
C
C METHOD           The routine takes an array of six thermistor values in deg K.
C                  In turn; notes each ones flag then clears the flag.
C                  Fits -50 deg C to +40 deg C to within +/- .07 deg C.
C                  Eqn: RT + (RCON +V.RT +W.RT^2 +X.RT^3 +Y.RT^4 +Z.RT^5)
C                  where RT  : Raw thermistor value  (converted to Celsius)
C                        RCON: A constant
C                   V,W,X,Y,Z: Coefficients of quintic equation correcting temp.
C
C                  Loop through six thermistor values:
C                  a)   note each one's flag
C                  b)   if flag indicates input is valid (flag <3)
C                       - clear the flag bits from the raw thermistor value
C                       - assign the value (converted to deg C.) to a working
C                         variable,  which becomes the input variable to a the
C                         quintic equation above.
C                       - derive the corrected output using that equation.
C                       - set input flag value in output thermistor temperature.
C                       else; for an 'invalid' flag
C                       - set the output thermistor value to zero C
C                       - set its output's flag to 3 (= invalid)
C                  next loop.
C
C                  n.b. The corrected thermistor values are not saved at the
C                       end of calibration and are only calculated for local
C                       use in deriving corrected solar fluxes.
C
C VERSION          1.02  30-07-91  A.D HENNINGS
C
C REFERENCES       Best-fit coefficients and constants taken from fitting to
C                  manufacturers calibration data sheet.
C
C ARGUMENTS        REAL*4 RTHM(6)  IN   Six uncorrected thermistor values. deg K
C                  REAL*4 THM (6) OUT   Six  corrected thermistor values.  deg C
C
C SUBPROGRAM       ITSTFLG ISETFLG
C
C CHANGES          1.01 201190 Documentation.
C                  1.02 300791 Documentation.
C                  1.03 17-01-96 D Lauchlan
C                  Unused variables removed
C                  1.04 22-03-04 D Tiddeman flag stripping before calculation
C                  changed to prevent crashes.
C-------------------------------------------------------------------------------
      SUBROUTINE CORR_THM (RTHM,THM)
CDEC$ IDENT 'V1.04'
C
      IMPLICIT NONE
      REAL*8 V,W,X,Y,Z,            !Coefficients of powers 1, 2, 3, 4 & 5
     -       RT,RCON               !placeholder  for thermistor for calc.
      REAL*4 RTHM(6),THM(6)        !Raw Thermistor, corrected thermistor.
```

(continues on next page)

(continued from previous page)

```
      INTEGER*4 I,IFLAG ,ITSTFLG
c      LOGICAL OFIRST_TIME/.TRUE./  !   "      "
      PARAMETER (RCON = -0.774,
     -              V =  6.08E-02,
     -              W =  2.47E-03,
     -              X = -6.29E-05,
     -              Y = -8.78E-07,
     -              Z =  1.37E-08)
!

      DO  I=1,6
      IFLAG = ITSTFLG(RTHM(I))
      CALL ISETFLG(RTHM(I),0)                      !Clear flag before calc.
      IF (IFLAG .LT. 3) THEN
        RT     = RTHM(I) - 273.16                  !convert to Celsius
        THM(I) = RT + (RCON + RT*(V+ RT*(W+RT*(X+RT*(Y+RT*Z)))))
        CALL ISETFLG(THM(I),IFLAG)                 !Replace original flag.
      ELSE
        THM(I) = 0.0                               !Set thermistors to failed.
        CALL ISETFLG(THM(I),3)                     !and flag as such
      ENDIF
      END DO

      RETURN
      END


C-------------------------------------------------------------------------------
C ROUTINE          RMEANOF        SUBROUTINE     FORTVAX         [C_RFLUX.FOR]
C
C PURPOSE          Calculate the mean of an array of real values.
C
C DESCRIPTION      An array containing NOELS  real elements is received.
C                  Each element is checked and, if it has a Flag value
C                  (bits 16+17) of zero, is accumulated to a total, and
C                  the  count of good elements incremented.
C                  When all elements have been checked, the mean is derived
C                  such that:
C                  If no good elements were found, the mean is zero, flagged 3.
C                  Otherwise, the mean is the total/count, flagged 0.
C
C ARGUMENTS        INTEGER*4  NOELS IN   Number of elements in array passed
C                  REAL*4     RARR  IN   Array of reals - dimensioned to NOELS
C                  REAL*4     RMEAN OUT  Arithmetic mean of good samples, or 0.
C                  INTEGER*4  IFLAG OUT  Flag value of mean, 0:good 3:invalid.
C
C VERSION          1.00   19-03-90  A.D.HENNINGS
C
C SUBPROGRAMS      ITSTFLG  ISETFLG
C
C REFERENCES       None
C
C------------------------------------------------------------------------------

      SUBROUTINE RMEANOF(NOELS,RARR,RMEAN,IFLAG)
CDEC$ IDENT 'V1.00'
C
```

(continues on next page)

(continued from previous page)

```
      IMPLICIT NONE
      INTEGER*4 NOELS,IX,ITSTFLG,ICOUNT,IFLAG
      REAL*4    RARR(NOELS),RMEAN,SUMM

      SUMM = 0.
      ICOUNT   = 0
      DO IX= 1,NOELS
      IF (ITSTFLG(RARR(IX)) .EQ. 0) THEN
        SUMM = SUMM + RARR(IX)
        ICOUNT = ICOUNT+1
      ENDIF
      END DO

      IF (ICOUNT .GT. 0 )THEN
        RMEAN = SUMM/FLOAT(ICOUNT)
        IFLAG = 0
      ELSE
        RMEAN  = 0.
        IFLAG  = 3
      ENDIF
      CALL ISETFLG(RMEAN,IFLAG)

      RETURN
      END
*--------------------------------------------------------------
C ROUTINE          CIRC_AVRG  FUNCTION   FORTVAX
C
C PURPOSE          CALCULATE MEAN OF A SET (>2 <1000)  OF ANGLES, IN DEG.
C
C ARGUMENTS        REAL*4    ARR  IN         Array of Angles (in  Degrees)
C                  INTEGER*4 NUM  IN         Number of angle in array ARR.
C                  REAL*4    CIRC_AVANG OUT  Average angle of set (0-360 deg)
C
C DESCRIPTION      Given a set of angles (0-360 Deg) calculates their mean.
C                  Handles values spanning 0 or 180.
C                  Returns mean    Flagged 0: If >2 and <= 1/2 of inputs valid
C                                          1: If < 1/2 of inputs valid.
C                                          3: If no valid inputs.
C                  N.B  ASSUMES ALL INPUT ANGLES ARE BETWEEN 0 & 360 DEG.
C
C VERSION          1.0   JAN 1992  A D HENNINGS
C                        MODIFIED FROM  "AVANG" V3.0 SEP 1984  D OFFILER
C                  1.01  DEC 1997  W D N JACKSON
C                        Stips flags before using data
C-------------------------------------------------------------------------------
      REAL  FUNCTION CIRC_AVRG( ARR , NUM)
CDEC$ IDENT 'V1.00'

      IMPLICIT NONE
      INTEGER NUM,NM1,I,ITSTFLG,ICOUNT,IFLAG
      REAL ARR(NUM)
      REAL TARR(1000),DIF


      DO I=1,NUM
      TARR (I) = ARR(I)                         !Move values to temporary array
      CALL ISETFLG(TARR(I),0)                   !Strip flag
```

(continues on next page)

(continued from previous page)

```
      END DO                                    !as they may be altered later.

C Alter angles to same sign .

      IF ( NUM .GT. 2 ) THEN
         NM1 = NUM - 1
         DO I = 1 , NM1
            DIF = TARR(I) - TARR(I+1)
            IF ( ABS ( DIF ) .GT. 180.0 ) THEN
                     TARR(I+1) = TARR(I+1) + SIGN (360.0 , DIF )
            ENDIF
         ENDDO
      ENDIF

C  Sum the good points.

      CIRC_AVRG= 0.0
      ICOUNT= 0

      DO I = 1 , NUM
         IF (ITSTFLG (ARR(I)) .LE. 1) THEN          !Do check on original array
            CIRC_AVRG = CIRC_AVRG + TARR(I)         !..but use changed data
            ICOUNT =ICOUNT+1
         ENDIF
      ENDDO

C Calculate average.

      IF (ICOUNT .GT. 0 )THEN
          CIRC_AVRG = CIRC_AVRG / FLOAT (ICOUNT )
          IF (ICOUNT .GT. NUM/2 ) THEN          !More than half rejected, then
              IFLAG = 0                         !flag as reduced quality data.
          ELSE
              IFLAG = 1
          ENDIF
      ELSE
          CIRC_AVRG  = 0.
          IFLAG  = 3
      ENDIF

      IF ( CIRC_AVRG .LT.   0.0 ) CIRC_AVRG = CIRC_AVRG + 360.0
      IF ( CIRC_AVRG .GE. 360.0 ) CIRC_AVRG = CIRC_AVRG - 360.0

C Set the flag in the returned value

      CALL ISETFLG(CIRC_AVRG,IFLAG)

      END
```

8.20 c_rvsm.for

```
!
! ROUTINE          C_RVSM SUBROUTINE FORTVAX
!
```

(continues on next page)

(continued from previous page)

```
! PURPOSE           Computes static pressure, pitot-static pressure, and pressure
!                   height from the 146 RVSM altitude and airspeed data.
!
! DESCRIPTION       RVSM altitude is available in ARINC-429 message 203 and is
!                   recorded as by the DRS as a 16 bit signed word, with the
!                   LSB representing 4 feet.
!
!                   RVSM computed airspeed is available in ARINC-429 message
!                   206 and is recorded by the DRS as a 16 bit signed word, with
!                   the LSB representing 1/32 kt, but always zero.
!
!                   These values should be within the system accuracy
!                   specification and do not require calibration.
!
!                   Note that altitude is updated by the RVSM at about 20 Hz
!                   and airspeed is updated at about 10 Hz.  Both signals are
!                   sampled by the DRS at 32 Hz so there will be multiple
!                   values and aliasing effects.
!
! METHOD            For each DRS parameter to be calibrated:
!                   1. If data is FFFF or FFFE or out of range then flag 3
!                   2. Decode the altitude and use the tables in NASA TN D-822
!                      to back compute the static pressure.
!                   3. Decode the airspeed and use fundamental equations to
!                      compute pitot-static pressure.
!                   4. Check the results for being within acceptable values.
!                   5. Set data flag bits (16+17) 0: Good data
!                                                 1: Data of lower quality
!                                                 2: Probably faulty, exceed lims
!                                                 3: Data absent or invalid.
!
!                   Flagging - If a value can't be computed, due to missing data
!                   missing constants, divide be zeroes, etc, a value of 0 is
!                   used, flagged with a three.  If a value is outside its
!                   limits for range, it is flagged with a two.
!                   If there are no problems with the data it is flagged with 0.
!                   Any flags on input data are propagated through subsequent
!                   calculations.
!
!                   Note that routine does not currently apply position error
!                   corrections, nor interpolate missing data.
!
! VERSION           1.00  23/07/03  W.D.N.JACKSON
!
! ARGUMENTS         Inputs:
!                     DRS para 222 RVAL 32 Hz RVSM altitude
!                         para 223 RVAS 32 Hz RVSM computed airspeed
!
!                   Outputs:
!                     Derived para 576 SPR  mb 32 Hz Static pressure
!                             para 577 PSP  mb 32 Hz Pitot-static pressure
!                             para 578 PHGT m  32 Hz Pressure height
!
!                   Flags:
!                     Missing/corrupt data output as 0 flagged 3.
!                     Out of range derived data flagged 2.
!
```

(continues on next page)

(continued from previous page)

```
! SUBPROGRAMS      S_PSP, ALT2PRESS, ISETFLG
!
! REFERENCES       NASA Technical Note D-822, Aug 1961, Tables of airspeed,
!                  altitude, and mach number.
!
!                  Interface Control Document, Air Data Display Unit, ISS
!                  1G-80130-22.
!
! CHANGES          V1.00 23/07/03  WDNJ Original version
!                  V1.01 23/10/03  WDNJ Now replicates data when missing
!                  V1.02 11/12/03  WDNJ Fixes bug if initial data missing
!                  V1.03 11/03/04  DAT Flags data outside altitude range 3
!                  V1.04 17/03/04  WDNJ Now handles negative heights correctly
!                                       and uses more accurate flagging criteria
!
!*******************************************************************************
      SUBROUTINE C_RVSM(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.04'
!
      INTEGER*4 IRAW(64,512),IFRQ(512),IS,IVAL,IFLG1,IFLG2,ILSTVAL1,
     &    ILSTVAL2
      REAL*4    RCONST(64),RDER(64,1024),RVAL,RALT,RCAS,RPSP,RSTP
      DATA      ILSTVAL1 /'FFFE'X/, ILSTVAL2 /'FFFE'X/
!
      PARAMETER RSTPMX=1050.            !Max static pressure (mb)
      PARAMETER RSTPMN=116.             !Min static pressure (mb)
      PARAMETER RPSPMX=159.             !Max Pitot-static press (mb) 305 kts at SL!
      PARAMETER RPSPMN=0.               !Min Pitot-static pressure (mb)
      SAVE ILSTVAL1,ILSTVAL2
!
! Derive static pressure, pressure height, and pitot-static.
!
      DO IS=1,32
        IFLG1=0
        RDER(IS,576)=0.
        RDER(IS,578)=0.
! Process height and pressure
        IVAL=IRAW(IS,222)
        IF((IVAL.AND.'FFFF'X).EQ.'FFFE'X) IVAL=ILSTVAL1
        ILSTVAL1=IVAL
        IF(((IVAL.AND.'FFFF'X).EQ.'FFFF'X.AND.IRAW(IS,223).EQ.'FFFF'X)
     &       .OR.(IVAL.AND.'FFFF'X).EQ.'FFFE'X) IFLG1=3
        IF(BTEST(IVAL,15)) IVAL=IVAL.OR.'FFFF0000'X !Extend sign
        IF(IVAL.LT.-250.OR.IVAL.GT.12500) IFLG1=3 !Outside table range
        IF(IFLG1.NE.3) THEN
          RVAL=FLOAT(IVAL)*4.           !Altitude in feet
          RALT=RVAL*0.3048              !Altitude in m
          CALL ALT2PRESS(RVAL,RSTP)     !Compute static pressure in mb
          IF(RSTP.LT.RSTPMN.OR.RSTP.GT.RSTPMX) IFLG1=2
          RDER(IS,576)=RSTP
          RDER(IS,578)=RALT
        END IF
        CALL ISETFLG(RDER(IS,576),IFLG1)
        CALL ISETFLG(RDER(IS,578),IFLG1)
! Process airspeed
        IFLG2=0
        RDER(IS,577)=0.
```

(continues on next page)

(continued from previous page)

```
          IVAL=IRAW(IS,223)
          IF((IVAL.AND.'FFFF'X).EQ.'FFFE'X) IVAL=ILSTVAL2 !No Arinc 429 signal
          ILSTVAL2=IVAL
          IF((IVAL.AND.'FFFF'X).EQ.'FFFF'X.OR.
     &   (IVAL.AND.'FFFF'X).EQ.'FFFE'X) IFLG2=3
          IF(BTEST(IVAL,15)) IVAL=IVAL.OR.'FFFF0000'X !Extend sign
          IF(IVAL.LT.0) IFLG2=3           !Should always be +ve
          IF(IVAL/32.GT.350) IFLG2=3      !Gross error (max 146 IAS is 305 kts)
          IF(IFLG1.NE.3.AND.IFLG2.NE.3) THEN
            IVAL=IVAL.AND.'FFFFFFF7'X     !Clear padding in LSB
            RCAS=FLOAT(IVAL)/32.          !computed airspeed in kt
            RCAS=RCAS*0.514444            !computed airspeed in m/s
            CALL S_PSP(RCAS,RSTP,RPSP)    !Compute pitot-static pressure in mb
            IF(RPSP.LT.RPSPMN.OR.RPSP.GT.RPSPMX) IFLG2=2
            RDER(IS,577)=RPSP
          END IF
          CALL ISETFLG(RDER(IS,577),MAX(IFLG1,IFLG2))
        END DO
!
        RETURN
        END
!*******************************************************************************
        SUBROUTINE S_PSP(RCAS,RSTP,RPSP)
CDEC$ IDENT 'V1.00'
!
! Computes pitot-static pressure from indicated (computed) airspeed and static
! pressure from the following equations (see S_MACH and C_AIRSPD modules):
!
! IAS = 340.294 * Mach * SQRT(Static/1013.25)
! Mach= SQRT(5*((1+Pitot/Static)**(2/7)-1))
!
! where 340.294 is the speed of sound in m/s.
!
! RCAS - Computed airspeed (m/s)
! RSTP - Static pressure (mb)
! RPSP - Pitot-static pressure (mb)
!
        REAL*4 RCAS,RSTP,RPSP,RMACH

        RMACH=RCAS/340.294/SQRT(RSTP/1013.25)
        RPSP=RSTP*((((RMACH**2.)/5.+1.)**3.5)-1.)
        RETURN
        END
!*******************************************************************************
        SUBROUTINE ALT2PRESS(RALT,RPRESS)
CDEC$ IDENT 'V1.00'
!
! Converts altitudes in feet to pressures in mb using the tables provided in
! NASA Technical Note D-822 (Tables of airspeed, altitude and mach number based
! on latest international values for atmospheric properties and physical
! constants. Sadie P Livingston and William Gracey. August 1961).  If altitude
! is outside the range -1000 to 50000 ft then returns a pressure of 0 mb.
!
! This routine is provided to convert the altitudes provided by the 146 RVSM
! system (Innovative Solutions & Support Inc Air Data Display Unit) into
! static pressures, using the same standard tables as are used by the RVSM
! system to convert pressure into altitude.
```

(continues on next page)

(continued from previous page)

```
!
! Pressure values in the NASA tables are given in lb/sq ft.  These have been
! converted to mb using 1 lb/sq in = 68.9476258 mb and 144 sq in = 1 sq ft.
!
! Only simple linear interpolation is used between the tabulated values. This
! will give maximum error of 0.005 mb which is well below the recorded
! resolution, let alone the system accuracy.
!
! V1.00  14/05/03  W.D.N.Jackson
!
      REAL*4 RALT,RPRESS,RTABLE(2,442)
      INTEGER*4 IL,IP,IH

      DATA RTABLE(1:2,1:88) / !Heights (ft) and pressures (mb)
     & -1000.0,1050.408,
     &  -900.0,1046.644,
     &  -800.0,1042.890,
     &  -700.0,1039.146,
     &  -600.0,1035.416,
     &  -500.0,1031.691,
     &  -400.0,1027.985,
     &  -300.0,1024.284,
     &  -200.0,1020.597,
     &  -100.0,1016.915,
     &     0.0,1013.252,
     &   100.0,1009.594,
     &   200.0,1005.951,
     &   300.0,1002.312,
     &   400.0,998.6872,
     &   500.0,995.0770,
     &   600.0,991.4716,
     &   700.0,987.8806,
     &   800.0,984.2991,
     &   900.0,980.7273,
     &  1000.0,977.1649,
     &  1100.0,973.6171,
     &  1200.0,970.0739,
     &  1300.0,966.5452,
     &  1400.0,963.0259,
     &  1500.0,959.5163,
     &  1600.0,956.0211,
     &  1700.0,952.5306,
     &  1800.0,949.0544,
     &  1900.0,945.5880,
     &  2000.0,942.1310,
     &  2100.0,938.6836,
     &  2200.0,935.2458,
     &  2300.0,931.8176,
     &  2400.0,928.4037,
     &  2500.0,924.9947,
     &  2600.0,921.5999,
     &  2700.0,918.2147,
     &  2800.0,914.8392,
     &  2900.0,911.4732,
     &  3000.0,908.1168,
     &  3100.0,904.7700,
     &  3200.0,901.4328,
```

(continues on next page)

(continued from previous page)

```
     &  3300.0,898.1098,
     &  3400.0,894.7917,
     &  3500.0,891.4880,
     &  3600.0,888.1891,
     &  3700.0,884.9045,
     &  3800.0,881.6294,
     &  3900.0,878.3640,
     &  4000.0,875.1033,
     &  4100.0,871.8571,
     &  4200.0,868.6204,
     &  4300.0,865.3932,
     &  4400.0,862.1757,
     &  4500.0,858.9677,
     &  4600.0,855.7693,
     &  4700.0,852.5804,
     &  4800.0,849.4012,
     &  4900.0,846.2316,
     &  5000.0,843.0715,
     &  5100.0,839.9209,
     &  5200.0,836.7800,
     &  5300.0,833.6486,
     &  5400.0,830.5268,
     &  5500.0,827.4146,
     &  5600.0,824.3120,
     &  5700.0,821.2189,
     &  5800.0,818.1354,
     &  5900.0,815.0615,
     &  6000.0,811.9971,
     &  6100.0,808.9376,
     &  6200.0,805.8924,
     &  6300.0,802.8568,
     &  6400.0,799.8259,
     &  6500.0,796.8095,
     &  6600.0,793.7979,
     &  6700.0,790.8005,
     &  6800.0,787.8080,
     &  6900.0,784.8251,
     &  7000.0,781.8517,
     &  7100.0,778.8879,
     &  7200.0,775.9337,
     &  7300.0,772.9891,
     &  7400.0,770.0540,
     &  7500.0,767.1238,
     &  7600.0,764.2078,
     &  7700.0,761.2966/
      DATA RTABLE(1:2,89:176) / !Heights (ft) and pressures (mb)
     &  7800.0,758.3951,
     &  7900.0,755.5032,
     &  8000.0,752.6208,
     &  8100.0,749.7479,
     &  8200.0,746.8847,
     &  8300.0,744.0263,
     &  8400.0,741.1822,
     &  8500.0,738.3429,
     &  8600.0,735.5132,
     &  8700.0,732.6930,
     &  8800.0,729.8824,
```

(continues on next page)

(continued from previous page)

```
     &  8900.0,727.0767,
     &  9000.0,724.2852,
     &  9100.0,721.4986,
     &  9200.0,718.7215,
     &  9300.0,715.9540,
     &  9400.0,713.1913,
     &  9500.0,710.4431,
     &  9600.0,707.6995,
     &  9700.0,704.9655,
     &  9800.0,702.2411,
     &  9900.0,699.5215,
     & 10000.0,696.8162,
     & 10100.0,694.1158,
     & 10200.0,691.4250,
     & 10300.0,688.7437,
     & 10400.0,686.0671,
     & 10500.0,683.4003,
     & 10600.0,680.7429,
     & 10700.0,678.0951,
     & 10800.0,675.4521,
     & 10900.0,672.8187,
     & 11000.0,670.1948,
     & 11100.0,667.5806,
     & 11200.0,664.9711,
     & 11300.0,662.3712,
     & 11400.0,659.7809,
     & 11500.0,657.2001,
     & 11600.0,654.6241,
     & 11700.0,652.0578,
     & 11800.0,649.5010,
     & 11900.0,646.9490,
     & 12000.0,644.4065,
     & 12100.0,641.8737,
     & 12200.0,639.3456,
     & 12300.0,636.8271,
     & 12400.0,634.3181,
     & 12500.0,631.8188,
     & 12600.0,629.3242,
     & 12700.0,626.8392,
     & 12800.0,624.3591,
     & 12900.0,621.8884,
     & 13000.0,619.4274,
     & 13100.0,616.9711,
     & 13200.0,614.5244,
     & 13300.0,612.0873,
     & 13400.0,609.6599,
     & 13500.0,607.2322,
     & 13600.0,604.8191,
     & 13700.0,602.4108,
     & 13800.0,600.0120,
     & 13900.0,597.6227,
     & 14000.0,595.2383,
     & 14100.0,592.8586,
     & 14200.0,590.4933,
     & 14300.0,588.1328,
     & 14400.0,585.7771,
     & 14500.0,583.4310,
```

(continues on next page)

(continued from previous page)

```
     & 14600.0,581.0944,
     & 14700.0,578.7626,
     & 14800.0,576.4405,
     & 14900.0,574.1278,
     & 15000.0,571.8200,
     & 15100.0,569.5169,
     & 15200.0,567.2235,
     & 15300.0,564.9396,
     & 15400.0,562.6605,
     & 15500.0,560.3910,
     & 15600.0,558.1262,
     & 15700.0,555.8710,
     & 15800.0,553.6255,
     & 15900.0,551.3846,
     & 16000.0,549.1487,
     & 16100.0,546.9222,
     & 16200.0,544.7054,
     & 16300.0,542.4933,
     & 16400.0,540.2908,
     & 16500.0,538.0931/
      DATA RTABLE(1:2,177:275) / !Heights (ft) and pressures (mb)
     & 16600.0,535.9050,
     & 16700.0,533.7216,
     & 16800.0,531.5431,
     & 16900.0,529.3789,
     & 17000.0,527.2147,
     & 17100.0,525.0601,
     & 17200.0,522.9150,
     & 17300.0,520.7748,
     & 17400.0,518.6441,
     & 17500.0,516.5183,
     & 17600.0,514.4019,
     & 17700.0,512.2904,
     & 17800.0,510.1837,
     & 17900.0,508.0865,
     & 18000.0,505.9990,
     & 18100.0,503.9113,
     & 18200.0,501.8382,
     & 18300.0,499.7697,
     & 18400.0,497.7061,
     & 18500.0,495.6472,
     & 18600.0,493.6028,
     & 18700.0,491.5583,
     & 18800.0,489.5233,
     & 18900.0,487.4980,
     & 19000.0,485.4727,
     & 19100.0,483.4617,
     & 19200.0,481.4507,
     & 19300.0,479.4493,
     & 19400.0,477.4565,
     & 19500.0,475.4686,
     & 19600.0,473.4877,
     & 19700.0,471.5137,
     & 19800.0,469.5462,
     & 19900.0,467.5851,
     & 20000.0,465.6311,
     & 20100.0,463.6833,
```

(continues on next page)

(continued from previous page)

```
     & 20200.0,461.7422,
     & 20300.0,459.8079,
     & 20400.0,457.8802,
     & 20500.0,455.9588,
     & 20600.0,454.0440,
     & 20700.0,452.1356,
     & 20800.0,450.2337,
     & 20900.0,448.3387,
     & 21000.0,446.4498,
     & 21100.0,444.5671,
     & 21200.0,442.6911,
     & 21300.0,440.8219,
     & 21400.0,438.9584,
     & 21500.0,437.1021,
     & 21600.0,435.2515,
     & 21700.0,433.4076,
     & 21800.0,431.5695,
     & 21900.0,429.7386,
     & 22000.0,427.9134,
     & 22100.0,426.0944,
     & 22200.0,424.2816,
     & 22300.0,422.4756,
     & 22400.0,420.6753,
     & 22500.0,418.8817,
     & 22600.0,417.0939,
     & 22700.0,415.3127,
     & 22800.0,413.5373,
     & 22900.0,411.7682,
     & 23000.0,410.0052,
     & 23100.0,408.2480,
     & 23200.0,406.4970,
     & 23300.0,404.7527,
     & 23400.0,403.0137,
     & 23500.0,401.2814,
     & 23600.0,399.5548,
     & 23700.0,397.8340,
     & 23800.0,396.1194,
     & 23900.0,394.4110,
     & 24000.0,392.7084,
     & 24100.0,391.0121,
     & 24200.0,389.3214,
     & 24300.0,387.6364,
     & 24400.0,385.9578,
     & 24500.0,384.2848,
     & 24600.0,382.6176,
     & 24700.0,380.9562,
     & 24800.0,379.3010,
     & 24900.0,377.6515,
     & 25000.0,376.0078,
     & 25100.0,374.3698,
     & 25200.0,372.7376,
     & 25300.0,371.1111,
     & 25400.0,369.4908,
     & 25500.0,367.8758,
     & 25600.0,366.2665,
     & 25700.0,364.6630,
     & 25800.0,363.0652,
```

(continues on next page)

(continued from previous page)

```
     & 25900.0,361.4733,
     & 26000.0,359.8865,
     & 26100.0,358.3059,
     & 26200.0,356.7307,
     & 26300.0,355.1612,
     & 26400.0,353.5970/
      DATA RTABLE(1:2,276:363) / !Heights (ft) and pressures (mb)
     & 26500.0,352.0389,
     & 26600.0,350.4862,
     & 26700.0,348.9387,
     & 26800.0,347.3969,
     & 26900.0,345.8609,
     & 27000.0,344.3302,
     & 27100.0,342.8047,
     & 27200.0,341.2850,
     & 27300.0,339.7705,
     & 27400.0,338.2618,
     & 27500.0,336.7584,
     & 27600.0,335.2607,
     & 27700.0,333.7678,
     & 27800.0,332.2806,
     & 27900.0,330.7987,
     & 28000.0,329.3221,
     & 28100.0,327.8512,
     & 28200.0,326.3856,
     & 28300.0,324.9248,
     & 28400.0,323.4697,
     & 28500.0,322.0199,
     & 28600.0,320.5753,
     & 28700.0,319.1356,
     & 28800.0,317.7015,
     & 28900.0,316.2723,
     & 29000.0,314.8488,
     & 29100.0,313.4301,
     & 29200.0,312.0167,
     & 29300.0,310.6086,
     & 29400.0,309.2057,
     & 29500.0,307.8076,
     & 29600.0,306.4147,
     & 29700.0,305.0272,
     & 29800.0,303.6449,
     & 29900.0,302.2673,
     & 30000.0,300.8947,
     & 30100.0,299.5272,
     & 30200.0,298.1649,
     & 30300.0,296.8076,
     & 30400.0,295.4554,
     & 30500.0,294.1081,
     & 30600.0,292.7655,
     & 30700.0,291.4282,
     & 30800.0,290.0957,
     & 30900.0,288.7684,
     & 31000.0,287.4455,
     & 31100.0,286.1279,
     & 31200.0,284.8155,
     & 31300.0,283.5074,
     & 31400.0,282.2045,
```

(continues on next page)

(continued from previous page)

```
     & 31500.0,280.9060,
     & 31600.0,279.6128,
     & 31700.0,278.3243,
     & 31800.0,277.0406,
     & 31900.0,275.7618,
     & 32000.0,274.4877,
     & 32100.0,273.2184,
     & 32200.0,271.9539,
     & 32300.0,270.6936,
     & 32400.0,269.4387,
     & 32500.0,268.1885,
     & 32600.0,266.9427,
     & 32700.0,265.7017,
     & 32800.0,264.4654,
     & 32900.0,263.2339,
     & 33000.0,262.0067,
     & 33100.0,260.7843,
     & 33200.0,259.5663,
     & 33300.0,258.3534,
     & 33400.0,257.1449,
     & 33500.0,255.9408,
     & 33600.0,254.7414,
     & 33700.0,253.5468,
     & 33800.0,252.3564,
     & 33900.0,251.1705,
     & 34000.0,249.9892,
     & 34100.0,248.8124,
     & 34200.0,247.6402,
     & 34300.0,246.4724,
     & 34400.0,245.3089,
     & 34500.0,244.1502,
     & 34600.0,242.9958,
     & 34700.0,241.8458,
     & 34800.0,240.7000,
     & 34900.0,239.5590,
     & 35000.0,238.4223,
     & 35100.0,237.2895,
     & 35200.0,236.1614/
      DATA RTABLE(1:2,364:442) / !Heights (ft) and pressures (mb)
     & 35300.0,235.0381,
     & 35400.0,233.9187,
     & 35500.0,232.8036,
     & 35600.0,231.6927,
     & 35700.0,230.5862,
     & 35800.0,229.4840,
     & 35900.0,228.3861,
     & 36000.0,227.2925,
     & 36100.0,226.2028,
     & 36200.0,225.1183,
     & 36400.0,222.9646,
     & 36600.0,220.8316,
     & 36800.0,218.7191,
     & 37000.0,216.6267,
     & 37200.0,214.5545,
     & 37400.0,212.5018,
     & 37600.0,210.4688,
     & 37800.0,208.4555,
```

(continues on next page)

(continued from previous page)

```
     & 38000.0,206.4613,
     & 38200.0,204.4857,
     & 38400.0,202.5298,
     & 38600.0,200.5921,
     & 38800.0,198.6731,
     & 39000.0,196.7727,
     & 39200.0,194.8900,
     & 39400.0,193.0256,
     & 39600.0,191.1793,
     & 39800.0,189.3503,
     & 40000.0,187.5385,
     & 40200.0,185.7444,
     & 40400.0,183.9676,
     & 40600.0,182.2075,
     & 40800.0,180.4647,
     & 41000.0,178.7381,
     & 41200.0,177.0283,
     & 41400.0,175.3348,
     & 41600.0,173.6570,
     & 41800.0,171.9961,
     & 42000.0,170.3504,
     & 42200.0,168.7211,
     & 42400.0,167.1070,
     & 42600.0,165.5083,
     & 42800.0,163.9249,
     & 43000.0,162.3563,
     & 43200.0,160.8036,
     & 43400.0,159.2652,
     & 43600.0,157.7416,
     & 43800.0,156.2325,
     & 44000.0,154.7376,
     & 44200.0,153.2572,
     & 44400.0,151.7911,
     & 44600.0,150.3393,
     & 44800.0,148.9010,
     & 45000.0,147.4766,
     & 45200.0,146.0655,
     & 45400.0,144.6679,
     & 45600.0,143.2842,
     & 45800.0,141.9134,
     & 46000.0,140.5560,
     & 46200.0,139.2110,
     & 46400.0,137.8795,
     & 46600.0,136.5603,
     & 46800.0,135.2542,
     & 47000.0,133.9600,
     & 47200.0,132.6787,
     & 47400.0,131.4094,
     & 47600.0,130.1520,
     & 47800.0,128.9072,
     & 48000.0,127.6738,
     & 48200.0,126.4523,
     & 48400.0,125.2424,
     & 48600.0,124.0444,
     & 48800.0,122.8580,
     & 49000.0,121.6825,
     & 49200.0,120.5185,
```

(continues on next page)

(continued from previous page)

```
     & 49400.0,119.3656,
     & 49600.0,118.2236,
     & 49800.0,117.0922,
     & 50000.0,115.9723/

      RPRESS=0.
      IF(RALT.LT.-1000..OR.RALT.GE.50000.) RETURN
      IL=1                             !Find nearest two altitudes
      IH=442
      DO WHILE(IL+1.NE.IH)
        IP=(IH+IL)/2
        IF(RALT.LT.RTABLE(1,IP)) IH=IP
        IF(RALT.GE.RTABLE(1,IP)) IL=IP
      END DO
      RPRESS=RTABLE(2,IL)+(RTABLE(2,IH)-RTABLE(2,IL)) !Linear interpolation
     &     *(RALT-RTABLE(1,IL))/(RTABLE(1,IH)-RTABLE(1,IL))
      RETURN
      END
```

8.21 c_so2.for

```
C
C ROUTINE        C_SO2 SUBROUTINE FORTVAX
C
C PURPOSE        A subroutine to calculate Carbon monoxide.
C
C DESCRIPTION        The SO2 analyser outputs one measurement.
C                This is input to the program as DRS bits, and converted
C                into PPB by multiplying the DRS bits by a calibration factor.
C
C
C TO COMPILE        $FORT C_SO2
C
C VERSION        1.00  8-Jul-2004          D.Tiddeman
C
C ARGUMENTS        IRAW(1,214) - on entry contains the raw SO2 signal
C               RCONST(1,2,3,4) XO and X1 voltage cal for SO2, v to ppb, ppb offs
C                RDER(1,740) - on exit contains the derived SO2 signal
C
C*******************************************************************************
      SUBROUTINE C_SO2(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.00'
      IMPLICIT NONE
      INTEGER*4 IRAW(64,1024),IFRQ(512)
      INTEGER   IFLG
      REAL*4 SO2,RERR
      REAL*4 RCONST(64),RDER(64,1024)
C
C Set default values
C
      RERR=0.
      CALL ISETFLG(RERR,3)
      RDER(1,740)=RERR
```

(continues on next page)

(continued from previous page)

```
C     Copy across raw signals
C
      SO2=FLOAT(IRAW(1,214))
C
C     Convert CO DRS signals first to voltage, then apply voltage to
C     ppb conversion, then add instrument offset.
C
      SO2=(RCONST(1)+SO2*RCONST(2))*RCONST(3)+RCONST(4)
C
      IFLG=0
      IF(IRAW(1,214).EQ.0) IFLG=3
      IF(IRAW(1,214).EQ.'FFFF'X) IFLG=3
      IF(SO2.LT.0.) IFLG=MAX(2,IFLG)
      CALL ISETFLG(SO2,IFLG)
      RDER(1,740)=SO2
C
      RETURN
      END
```

8.22 c_sols.for

```
C ROUTINE      C_SOLS    SUBROUTINE  FORTVAX
C
C PURPOSE     CALIBRATE  PYRANOMETER & PYRGEOMETER RAW SIGNALS AND THERMISTORS.
C
C DESCRIPTION  Apply calibration coefficients to RAW parameters 81-89 and 91-99
C               to obtain uncorrected values of signal flux, zero offset signal
C              (W/m-2) and thermistor output (deg K)  for  each  of  the
C              upward-facing and downward-facing sets of: clear dome & red dome
C              pyranometers and pyrgeometer.
C
C NOTE             The actual configuration is specified by the array
C                  ICONF, which has six elements whose meaning interpreted as:
C                    1,4 : Clear dome pyranometer  (upper/lower)
C                    2,5 : red    "       "           "     "
C                    3,6 : Silver "   pyrgeometer     "     "
C                  (normally: ICONF(1-3) Upper instruments.
C                             ICONF(4-6) Lower instruments.)
C
C                  This value assists the processing of each instrument by
C                  selecting the correct range checking values to use.
C                  Should the configuration aboard the aircraft be changed
C                  the array ICONF should be adjusted accordingly by adding
C                  offsets to the second calibration constant in the constants
C                  file.
C                  e.g. If  the second constant for, say, the second instrument
C                  was changed from 1.23456E-1 to 21.23456E-1, the offset would
C                  decode to "2" after decoding.
C                  This is assigned to ICONF(2) and would imply that the
C                  'channel' contained raw flux, zero-offset and thermistor
C                  values for a red dome - rather than clear dome - pyranometer.
C                  and should be range-checked for that type of output only.
C
C METHOD       For each RAW parameter to be calibrated, for the six instruments:
```

(continues on next page)

(continued from previous page)

```
C
C             1. Check all its required constants are present (Flag <3)
C                (if not, the calibration of that parameter will not proceed)
C                [Also check that the normal configuration of instruments is to
C                be used. Any changes are indicated by the presence of a large
C                offset to the second calibration constant for any instrument.
C                If this is present the offset is decoded to generate a revised
C                ICONF indicator for that instrument. See note below.]
C             2. Obtain the raw signal/zero value and float the result.
C             3. Calibrate by applying the appropriate instrument cal in RCALB
C                (which is loaded from the RCONST arguments) to both raw
C                signal flux and zero offset, which  use the same coefficients
C                The gains are in W/m-2 /DRS count. DRS counts are related
C                to radiometer output Voltage.
C                Note that the output Voltage from the instrument  is the
C                value after being amplified by the head amplifier.
C             4. Range check and Rate-of-change check: (S/R QCPT)
C                - the calibrated signal (Wm-2)
C                - Zero offset           (DRS units)
C                - temperature           (deg K)
C
C             5. Calibrate the thermistor input using two RCALB coefficients.
C                Add 273.15 deg to thermistor results to express the
C                instrument thermopile temperature in degrees Kelvin.
C             6. Check the result is within pre-defined limits
C             7. Set the calibrated values flag bits (16+17) as follows:
C                        0: Good data
C                        1: Data of lower quality
C                        2: Data probably faulty, exceeding limits
C                        3: Data absent or known to be invalid.
C
C VERSION     1.04 250692   A D HENNINGS
C
C ARGUMENTS      RCONST(1)  - REAL*4 IN  Upper Clear dome Signal & Zero const.
C             *  RCONST(2)  - REAL*4 IN  Upper Clear dome Signal & Zero gain.
C                RCONST(3)  - REAL*4 IN  Upper Clear dome Thermistor: constant
C                RCONST(4)  - REAL*4 IN  Upper Clear dome Thermistor: coeff x.
C                RCONST(5)  - REAL*4 IN  Upper Red   dome Signal & Zero const.
C             *  RCONST(6)  - REAL*4 IN  Upper Red   dome Signal & Zero gain.
C                RCONST(7)  - REAL*4 IN  Upper Red   dome Thermistor: constant
C                RCONST(8)  - REAL*4 IN  Upper Red   dome Thermistor: coeff x.
C                RCONST(9)  - REAL*4 IN  Upper I/R   dome Signal & Zero const.
C             *  RCONST(10) - REAL*4 IN  Upper I/R   dome Signal & Zero gain.
C                RCONST(11) - REAL*4 IN  Upper I/R   dome Thermistor: constant
C                RCONST(12) - REAL*4 IN  Upper I/R   dome Thermistor: coeff x.
C                RCONST(13) - REAL*4 IN  Lower Clear dome Signal & Zero const.
C             *  RCONST(14) - REAL*4 IN  Lower Clear dome Signal & Zero gain.
C                RCONST(15) - REAL*4 IN  Lower Clear dome Thermistor: constant
C                RCONST(16) - REAL*4 IN  Lower Clear dome Thermistor: coeff x.
C                RCONST(17) - REAL*4 IN  Lower Red   dome Signal & Zero const.
C             *  RCONST(18) - REAL*4 IN  Lower Red   dome Signal & Zero gain.
C                RCONST(19) - REAL*4 IN  Lower Red   dome Thermistor: constant
C                RCONST(20) - REAL*4 IN  Lower Red   dome Thermistor: coeff x.
C                RCONST(21) - REAL*4 IN  Lower I/R   dome Signal & Zero const.
C             *  RCONST(22) - REAL*4 IN  Lower I/R   dome Signal & Zero gain.
C                RCONST(23) - REAL*4 IN  Lower I/R   dome Thermistor: constant
C                RCONST(24) - REAL*4 IN  Lower I/R   dome Thermistor: coeff x.
```

(continues on next page)

(continued from previous page)

```
C             (*  also contains an offset evaluated to ICONF() ).
C
C                IFRQ(par)  _ INT*4  IN  Input frequency of each sample.
C                IRAW(n,par)- INT*4  IN  Raw instrument voltage conversion.
C                                        (samples n=1; par=81-89, 91-99)
C                RDER(op,opar)REAL*4 OUT Raw flux signal, zero-offset signal
C                                        and instrument temperature.
C                                        (samples op=1; opar=673-690)
C
C
C SUBPROGRAMS  ITSTFLG, ISETFLG
C
C FILES        none
C
C REFERENCES   Equations from MRF Instrument section.
C
C CHANGES      020490 Revised  range limits introduced.                 ADH
C              100191                                                   ADH
C                     a) Range limits revised to allow for Pyranometer changes
C               "     b) New arrays to hold raw input, constants etc for
C                        more straightforward indexing.
C               "     c) Include ICONF to aid reconfiguring instrument types.
C              010891 Range limits for ZERO now in terms of DRS units, revised
C                     limits in Wm-2 for signal.
C              030292 Rates of change checks instituted on all BBR inputs.  ADH
C              120698 Bug fixed in quality control processing when using non-
C                     standard configurations. MDG/WDNJ
C              270600 I/R signal maximum increased to stop flagging good data
C                     value arbitary, as no explanation of numbers found.
C                     1050. > 1500. DAT
C              V1.06  02/10/02  Changed to use 16 bit DRS data.
C              V1.07  27/11/02  Now takes X0 sensitivity constant as well as X1
C              V1.08  22/07/04  Bug so doesn't crash if first data flagged 3
C              V1.09  13/08/04  Quality Control zero limits increased for
C                               16 bit data
```

8.23 c_sun.for

```
C
C ROUTINE          C_SUN         SUBROUTINE FORTVAX  C_SUN.FOR
C
C PURPOSE          PUT SOLAR ZENITH AND AZIMUTH ANGLES IN MFD
C
C DESCRIPTION      Given date, time and location on the earth's
C                  surface this routine puts a solar zenith and
C                  azimuth angle in the array of derived parameters.
C                  It computes a value once every second. The
C                  angles are only obtained if all the flags are
C                  set to less than 3 and the date, time and location
C                  are all within sensible limits. Any flags set on input
C                  are also set in the solar angles derived. If
C                  the input is in error or the flags are set to 3
C                  a value of -99. is returned for ZEN and AZIM.
C                  To test the routine:
```

(continues on next page)

(continued from previous page)

```
C                  $ FOR C_SUN
C                  $ FOR TEST_C_SUN
C                  $ LINK TEST_C_SUN,C_SUN
C                  Ensure contents of files RCONST.DAT and TEST_C_SUN.DAT
C                  contain simulated data you require to test the routine
C                  with.
C
C VERSION          1.02  1st May 1992   J.A.Smith
C
C ARGUMENTS        RDER(1,515)  R*4 IN Time GMT (seconds from midnight)
C                  RDER(1,550)  R*4 IN Omega latitude degrees (north +ve)
C                  RDER(1,551)  R*4 IN Omega longitude degrees (east +ve)
C               or RDER(1,541)  R*4 IN INU latitude degrees (north +ve)
C               or RDER(1,542)  R*4 IN INU longitude degrees (east +ve)
C                  RCONST(1)    R*4 IN Day in month (1-31)
C                  RCONST(2)    R*4 IN Month in year (1-12)
C                  RCONST(3)    R*4 IN Year (eg 1984)
C                  RDER(1,642)  R*4 OUT Solar azimuth in degrees
C                  RDER(1,643)  R*4 OUT Solar zenith in degrees
C
C SUBPROGRAMS      S_SUN , ITSTFLG, ISETFLG
C
C CHANGES          01 Range checks for input data now done in S_SUN
C                     RWS 30/10/90
C                1.02 Check added if time RSECS has reached midnight and
C                     if so to reduce RSECS to less than 86400 s and increase
C                     the date.  JAS 1/5/92
C                1.03 Following the demise of the Omega, now uses INU position
C                     for flights after 30/09/97.  Note that this routine is
C                     now always called by CALIBRATE, even if neither Omega or
C                     INU were available.  WDNJ 20/10/97
C                1.04 Now strips flags from data before use.  WDNJ 22/12/97
C#######################################################################
      SUBROUTINE C_SUN ( IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.04'
C
      INTEGER*4 IRAW(64,512), IFRQ(512), IFLAG(6)
      INTEGER*4 DAYM(12)/31,29,31,30,31,30,31,31,30,31,30,31/
      INTEGER*4 IMIDNIGHTS           ! added for v1.02
      REAL*4    RCONST(64), RDER(64,1024)
      LOGICAL   BAD_INPUT
C

      RSECS = RDER(1,515)             ! Seconds elapsed since midnight GMT
      IDAY = INT(RCONST(1))           ! Date in month
      IMON = INT(RCONST(2))           ! Month in Year
      IYR = INT(RCONST(3))            ! Year
      IF((IYR.EQ.1997.AND.IMON.GE.10).OR.IYR.GT.1997) THEN
        RLAT = RDER(1,541)            ! INU latitude
        RLON = RDER(1,542)            ! INU longitude
      ELSE
        RLAT = RDER(1,550)            ! Omega latitude
        RLON = RDER(1,551)            ! Omega longitude
      END IF
C
      BAD_INPUT = .FALSE.
C
```

(continues on next page)

(continued from previous page)

```
C   Check flags and only proceed if all less than 3
C
      DO I = 1 , 3
      IFLAG(I) = ITSTFLG(RCONST(I))
      ENDDO
      IFLAG(4) = ITSTFLG(RSECS)
      IFLAG(5) = ITSTFLG(RLAT)
      IFLAG(6) = ITSTFLG(RLON)
      CALL ISETFLG(RSECS,0)
      CALL ISETFLG(RLAT,0)
      CALL ISETFLG(RLON,0)
C
      IMAXFL = 0
      DO I = 1 , 6
      IMAXFL = MAX ( IMAXFL , IFLAG(I) )       ! Get highest flag value
      IF (IFLAG(I) .GE. 3)THEN
        BAD_INPUT = .TRUE.
        CALL ISETFLG ( AZIM , 3 )             ! Set invalid data flags
        CALL ISETFLG ( ZEN  , 3 )
      ENDIF
      ENDDO
C
C If input parameters OK proceed
C
      IF ( .NOT. BAD_INPUT )THEN
C.......................................................................
C v1.02 If time has run over midnight reduce RSECS to less than 24 hours of
C        seconds, ( 86400 ). The day of the month IDAY is then increased by
C        the number of midnights passed over.
C        If this gives too many days for the month then IDAY is set to the
C        first day and IMON to the next month.
C        If the data has crossed into a New Year then IMON is set to January
C        and the year is incremented.
C
      IF (RSECS.GE.86400.) THEN
          IMIDNIGHTS = (NINT(RSECS))/86400
          RSECS = RSECS - REAL(IMIDNIGHTS*86400)
          IDAY = IDAY + IMIDNIGHTS
          IF (MOD(IYR,4).NE.0) DAYM(2)=28  !reduce February if not a leap year
          IF (IDAY.GT.DAYM(IMON)) THEN
              IDAY = IDAY - DAYM(IMON)
              IMON = IMON + 1
              IF (IMON.EQ.13) THEN
                  IMON = 1
                  IYR = IYR + 1
              ENDIF
          ENDIF
      ENDIF
C.......................................................................
C
C  Now compute solar zenith and azimuth angle
C
      CALL S_SUN(IDAY,IMON,IYR,RSECS,RLAT,RLON,AZIM,ZEN)

C
C  Flag values with highest input flag value
C
```

(continues on next page)

(continued from previous page)

```
C  If azimuth or zenith angle not computed in S_SUN set flags to 3
C
      IF (AZIM.EQ.-99) THEN
        CALL ISETFLG(AZIM,3)
      ELSE
        CALL ISETFLG(AZIM,IMAXFL)
      ENDIF

      IF (ZEN.EQ.-99) THEN
        CALL ISETFLG(ZEN,3)
      ELSE
        CALL ISETFLG(ZEN,IMAXFL)
      ENDIF
C
      ELSE
      BAD_INPUT = .TRUE.
      AZIM = -99.0
      ZEN = -99.0
      CALL ISETFLG ( AZIM , 3 )              ! Set invalid data flags
      CALL ISETFLG ( ZEN  , 3 )
C
      ENDIF
C
C  Transfer to output array
C
      RDER(1,642) = AZIM
      RDER(1,643) = ZEN
C
      RETURN
      END
```

8.24 c_temps2.for

```
C
C ROUTINE          C_TEMPS2 SUBROUTINE FORTVAX
C
C PURPOSE          Produces calibrated deiced and non-deiced temperatures
C
C DESCRIPTION      Calculates indicated and true air temperatures in K for the
C                  Deiced and Non-Deiced temperature sensors as follows:
C
C                  519 - Indicated Air Temperature  from Deiced     [K] at 32Hz
C                  520 - True Air Temperature       from Deiced     [K] at 32Hz
C                  524 - Indicated Air Temperature  from Non-deiced [K] at 32Hz
C                  525 - True Air Temperature       from Non-deiced [K] at 32Hz
C
C                  Note that this module only processes data recorded on the
C                  146 which only uses one parameter per temperature.
C
C                  The Deiced Temperature is recorded on the DRS at 32Hz as
C                  parameter 10 and the Non-deiced Temperature is recorded on
C                  the DRS as parameter 23.
C
C                  Indicated Air Temperature is derived by application of
```

(continues on next page)

(continued from previous page)

```
C                    the appropriate second order calibration coefficients to the
C                    raw data.
C
C                    A correction for heating from the deicing heater is made to
C                    the deiced indicated air temperature if the heater is
C                    switched on, as indicated by bit 5 of the signal register
C                    (parameter 27) being clear.  This heating correction is
C                    obtained from graphs of Temperature vs Machno in Rosemount
C                    Technical Reports 7597 & 7637.  If Machno is less than
C                    0.1 the data is flagged 1, because the Rosemount graph is
C                    invalid below 0.1, and if Machno below 0.05, a value of 0.05
C                    is use to ensure a valid logarithm.  The algorithm used for
C                    heating correction is:
C
C               (exp(exp(1.171+(log(Machno)+2.738)*(-0.000568*(s+q)-0.452))))*0.1
C
C                    where: s=static pressure       [mbs]
C                           q=pitot static pressure [mbs]
C
C
C                    True Air Temperature is derived as:
C
C                    TAT[K] = (Indicated Air Temperature[K]) /
C                                  (1.0 +(0.2 * MACHNO * MACHNO * TRECFTR))
C
C                    where: MACHNO  is computed by scientific subroutine S_MACH.
C                           TRECFTR is the Temperature recovery factor - used to
C                                   compensate for effects of kinetic heating.
C                                   This is supplied as a constant from the
C                                   flight constants file to this routine.
C
C                                   It can be calculated from flight results of
C                                   slow/fast runs as:
C
C                     (Tindfast-Tindslow)/(Ffast*Tindslow-Fslow*Tindfast)
C
C                                   where: Tind = indicated temperature [K]
C                                          F    = 0.2 * Machno * Machno
C
C                    Flagging:
C
C                    Both deiced and non-deiced temperature calculations follow
C                    a similar scheme for error flagging, with worst case flags
C                    being propagated through the calculations.  Sources of error
C                    flags are:
C
C                        Absence of calibration constants    - flag 3
C                        Absence of recovery factor constant - flag 3
C                        Static pressure errors              - Parameter 576 flag
C                        Pitot pressure errors               - Parameter 577 flag
C                        Max/min/rate of change errors       - flag 2
C                        Mach No less than 0.1               - flag 1
C
C                    Not all the above errors need affect all measurements.  For
C                    instance pressure errors will not affect Indicated Air
C                    Temperatures, unless the deicing heater is on.  Note that
C                    this module cannot be called if any of the raw (not derived)
```

(continues on next page)

(continued from previous page)

```
C                   parameters are missing.  Also note that no raw data on which
C                   this module can be used will be carrying flags (only raw
C                   data transcribed on the Gould computer can carry flags).  If
C                   any temperature has a flag of three, its value is set to
C                   0.0 K (and flagged with a three).
C
C VERSION           1.00  10/09/92  W.D.N.JACKSON
C
C ARGUMENTS
C                   Constants:
C                   RCONST(1)   Recovery factor for Deiced sensor
C                   RCONST(2)   Recovery factor for Non-deiced sensor
C                   RCONST(3)   Deiced X0 calibration constant (deg C)
C                   RCONST(4)   Deiced X1 calibration constant (deg C)
C                   RCONST(5)   Deiced X2 calibration constant (deg C)
C                   RCONST(6)   Non-deiced X0 calibration constant (deg C)
C                   RCONST(7)   Non-deiced X1 calibration constant (deg C)
C                   RCONST(8)   Non-deiced X2 calibration constant (deg C)
C
C                   Inputs:
C                   DEICED TEMPERATURE              [bits 0-15]     Para  10 32Hz
C                   NON DEICED TEMPERATURE          [bits 0-15]     Para  23 32Hz
C                   SIGNAL REGISTER                 [drs units-bcd] Para  27  2Hz
C                   STATIC PRESSURE                 [mbs]           Para 576 32Hz
C                   PITOT STATIC PRESSURE           [mbs]           Para 577 32Hz
C
C                   Outputs:
C                   INDICATED AIR TEMPERATURE  (Deiced)    [K]      Para 519 32Hz
C                   TRUE AIR TEMPERATURE       (Deiced)    [K]      Para 520 32Hz
C                   INDICATED AIR TEMPERATURE  (NonDeiced) [K]      Para 524 32Hz
C                   TRUE AIR TEMPERATURE       (NonDeiced) [K]      Para 525 32Hz
C
C SUBPROGRAMS       S_MACH           Calculates Mach no
C                   ITSTFLG          Examines bits 16,17 for flags
C                   ISETFLG          Sets flag bits 16,17 = 0 --> 3
C                   S_QCPT           Performs range and rate of change check
C
C REFERENCES        Code adapted from C_TEMPS module.  See MRF Internal Note 55 -
C                   'Temperature Measurement Working Group Report' for full
C                   details of C-130 temperature measurement.
C
C CHANGES           V1.01  27/09/02  W.D.N.JACKSON
C                   Changed to handle 16 bit temperature recording.
C                   V1.02  23/05/05  D.A.TIDDEMAN
C                   Temperature heater correction changed to opposite sense
C                   Now raw para 27 bit 5 on = heater on
C*******************************************************************************
      SUBROUTINE C_TEMPS2(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.01'
      INTEGER*4 IRAW(64,512),IFRQ(512)
      REAL*4    RCONST(64),RDER(64,1024),RMACH(32)

      DATA      RLV519,RLV520,RLT519,RLT520/4*0./     !Init first time through
      DATA      RLV524,RLV525,RLT524,RLT525/4*0./     !Init first time through
      DATA      R519ERCNT,R520ERCNT/2*1.0/            !Init first time through
      DATA      R524ERCNT,R525ERCNT/2*1.0/            !Init first time through
```

(continues on next page)

(continued from previous page)

```
      PARAMETER (R519MX=320.,R519MN=203.,R519RG=1.)  !Limits checks TEMPS [K]
      PARAMETER (R520MX=320.,R520MN=203.,R520RG=1.)  !Limits checks TEMPS [K]
      PARAMETER (R524MX=320.,R524MN=203.,R524RG=1.)  !Limits checks TEMPS [K]
      PARAMETER (R525MX=320.,R525MN=203.,R525RG=1.)  !Limits checks TEMPS [K]
C
      SAVE
      RSEC=RDER(1,515)                  !Time in seconds past midnight
      DO IS=1,32                        !Compute mach no for each sample
        RMACH(IS)=0.
        CALL ISETFLG(RMACH(IS),3)
        IF(ITSTFLG(RDER(IS,576)).NE.3.AND.ITSTFLG(RDER(IS,577)).NE.3)
     &    CALL S_MACH(RDER(IS,576),RDER(IS,577),RMACH(IS)) !Compute mach no
        IFLAG=ITSTFLG(RMACH(IS))        !Save its flag
        CALL ISETFLG(RMACH(IS),0)       !Strip flag
        IF(RMACH(IS).LT.0.05) RMACH(IS)=0.05 !Must be small and +ve
        IF(RMACH(IS).LT.0.1) IFLAG=MAX(IFLAG,1) !If airspeed <.1 set flag to 1
        CALL ISETFLG(RMACH(IS),IFLAG)   !Reapply flag
      END DO
C
C Calculate indicated and true deiced temperatures.
C
      ICFLAG=ITSTFLG(RCONST(3))         !Find worst flag on cal constants
      ICFLAG=MAX(ICFLAG,ITSTFLG(RCONST(4)))
      ICFLAG=MAX(ICFLAG,ITSTFLG(RCONST(5)))
      DO IS=1,IFRQ(10)                  !For each sample of data
        RV=FLOAT(IRAW(IS,10))           !Convert to real
C Calibrate to get indicated temperature
        RDER(IS,519)=RCONST(3)+RCONST(4)*RV+RCONST(5)*RV**2+273.16 !Calibrate
        IFLAG=ICFLAG                    !Set flag if constants were invalid
        IP=((IS*IFRQ(27)-1)/IFRQ(10))+1 !Signal register sample (1 or 2)
C If deicing heater is on, correct for the heating effect
        IF(BTEST(IRAW(IP,27),5)) THEN !If heater was on - removed.NOT.23/05/05
          RM=RMACH(IS)                  !Get mach no
          RS=RDER(IS,576)               !Get static pressure
          RP=RDER(IS,577)               !Get pitot-static pressure
          CALL ISETFLG(RM,0)            !Clear any flag bits
          CALL ISETFLG(RS,0)            !Clear any flag bits
          CALL ISETFLG(RP,0)            !Clear any flag bits
          RHCORR=0.1*EXP(EXP(1.171+(ALOG(RM)+2.738)* !Compute heater correction
     -        (-0.000568*(RS+RP)-0.452)))
          RDER(IS,519)=RDER(IS,519)-RHCORR !Apply to indicated temperature
          IFLAG=MAX(IFLAG,ITSTFLG(RMACH(IS))) !Note errors on mach no
          IFLAG=MAX(IFLAG,ITSTFLG(RDER(IS,576))) !Note errors on static
          IFLAG=MAX(IFLAG,ITSTFLG(RDER(IS,577))) !Note errors on pitot
        END IF
C Apply any flags, do quality control, any reflag if necessary
        IF(IFLAG.EQ.3) RDER(IS,519)=0.0 !If completely invalid set to zero
        CALL S_QCPT(RSEC,RLT519,RDER(IS,519),RLV519,R519MX,R519MN,
     -      R519RG,64.0,R519ERCNT,IQFLAG) !Carry out quality control
        IFLAG=MAX(IFLAG,IQFLAG)         !Use QC flag if worse
C Now work out true temperature
        RM=RMACH(IS)                    !Get mach no
        CALL ISETFLG(RM,0)              !Clear any flag bits
        RDER(IS,520)=RDER(IS,519)/(1.0+(0.2*RM**2*RCONST(1))) !Convert to true
        CALL ISETFLG(RDER(IS,519),IFLAG) !Apply flag to indicated temperature
C Apply any flags, do quality control, and reflag if necessary
        IFLAG=MAX(IFLAG,ITSTFLG(RMACH(IS))) !Note any errors on mach no
```

(continues on next page)

(continued from previous page)

```
        IFLAG=MAX(IFLAG,ITSTFLG(RCONST(1))) !Note any errors on recovery factor
        IF(IFLAG.EQ.3) RDER(IS,520)=0.0 !If completely invalid set to zero
        CALL S_QCPT(RSEC,RLT520,RDER(IS,520),RLV520,R520MX,R520MN,
     -      R520RG,64.0,R520ERCNT,IQFLAG) !Carry out quality control
        IFLAG=MAX(IFLAG,IQFLAG)         !Use QC value if worse
        CALL ISETFLG(RDER(IS,520),IFLAG) !Apply flag to true temperature
      END DO                            !Get next temperature sample
C
C Calculate indicated and true non-deiced temperatures.
C
      ICFLAG=ITSTFLG(RCONST(6))         !Find worst flag on cal constants
      ICFLAG=MAX(ICFLAG,ITSTFLG(RCONST(7)))
      ICFLAG=MAX(ICFLAG,ITSTFLG(RCONST(8)))
      DO IS=1,IFRQ(23)                  !For each sample of data
        RV=FLOAT(IRAW(IS,23))           !Convert to real
C Calibrate to get indicated temperature
        RDER(IS,524)=RCONST(6)+RCONST(7)*RV+RCONST(8)*RV**2+273.16 !Calibrate
C Apply any flags, do quality control, any reflag if necessary
        IFLAG=ICFLAG                    !Set flag if constants were invalid
        IF(IFLAG.EQ.3) RDER(IS,524)=0.0 !If completely invalid set to zero
        CALL S_QCPT(RSEC,RLT524,RDER(IS,524),RLV524,R524MX,R524MN,
     -      R524RG,64.0,R524ERCNT,IQFLAG) !Carry out quality control
        IFLAG=MAX(IFLAG,IQFLAG)         !Use QC flag if worse
C Now work out true temperature
        RM=RMACH(IS)                    !Get mach no
        CALL ISETFLG(RM,0)              !Clear any flag bits
        RDER(IS,525)=RDER(IS,524)/(1.0+(0.2*RM**2*RCONST(2))) !Convert to true
        CALL ISETFLG(RDER(IS,524),IFLAG) !Apply flag to indicated temperature
C Apply any flags, do quality control, and reflag if necessary
        IFLAG=MAX(IFLAG,ITSTFLG(RMACH(IS))) !Note any errors on mach no
        IFLAG=MAX(IFLAG,ITSTFLG(RCONST(2))) !Note any errors on recovery factor
        IF(IFLAG.EQ.3) RDER(IS,525)=0.0 !If completely invalid set to zero
        CALL S_QCPT(RSEC,RLT525,RDER(IS,525),RLV525,R525MX,R525MN,
     -      R525RG,64.0,R525ERCNT,IQFLAG) !Carry out quality control
        IFLAG=MAX(IFLAG,IQFLAG)         !Use QC value if worse
        CALL ISETFLG(RDER(IS,525),IFLAG) !Apply flag to true temperature
      END DO                            !Get next temperature sample
C
      RETURN
      END
```

8.25 c_tpress.for

```
!
! ROUTINE          C_TPRESS SUBROUTINE FORTVAX
!
! PURPOSE          Calibrates the five turbulence probe pressure transducers
!                  into mb.
!
! DESCRIPTION      Apply calibration the combined transducer and DRS
!                  coefficients to DRS parameters 215 to 219 to obtain derived
!                  parameters 773 to 777.  Invalid data is flagged with 3, data
!                  outside limits is flagged with 2.
!
```

(continues on next page)

(continued from previous page)

```
! METHOD            For each DRS parameter to be calibrated:
!                   1. If data is FFFF or FFFE then flag 3
!                   2. Apply the calibration constants
!                   3. Check the results for being within acceptable values.
!                   4. Set data flag bits (16+17) 0: Good data
!                                                 1: Data of lower quality
!                                                 2: Probably faulty, exceed lims
!                                                 3: Data absent or invalid.
!
!                   Flagging - If a value can't be computed, due to missing data
!                   missing constants, divide be zeroes, etc, a value of 0 is
!                   used, flagged with a three.  If a value is outside its
!                   limits for range or rate of change, it is flagged with a two.
!                   If there are no problems with the data it is flagged with 0.
!
! VERSION           1.00  23/07/03  W.D.N.JACKSON
!
! ARGUMENTS         Inputs:
!                     DRS para 215 TBP1 32 Hz Turbulence probe centre port
!                         para 216 TBP2 32 Hz Turbulence probe attack ports
!                         para 217 TBP3 32 Hz Turbulence probe sideslip ports
!                         para 218 TBP4 32 Hz Turbulence probe attack check
!                         para 219 TBP5 32 Hz Turbulence probe sideslip check
!
!                   Constants:
!                         RCONST(1 to 4) Para 215 cal constants X0 to X3
!                         RCONST(5 to 8) Para 216 cal constants X0 to X3
!                         RCONST(9 to 12) Para 217 cal constants X0 to X3
!                         RCONST(13 to 14) Para 218 cal constants X0 to X1
!                         RCONST(15 to 16) Para 219 cal constants X0 to X1
!
!                   Outputs:
!                     Derived para 773 TBP0 mb 32 Hz Centre pressure
!                             para 774 TBPA mb 32 Hz Attack pressure
!                             para 775 TBPB mb 32 Hz Sideslip pressure
!                             para 776 TBPC mb 32 Hz Attack check pressure
!                             para 777 TBPD mb 32 Hz Sideslip check pressure
!
!                   Flags:
!                     Missing/corrupt data output as 0 flagged 3.
!                     Out of range data flagged 2.
!
! SUBPROGRAMS       ISETFLG
!
! REFERENCES
!
! CHANGES           V1.00 23/07/03  WDNJ Original version
!                   Note that V1.00 has no limit checking and no use is made of
!                   the check pressures.
!                   V1.01 25/03/04  WDNJ
!                   Now takes third order calibration constants for the main
!                   transducers, and first order for the check transducers.
!                   V1.02 26/01/06 Phil Brown
!                   Realistic min/max values provided for centre-port, Pa, Pb
!                   for flagging purposes. Values alsoe provided for check
!                   pressures Ca, Cb based on current (and probably wrong)
!                   calibration coefficients.
```

(continues on next page)

(continued from previous page)

```
!                    V1.03 09/02/11 Axel Wellpott
!                    From an email from Phil Brown: "The P0-S10 differential pressure
!                    (para 773) is flagged 2 if it exceeds 130.0 hPa. This is easily
!                    exceeded when we do acceleration to max speed (min Angle of Attack)
!                    so all the subsequent parameters calculated n C_TURB.for end up␣
↪with a
!                    flag-3 saetting. I reckon a better value would be 180.0 hPa."
!
!*******************************************************************************
      SUBROUTINE C_TPRESS(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.03'
!
      INTEGER*4 IRAW(64,512),IFRQ(512),IS,IP,IFLG,IVAL
      REAL*4    RCONST(64),RDER(64,1024),RVAL,RMAX(5),RMIN(5)
      DATA RMIN /30.,-30.,-20.,50.,50./
!      DATA RMAX /130.,30.,20.,200.,200./
!     Values changed on 09/02/2011
      DATA RMAX /180.,30.,20.,200.,200./
!
! Derive port pressures.  Note that by default derived parameters will have a
! value of 0 flagged with a 3.
!
      DO IP=1,5                             !For each parameter
        DO IS=1,IFRQ(215)                   !For each sample
          IFLG=0
          IVAL=IRAW(IS,214+IP)
          IF(IVAL.EQ.'FFFF'X.OR.IVAL.EQ.'FFFE'X) IFLG=3
          IF(IFLG.NE.3) THEN
            RVAL=FLOAT(IVAL)
            IF(IP.EQ.1) THEN
              RVAL=RCONST(1)+RVAL*RCONST(2)+RVAL*RVAL*RCONST(3)
     &             +RVAL*RVAL*RVAL*RCONST(4)
            ELSE IF(IP.EQ.2) THEN
              RVAL=RCONST(5)+RVAL*RCONST(6)+RVAL*RVAL*RCONST(7)
     &             +RVAL*RVAL*RVAL*RCONST(8)
            ELSE IF(IP.EQ.3) THEN
              RVAL=RCONST(9)+RVAL*RCONST(10)+RVAL*RVAL*RCONST(11)
     &             +RVAL*RVAL*RVAL*RCONST(12)
            ELSE IF(IP.EQ.4) THEN
              RVAL=RCONST(13)+RVAL*RCONST(14)
            ELSE IF(IP.EQ.5) THEN
              RVAL=RCONST(15)+RVAL*RCONST(16)
            END IF
            IF(RVAL.LT.RMIN(IP).OR.RVAL.GT.RMAX(IP)) IFLG=2
            RDER(IS,772+IP)=RVAL
            CALL ISETFLG(RDER(IS,772+IP),IFLG)
          END IF
        END DO
      END DO
!
      RETURN
      END
```

8.26 c_turb.for

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!         ROUTINE      C_TURB
!
!         PURPOSE      To calibrate and apply designated correction factors to
!                      angle of attack (AOA), angle of sideslip (AOSS) and the
!                      centre-static differential pressure (to derive TAS)).
!
!         DESCRIPTION  Calibration of AOA and AOSS is assumed to be of the form:
!
!                      PA/q = a0(M) + a1(M)*alpha
!                      PB/q = b0(M) + b1(M)*beta
!                      where q is the pitot(dynamic) pressure.
!                      Calculations follow the scheme described in BAES doc
!                      ADE-46S-R-463-34 1233 (Page 78 of 116).
!                      Initial value of pitot pressure is taken from RVSM and
!                      used to calculate first guess AOA and AOSS. These are
!                      to derive corrections to the centre-port along with
!                      separate calculation of static position error in the
!                      centre-port measurement. AOA and AOSS are recalculated
!                      with iteration continuing until specified tolerance is
!                      achieved or max.iteration count exceeded. Corrected
!                      centre-port pressure is then used to calculate TAS
!                      (currently only the dry value) using:
!
!                      TAS = Corrtn.fac * 340.294*M*SQRT(T/288.15)
!
!         VERSION      1.01   Phil Brown 24/5/2004
!
!         CHANGES      1.02   Phil Brown 11/6/2004
!                             Logic changed to reproduce PVWAVE test version
!                             MRFB:[BROWN.PVWAVE]TURB.PRO at this date
!                      1.03   Phil Brown 28/6/2004
!                             Check flags and values following return of calls
!                             to S_MACH. Unacceptable causes C_TURB to return
!                             its default values of output parameters (flag 3)
!                      1.04   Phil Brown 2/7/04
!                             Uses G_MACH routine to calculate Mach no. and
!                             avoid complications due to flagging.
!                      1.05   Phil Brown 08/07/04
!                             Uses simpler Mach-dependent PE.Corrtn derived
!                             empirically from B001-012 s&l legs.
!                      1.06   Phil Brown 09/07/04
!                             No position error correction currently applied
!                             to P0 differential pressure.
!                      1.07   Phil Brown 26/08/04
!                             Change sign of AOSS. Cals were done against INS
!                             drift angle (-ve for +ve AOSS).
!                      1.08   Phil Brown 27/8/04
!                             AOSS calcs revert to original, but assumed to use
!                             new fit coefficients for B0 and B1
!                      1.09   26/01/06 Phil Brown
!                             Min/max limits provided for AoA, AoSS and TAS for
!                             flagging purposes.
!                      1.10   20/06/06 Phil Brown
!                             Takes additional input of non-deiced temp, used as
```

(continues on next page)

(continued from previous page)

```
!                             alternative when de-iced is flagged 2 or more.
!                      1.11   24/10/06 Phil Brown
!                             Fix bug setting flag on TTND to zero before use.
!                             Define 4 additional flight constants to apply
!                             fudge factors to the calculated AoA / AoSS
!                             These have the form:
!                             AoA_new = AoA * ALPH1 + ALPH0
!                             AoSS_new= AoSS * BET1 + BET0
!                      1.12   08/10/2010 Axel Wellpott
!                             added line "DATA TAS/-9999./"
!                             Missing TAS data values were set to -999.
!                             and not to -9999. as specified in the netcdf
!                             files.
!
!         SUBROUTINES: S10_PECORR, ITSTFLG, ISETFLG, G_MACH
!
!         FILES
!
!         REFERENCES
!
!         CONSTANTS
!           RCONST(1-3)  Coefficients of 2nd order polynomial in Mach to
!                        calculate AOA offset, A0
!           RCONST(4-6)  Coefficients of 2nd order polynomial in Mach to
!                        calculate AOA sensitivity, A1
!           RCONST(7-9)  Coefficients of 2nd order polynomial in Mach to
!                        calculate AOSS offset, B0
!           RCONST(10-12) Coefficients of 2nd order polynomial in Mach to
!                        calculate AOSS sensitivity, B1
!           RCONST(13)   Tolerance for AOA/AOSS iteration
!           RCONST(14)   True Airspeed correction factor (fudge factor to
!                        remove residual along-heading wind errors).
!           RCONST(15-16) Coefficients of linear correction to calculated AoA
!           RCONST(17-18) Coefficients of linear correction to calculated AoSS
!
!         INPUT PARAMETERS
!           516  IAS    32Hz
!           520  TTDI   32Hz
!           525  TTND   32Hz
!           576  SPR    32Hz
!           577  PSP    32Hz
!           578  PHGT   32Hz
!           773  TBP0   32Hz
!           774  TBPA   32Hz
!           775  TBPB   32Hz
!           776  TBPC   32Hz
!           777  TBPD   32Hz
!
!         OUTPUT PARAMETERS
!
!           548  AOA    32Hz  deg
!           549  AOSS   32Hz  deg
!           779  TASD   32Hz  ms-1
!           780  TASW   32Hz  ms-1
!           781  TPSP   32Hz  mb
!
!         TURBULENCE PROBE CONSTANT KEYWORDS
```

(continues on next page)

(continued from previous page)

```
!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      SUBROUTINE C_TURB(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.12'
      IMPLICIT NONE
      INTEGER*4 IRAW(64,512),IFRQ(512),I,J
      INTEGER*4 ITSTFLG, ITTDIFLG, ITTNDFLG, ISPRFLG, IPSPFLG, IPHGTFLG
      INTEGER*4 IP0FLG, IPAFLG, IPBFLG, ICAFLG, ICBFLG, IMACHFLG
      INTEGER*4 IASFLG, IFLAG
      REAL*4    RCONST(64),RDER(64,1024)
      REAL*4    AIAS,TTDI,TTND,SPR,PSP,PHGT,TP0,TPA,TPB,CA,CB
      REAL*4    AOA, AOANEW, AOSS, AOSSNEW, TOL
      REAL*4    DCP_S10, DCPA, DCPB, P0, Q
      REAL*4    AMACH, A0, A1, B0, B1, DAOA, DAOSS, TAS
      DATA TAS/-9999./                    ! set all TAS values to -9999.
!
      REAL*4    AIASMIN,TASMIN,TASMAX
      DATA AIASMIN,TASMIN,TASMAX/50.0,50.,250./
!
      REAL*4    AOAMIN,AOAMAX,AOSSMIN,AOSSMAX
      DATA AOAMIN,AOAMAX,AOSSMIN,AOSSMAX/0.,15.0,-5.0,5.0/
!
      INTEGER*4 ITER, ITERMAX, MAXITER
      DATA ITER,ITERMAX,MAXITER/0,5,0/    ! iteration counters, within each
!                                         ! sample and each second
!
      DO I=1,32                          ! set placeholder values and flags
        RDER(I,548)=-99.9                 !Set AoA to 0
        CALL ISETFLG(RDER(I,548),3)
        RDER(I,549)=-99.9                 !Set AoSS to 0
        CALL ISETFLG(RDER(I,549),3)
        RDER(I,779)=-9999.                 !Set dry turbulence probe airspeed
        CALL ISETFLG(RDER(I,779),3)
        RDER(I,780)=-9999.                 !Set wet turbulence probe airspeed
        CALL ISETFLG(RDER(I,780),3)
        RDER(I,781)=-950.                 !Set turbulence probe pitot-static
        CALL ISETFLG(RDER(I,781),3)
      END DO

! now do the real calculations
      MAXITER = 0

      DO I=1,32                          ! outer loop over 32 samples
        AIAS = RDER(I,516)
        IASFLG   = ITSTFLG(AIAS)         ! preserve flags from all inputs params
        CALL ISETFLG(AIAS,0)             ! and reset flag to zero
        TTDI = RDER(I,520)
        ITTDIFLG = ITSTFLG(TTDI)
        CALL ISETFLG(TTDI,0)
        TTND = RDER(I,525)
        ITTNDFLG = ITSTFLG(TTND)
        CALL ISETFLG(TTND,0)                    ! V1.11 bug fix here
        SPR  = RDER(I,576)
        ISPRFLG  = ITSTFLG(SPR)
        CALL ISETFLG(SPR,0)
        PSP  = RDER(I,577)
```

(continues on next page)

(continued from previous page)

```
        IPSPFLG   = ITSTFLG(PSP)
        CALL ISETFLG(PSP,0)
        PHGT = RDER(I,578)
        IPHGTFLG = ITSTFLG(PHGT)
        CALL ISETFLG(PHGT,0)
        TP0    = RDER(I,773)
        IP0FLG    = ITSTFLG(TP0)
        CALL ISETFLG(TP0,0)
        TPA    = RDER(I,774)
        IPAFLG    = ITSTFLG(TPA)
        CALL ISETFLG(TPA,0)
        TPB    = RDER(I,775)
        IPBFLG    = ITSTFLG(TPB)
        CALL ISETFLG(TPB,0)
        CA    = RDER(I,776)
        CB    = RDER(I,777)

!         IF(AIAS.LE.AIASMIN) THEN
!           PRINT *,'Inputs'
!           PRINT *,AIAS,TTDI,SPR,PSP,PHGT,TP0,TPA,TPB,CA,CB
!           PRINT *,'RDER(515-520)'
!           PRINT *,(RDER(I,J),J=515,1024)
!         ENDIF

! Proceed only with acceptable flag settings and IAS > 50m/s
        IF(ISPRFLG.LE.1.AND.IPSPFLG.LE.1.AND.IPHGTFLG.LE.1.AND.
     -     IP0FLG.LE.1.AND.IPAFLG.LE.1.AND.IPBFLG.LE.1.AND.
     -     AIAS.GT.AIASMIN) THEN

! Mach number from RVSM pitot pressure
!         CALL ISETFLG(SPR,0)
!         CALL ISETFLG(PSP,0)
        Q = PSP
        CALL G_MACH(SPR, Q, AMACH)
! Check values returned by Mach number calculation
        IF(AMACH.GT.0.0.AND.AMACH.LT.1.0) THEN
!         PRINT *,SPR,PSP,AMACH

! First guess AOA and AOSS
        A0  = RCONST(1)+AMACH*(RCONST(2)+AMACH*RCONST(3))
        A1  = RCONST(4)+AMACH*(RCONST(5)+AMACH*RCONST(6))
        B0  = RCONST(7)+AMACH*(RCONST(8)+AMACH*RCONST(9))
        B1  = RCONST(10)+AMACH*(RCONST(11)+AMACH*RCONST(12))

!         PRINT *,'A0,A1,B0,B1',A0,A1,B0,B1,AMACH

        AOA   = (TPA/Q - A0)/A1
        AOSS  = (TPB/Q - B0)/B1

! Calculate position error in S10 static pressure.
!         PRINT *,P0,PHGT,PSP,AOA,AOSS
        CALL S10_PECORR(DCP_S10,AMACH)

! Calculate and apply flow angle corrections to derive true pitot pressure
! from centre-port measurement.
        DCPA = 0.0273+ AOA*(-0.0141+ AOA*(0.00193- AOA*5.2E-5))
        DCPB = 0.0    +AOSS*(0.0     + AOSS*7.6172E-4)
```

(continues on next page)

(continued from previous page)

```
! Apply corrections to measured differential pressure
!         P0 = TP0+(DCPA+DCPB+DCP_S10)*Q
        P0 = TP0+(DCPA+DCPB)*Q
        Q = P0
!         PRINT *,'P0 = ',P0

! Recalculate Mach number
!         CALL ISETFLG(SPR,0)
        CALL ISETFLG(Q,0)
        CALL G_MACH(SPR,Q,AMACH)
! Check values returned by Mach number calculation
        IF(AMACH.GT.0.0.AND.AMACH.LT.1.0) THEN
!         PRINT *,SPR,Q,AMACH

        ITER=0
! Recalculate AOA/AOSS
100     ITER=ITER+1
        A0 = RCONST(1)+AMACH*(RCONST(2)+AMACH*RCONST(3))
        A1 = RCONST(4)+AMACH*(RCONST(5)+AMACH*RCONST(6))
        B0 = RCONST(7)+AMACH*(RCONST(8)+AMACH*RCONST(9))
        B1 = RCONST(10)+AMACH*(RCONST(11)+AMACH*RCONST(12))

        AOANEW  = (TPA/Q - A0)/A1
        AOSSNEW = (TPB/Q - B0)/B1

        DAOA = AOANEW-AOA
        DAOSS= AOSSNEW-AOSS
        TOL  = RCONST(13)

        AOA = AOANEW
        AOSS= AOSSNEW

! Recalculate position error correction to S10
        CALL S10_PECORR(DCP_S10,AMACH)

! Recalculate flow-angle corrections to centre-port.
        DCPA = 0.0273+ AOA*(-0.0141+ AOA*(0.00193- AOA*5.2E-5))
        DCPB = 0.0   +AOSS*(0.0    + AOSS*7.6172E-4)

! Apply corrections to measured pressure
!         P0 = TP0+(DCPA+DCPB+DCP_S10)*Q
        P0 = TP0+(DCPA+DCPB)*Q
        Q  = P0

! Recalculate Mach number
!         CALL ISETFLG(SPR,0)
        CALL ISETFLG(Q,0)
        CALL G_MACH(SPR,Q,AMACH)
! Check values returned by Mach number calculation
        IF(AMACH.GT.0.0.AND.AMACH.LT.1.0) THEN

! Test flow angles changes wrt tolerance
        IF((ABS(DAOA).GT.TOL.OR.ABS(DAOSS).GT.TOL)
     *      .AND.ITER.LT.ITERMAX) GOTO 100

! Calculate dry (and later also wet) TAS from Mach number and temperature
```

(continues on next page)

(continued from previous page)

```
!        PRINT *,'AMACH / TTDI', AMACH, TTDI
        IF(ITTDIFLG.LE.1) THEN      ! de-iced OK
          TAS = RCONST(14) * 340.294 * AMACH * SQRT(TTDI/288.15)
        ELSE
          TAS = RCONST(14) * 340.294 * AMACH * SQRT(TTND/288.15)
        ENDIF

! Apply linear corrections to calculated AoA and AoSS, derived from Al Rodi
! analysis - minimization of residual vertical wind during yawing orbits.
        IF((ITSTFLG(RCONST(15)).EQ.0.).AND.
     &     (ITSTFLG(RCONST(16)).EQ.0.).AND.
     &     (ITSTFLG(RCONST(17)).EQ.0.).AND.
     &     (ITSTFLG(RCONST(18)).EQ.0))THEN
            AOA = AOA*RCONST(16) + RCONST(15)
            AOSS= AOSS*RCONST(18) + RCONST(17)
        ENDIF

! Check data flagging and output parameters.

        RDER(I,548) = AOA
        IFLAG=0
        IF(AOA.LT.AOAMIN.OR.AOA.GT.AOAMAX) IFLAG=2
        IFLAG=MAX(IFLAG,IPAFLG)
        CALL ISETFLG(RDER(I,548),IFLAG)

        RDER(I,549) = AOSS
        IFLAG=0
        IF(AOSS.LT.AOSSMIN.OR.AOSS.GT.AOSSMAX) IFLAG=2
        IFLAG=MAX(IFLAG,IPBFLG)
        CALL ISETFLG(RDER(I,549),IPBFLG)

        RDER(I,779) = TAS
        IFLAG=0
        IF(TAS.LT.TASMIN.OR.TAS.GT.TASMAX) IFLAG=2
        IFLAG=MAX(IFLAG,IP0FLG)
        CALL ISETFLG(RDER(I,779),IP0FLG)

        RDER(I,780) = -99.9
        CALL ISETFLG(RDER(I,780),3)
        RDER(I,781) = Q
        CALL ISETFLG(RDER(I,781),IP0FLG)

        ENDIF   ! test on third Mach number calculation
        ENDIF   ! test on second Mach number calculation
        ENDIF   ! test on first Mach number calculation

        ENDIF ! end of test on flag settings

! track maximum iteration count in this 1-second sample
        IF(ITER.GT.MAXITER) MAXITER=ITER

      ENDDO ! end of calculation loop
!      PRINT *,'Max. iteration count this second =',MAXITER

      RETURN
      END
!
```

(continues on next page)

(continued from previous page)

```
      SUBROUTINE S10_PECORR(DCP_S10,AMACH)
CDEC$ IDENT 'V1.00'
!
! PURPOSE  To calculate values of the S10 static pressure position error
!          as a function of Mach number, derived from B001-B012 calibrations
!
! AUTHOR   Phil Brown
!
! VERSION  1.00 08/07/2004
!
      IMPLICIT NONE
      REAL*4 DCP_S10, AMACH, A0,A1,A2
      DATA A0,A1,A2/-0.011482, -0.148295, 0.407040/

      DCP_S10 = A0 + AMACH*(A1 + AMACH*A2)

      RETURN
      END
```

8.27 c_twc.for

```
C
C ROUTINE             C_TWC   subroutine fortvax/fort77
C
C PURPOSE             To calibrate DRS pars. 70-78 into TARDIS parameters 664-672
C
C DESCRIPTION        The same algorithm is used for all nine parameters. First
C                     check to see if the right frequency has been set. Find
C                     the flag of the raw data. Work out the derived parameter,
C                     #665-#671, values of RCONST are used in a polynomial
C                     fit. For #664(Detector) and #672(status word) the raw
C                     data is converted from an integer to a real. Then the
C                     derived data is tested to see if it lies between the
C                     accepted envelope of values for that parameter. The flag
C                     is set to 2 if it lies outside the envelope. If any
C                     other tests are failed the derived parameter is set to
C                     -9999.0 with the flag at 3. At the end, with all the
C                     parameters calculated,  a rate of change check is made.
C                     This looks at the values set in RATE_CHANGE.
C
C                     Derived data limits and rate of change limits;
C
C                     DRS  TARDIS   min    max   rate/change   units
C                     par   par
C                      70    664     0      4094      -          DRS
C                    71    665     314    383      10.0         K
C                    72    666     323    388       3.0         K
C                    73    667     289    343       2.0         K
C                    74    668     378    393       5.0         K
C                    75    669     0.3    6.6       0.5         A
C                    76    670     0.3    6.6       0.5         A
C                    77    671     0.4E-3 1.1E-3    0.05E-3     A
C                    78    672     0      4095      -          DRS
C
```

(continues on next page)

(continued from previous page)

```
C VERSION             1.00 080190 M.J.GLOVER
C
C ARGUMENTS          IRAW(64,512) I*4  IN   Raw data for the parameters
C                     IFRQ(512)    I*4  IN   Frequencies of the data
C                     RCONST(64)   R*4  IN   Constants required by routine,(1-32)
C                     RDER(64,1024)R*4  OUT  Tardis parameters
C
C COMMON             None.
C
C SUBPROGRAMS             ISETFLG (linked automatically)
C
C FILES                     None.
C
C REFERENCES            MRF2 Specification for Total Water Hygrometer 4 Dec 1989
C
C CHANGES            V1.01  10/06/94  W.D.N.JACKSON / S.J.MOSS
C                    Modified to correctly compute evaporator currents when the
C                    modified TWC instrument is flown (ie for A188 onwards).  In
C                    this case DRS parameters 173 and 174 are also used.  If
C                    the CALHTR1 and CALHTR2 keywords in the flight constants
C                    file have four values then processing for the modified
C                    probe is used; if they have two values then the old
C                    processing is used.  Note that parameters 173 and 174 are
C                    optional for this routine and CALIBRATE does not insist
C                    that they are present.  Also note that when this routine is
C                    used for flights before A188 CALIBRATE issues a warning
C                    that some of the constants are absent; this can be ignored.
C
C################################################################################

        SUBROUTINE C_TWC(IRAW, IFRQ, RCONST, RDER)
CDEC$ IDENT 'V1.01'
        INTEGER*4 IRAW(64,512), IFRQ(512)
        REAL*4 RCONST(64), RDER(64, 1024)
        REAL OLD_PARAS(7), RATE_CHANGE(7)

        DATA RATE_CHANGE/10.0, 3.0, 2.0, 5.0, 0.5, 0.5, 0.05E-03/

C        Calibrate the hygrometer detector output - DRS parameter 70, sample
C        rate 64 Hz. To be left as bits.
        IF (IFRQ(70).EQ.64) THEN   ! See if the right frequency is there.
                       DO IS=1, IFRQ(70)  ! Do for each sample.

C                        If the raw data is inside the bounds, process it.
                        IF (((IRAW(IS, 70).AND.'FFF'X).GT.-1).AND.
     1                                  ((IRAW(IS,70).AND.'FFF'X)
     2                                                  .LT.4095)) THEN
C                                IFLAG=ITSTFLG(IRAW(IS, 70))
                                IFLAG=0

C                                     If the flag of the raw data is less than three,
C                                      then convert the raw data into derived data.
                                 IF (IFLAG.LT.3) THEN
                                        RDER(IS, 664)=FLOAT(IRAW(IS,70)
     1                                                 .AND.'FFF'X)

C                                If the flag is three or above, set the
```

(continues on next page)

(continued from previous page)

```
C                                derived data to -9999.0.
                                     ELSE
                                        RDER(IS, 664)=-9999.0
                               END IF
                       ELSE

C                                If the raw data is outside the bounds, set it
C                                to -9999.0.
                               RDER(IS, 664)=-9999.0
                         ENDIF


C                        If the derived data is outside the bounds of 0 and
C                        4094, set the flag to three.
                       IF ((RDER(IS, 664).LT.0.0).OR.
     1                                    (RDER(IS, 664).GT.4094)) THEN
                                     CALL ISETFLG(RDER(IS, 664), 3)
                             ELSE

C                                If the derived data is within the bounds of 0
C                                and 4094, then set the flag to that of the raw
C                                data's.
                               CALL ISETFLG(RDER(IS, 664), IFLAG)
                           END IF
                  END DO
        ELSE

C                If the wrong frequency is there for the detector, then set all
C                the samples for this second to -9999.0, with their flags set
C                to 3.
               DO IS=1, 64
                       RDER(IS, 664)=-9999.0
                       CALL ISETFLG(RDER(IS,664),3)
               END DO
        ENDIF

C        Calibrate the nose temperature - DRS parameter 71, sample rate 1 Hz
C        This is to be put into Kelvin. A do loop is used, as the sample rate
C        may well change. This uses the elements of RCONST from 1 to 5.
        IF (IFRQ(71).EQ.1) THEN  ! check the frequency.
               DO IS=1, IFRQ(71)  ! for each sample

C                        See if all the const are there,if not set the flag to 3
                              ICHECK=1
                       DO I=1,5
                               IF (ITSTFLG(RCONST(I)).GT.2) THEN
                                       ICHECK=ICHECK+1
                               END IF
                       END DO

C                        ICHECK will be more than one if any of the constants
C                        are missing
                       IF (ICHECK.EQ.1) THEN
C                                IFLAG=ITSTFLG(IRAW(IS, 71))
                               IFLAG=0
                       ELSE
                               IFLAG=3
```

(continues on next page)

(continued from previous page)

```
                        END IF

C                        If the flag of the raw data is less than three, then
C                        convert the raw data into derived data. This is done
C                        using a polynomial fit.
                                     IF (IFLAG.LT.3) THEN
                               RAW=FLOAT(IRAW(IS,71).AND.'FFF'X)
                               RDER(IS, 665)=RCONST(1)

                               DO INC=2,5
                                       RDER(IS, 665)=RDER(IS, 665)+
     1                                                     RCONST(INC)*
     2                                                     (RAW**(INC-1))
                                     END DO
                            ELSE

C                                If the flag is three or above, set the
C                                derived data to -9999.0.
                               RDER(IS, 665)=-9999.0
                            END IF


C                        If the derived data is outside the bounds but not
C                        -9999.0, then set the flag to two.
                        IF (((RDER(IS, 665).LT.314.0).OR.
     1                                      (RDER(IS, 665).GT.383.0)).AND.
     2                                      (RDER(IS, 665).GT.-9000.0)) THEN
                               CALL ISETFLG(RDER(IS, 665), 2)
                        ELSE

C                                The derived data is within the limits then
C                                set the flag to that of the raw data. If the
C                                data is -9999.0 the flag will be three.
                                CALL ISETFLG(RDER(IS, 665), IFLAG)
                            ENDIF

                 END DO
        ELSE

C                The data has not got the right frequency.
                RDER(1, 665)=-9999.0
                CALL ISETFLG(RDER(1,665),3)
        ENDIF

C        Calibrate the sample temp -DRS parameter 72, sample rate 1 HZ. This is
C        to be turned into Kelvin. This uses the elements of RCONST from 6 to 11
        IF (IFRQ(72).EQ.1) THEN  ! check the frequency.
                DO IS=1, IFRQ(72)  ! for each sample

C                        See if all the const are there,if not set the flag to 3
                              ICHECK=1
                        DO I=6,11
                               IF (ITSTFLG(RCONST(I)).GT.2) THEN
                                       ICHECK=ICHECK+1
                               END IF
                        END DO
```

(continues on next page)

(continued from previous page)

```
C                         ICHECK will be more than one if any of the constants
C                         are missing.
                         IF (ICHECK.EQ.1) THEN
C                                 IFLAG=ITSTFLG(IRAW(IS, 72))
                                IFLAG=0
                         ELSE
                                IFLAG=3
                         END IF

C                         If the flag of the raw data is less than three, then
C                               convert the raw data into derived data. This is done
C                         using a polynomial fit.
                                       IF (IFLAG.LT.3) THEN
                                   RAW=FLOAT(IRAW(IS,72).AND.'FFF'X)
                                       RDER(IS, 666)=RCONST(6)

                                DO INC=7,11
                                        RDER(IS, 666)=RDER(IS, 666)+
     1                                                    RCONST(INC)*
     2                                               (RAW**(INC-6))
                                       END DO
                             ELSE

C                                 If the flag is three or above, set the
C                                 derived data to -9999.0.
                                RDER(IS, 666)=-9999.0
                           END IF

C                         If the derived data is outside the bounds but not
C                         -9999.0, then set the flag to two.
                             IF ( ( (RDER(IS, 666).LT.323.0).OR.
     1                                     (RDER(IS, 666).GT.388.0) ).AND.
     2                                     (RDER(IS, 666).GT.-9000.0)) THEN
                                 CALL ISETFLG(RDER(IS, 666), 2)
                         ELSE

C                                 The derived data is within the limits then
C                                 set the flag to that of the raw data. If the
C                                 data is -9999.0 the flag will be three.
                                 CALL ISETFLG(RDER(IS, 666), IFLAG)
                           ENDIF

                 END DO
         ELSE

C                 The data has not got the right frequency.
                 RDER(1, 666)=-9999.0
                 CALL ISETFLG(RDER(1,666),3)
         ENDIF

c        Calibrate the ambient temp - DRS parameter 73, sample rate 1 Hz. This
c        is to be turned into Kelvin. This uses the elements of RCONST from 12
C        to 16
         IF (IFRQ(73).EQ.1) THEN  ! check the frequency.
                 DO IS=1, IFRQ(73)  ! Do for each sample.
```

(continues on next page)

(continued from previous page)

```
C                        See if all the const are there, if not set the flag to 3
                               ICHECK=1
                        DO I=12,16
                                IF (ITSTFLG(RCONST(I)).GT.2) THEN
                                        ICHECK=ICHECK+1
                                END IF
                        END DO

C                        ICHECK will be more than one if any of the constants
C                        are missing.
                        IF (ICHECK.EQ.1) THEN
C                                IFLAG=ITSTFLG(IRAW(IS, 73))
                                IFLAG=0
                        ELSE
                                IFLAG=3
                        END IF

C                        If the flag of the raw data is less than three, then
C                        convert the raw data into derived data. This is done
C                        using a polynomial fit.
                                      IF (IFLAG.LT.3) THEN
                                RAW=FLOAT(IRAW(IS,73).AND.'FFF'X)
                                     RDER(IS, 667)=RCONST(12)

                                DO INC=13,16
                                        RDER(IS, 667)=RDER(IS, 667)+
     1                                                    RCONST(INC)*
     2                                                    (RAW**(INC-12))
                                         END DO
                             ELSE

C                                If the flag is three or above, set the
C                                derived data to -9999.0.
                                RDER(IS, 667)=-9999.0

                        END IF

C                        If the derived data is outside the bounds but not
C                        -9999.0, then set the flag to two.
                             IF (((RDER(IS, 667).LT.289.0).OR.
     1                                     (RDER(IS, 667).GT.343.0)).AND.
     2                                     (RDER(IS, 667).GT.-9000.0)) THEN
                                    CALL ISETFLG(RDER(IS, 667), 2)
                              ELSE

C                                The derived data is within the limits then
C                                set the flag to that of the raw data. If the
C                                data is -9999.0 the flag will be three.
                                CALL ISETFLG(RDER(IS, 667), IFLAG)
                         ENDIF

                END DO
        ELSE

C                The data has not got the right frequency.
                RDER(1, 667)=-9999.0
                CALL ISETFLG(RDER(1,667),3)
```

(continues on next page)

(continued from previous page)

```
        ENDIF

C        Calibrate the source temp - DRS parameter 74, sample rate 1 Hz. This
C        will be in Kelvin.This uses the elements of RCONST from 17 to 22.
        IF (IFRQ(74).EQ.1) THEN  ! check the frequency.
                DO IS=1, IFRQ(74)  ! Do for each sample.

C                        See if all the const are there, if not set the flag to 3
                        ICHECK=1
                        DO I=17,22
                                IF (ITSTFLG(RCONST(I)).GT.2) THEN
                                        ICHECK=ICHECK+1
                                END IF
                        END DO

C                        ICHECK will be more than one if any of the constants
C                        are missing.
                        IF (ICHECK.EQ.1) THEN
C                                IFLAG=ITSTFLG(IRAW(IS, 74))
                                IFLAG=0
                        ELSE
                                IFLAG=3
                        END IF

C                        If the flag of the raw data is less than three, then
C                        convert the raw data into derived data. This is done
C                        using a polynomial fit.
                                     IF (IFLAG.LT.3) THEN
                                RAW=FLOAT(IRAW(IS,74).AND.'FFF'X)
                                  RDER(IS, 668)=RCONST(17)

                                DO INC=18,22
                                        RDER(IS, 668)=RDER(IS, 668)+
     1                                                          RCONST(INC)*
     2                                                     (RAW**(INC-17))
                                         END DO
                             ELSE

C                                If the flag is three or above, set the
C                                derived data to -9999.0.
                                RDER(IS, 668)=-9999.0
                              END IF

C                        If the derived data is outside the bounds but not
C                        -9999.0, then set the flag to two.
                              IF (((RDER(IS, 668).LT.378.0).OR.
     1                                      (RDER(IS, 668).GT.393.0)).AND.
     2                                      (RDER(IS, 668).GT.-9000.0)) THEN
                                     CALL ISETFLG(RDER(IS, 668), 2)
                               ELSE

C                                The derived data is within the limits then
C                                set the flag to that of the raw data. If the
C                                data is -9999.0 the flag will be three.
                                CALL ISETFLG(RDER(IS, 668), IFLAG)
                          ENDIF
```

(continues on next page)

(continued from previous page)

```
                 END DO
        ELSE

C                The data has not got the right frequency.
                RDER(1, 668)=-9999.0
                CALL ISETFLG(RDER(1,668),3)
        ENDIF

C        Calibrate the evaporator current 1- DRS parameter 75, sample rate 1 Hz
C        If it is a modified probe, ie there are four constants in the flight
C        constants file for the CALHTR1 keyword, then parameter 173 is also used.
C        This will be in amps. This uses the elements of RCONST from 23 to 26.
        IF (IFRQ(75).EQ.1) THEN  ! check the frequency.
                DO IS=1, IFRQ(75)  ! Do for each sample.

C                        See if all the const are there, if not set the flag to 3
                          ICHECK=1
                        DO I=23,26
                                IF (ITSTFLG(RCONST(I)).GT.2) THEN
                                        ICHECK=ICHECK+1
                                END IF
                        END DO

C                        ICHECK will be more than one if any of the constants
C                        are missing.
                        IF (ICHECK.EQ.1.OR.ICHECK.EQ.3) THEN
C                                IFLAG=ITSTFLG(IRAW(IS, 75))
                                IFLAG=0
                        ELSE
                                IFLAG=3
                        END IF

C                        If the flag of the raw data is less than three, then
C                        convert the raw data into derived data. This is done
C                        using a polynomial fit.
                                      IF (IFLAG.LT.3) THEN
                                RAW=FLOAT(IRAW(IS,75).AND.'FFF'X)
                                IF(ICHECK.EQ.1) THEN !It is a modified probe
                                  RAW2=FLOAT(IRAW(IS,173).AND.'FFF'X)
                                  RDER(IS, 669)=RCONST(23)+(RCONST(24)*RAW2)
     1                                      +RCONST(25)*(RAW+RCONST(26))
                                ELSE                 !It is an unmodified probe
                                  RDER(IS, 669)=RCONST(23)+RCONST(24)*RAW
                                END IF
                             ELSE

C                                If the flag is three or above, set the
C                                derived data to -9999.0.
                                RDER(IS, 669)=-9999.0
                         END IF

C                        If the derived data is outside the bounds but not
C                        -9999.0, then set the flag to two.
                              IF (((RDER(IS, 669).LT.0.3).OR.
     1                                       (RDER(IS, 669).GT.6.6)).AND.
     2                                       (RDER(IS, 669).GT.-9000.0)) THEN
                                  CALL ISETFLG(RDER(IS, 669), 2)
```

(continues on next page)

(continued from previous page)

```
                          ELSE

C                                The derived data is within the limits then
C                                set the flag to that of the raw data. If the
C                                data is -9999.0 the flag will be three.
                                 CALL ISETFLG(RDER(IS, 669), IFLAG)
                         ENDIF

                END DO
        ELSE

C                The data has not got the right frequency.
                RDER(1, 669)=-9999.0
                CALL ISETFLG(RDER(1,669),3)
        ENDIF

C        Calibrate the evaporator current 2- DRS parameter 76, sample rate 1Hz.
C        If it is a modified probe, ie there are four constants in the flight
C        constants file for the CALHTR2 keyword, then parameter 174 is also used.
C        This will be in amps. This uses the elements of RCONST from 27 to 30.
        IF (IFRQ(76).EQ.1) THEN  ! check the frequency.
                DO IS=1, IFRQ(76)  ! Do for each sample.

C                        See if all the const are there, if not set the flag to 3
                              ICHECK=1
                        DO I=27,30
                                IF (ITSTFLG(RCONST(I)).GT.2) THEN
                                        ICHECK=ICHECK+1
                                END IF
                        END DO

C                        ICHECK will be more than one if any of the constants
C                        are missing.
                        IF (ICHECK.EQ.1.OR.ICHECK.EQ.3) THEN
C                                IFLAG=ITSTFLG(IRAW(IS, 76))
                                IFLAG=0
                        ELSE
                                IFLAG=3
                        END IF

C                        If the flag of the raw data is less than three, then
C                        convert the raw data into derived data. This is done
C                        using a polynomial fit.
                                     IF (IFLAG.LT.3) THEN
                                RAW=FLOAT(IRAW(IS,76).AND.'FFF'X)
                                IF(ICHECK.EQ.1) THEN !It is a modified probe
                                  RAW2=FLOAT(IRAW(IS,174).AND.'FFF'X)
                                  RDER(IS, 670)=RCONST(27)+(RCONST(28)*RAW2)
     1                                      +RCONST(29)*(RAW+RCONST(30))
                                ELSE                 !It is an unmodified probe
                                  RDER(IS, 670)=RCONST(27)+RCONST(28)*RAW
                                END IF
                             ELSE

C                                If the flag is three or above, set the
C                                derived data to -9999.0.
                                RDER(IS, 670)=-9999.0
```

(continues on next page)

(continued from previous page)

```
                                END IF

C                        If the derived data is outside the bounds but not
C                        -9999.0, then set the flag to two.
                        IF (((RDER(IS, 670).LT.0.3).OR.
     1                                    (RDER(IS, 670).GT.6.6)).AND.
     2                                    (RDER(IS, 670).GT.-9000.0)) THEN
                                CALL ISETFLG(RDER(IS, 670), 2)
                            ELSE

C                                The derived data is within the limits then
C                                set the flag to that of the raw data. If the
C                                data is -9999.0 the flag will be three.
                                 CALL ISETFLG(RDER(IS, 670), IFLAG)
                           ENDIF

                END DO
        ELSE

C                   The data has not got the right frequency.
                RDER(1, 670)=-9999.0
                CALL ISETFLG(RDER(1,670),3)
        ENDIF

C        Calibrate the source current - DRS parameter 77, sample rate 1 Hz.
C        This will be in amps. This uses the elements of RCONST from 31 to 32.
        IF (IFRQ(77).EQ.1) THEN  ! check the frequency.
                DO IS=1, IFRQ(77)  ! Do for each sample.

C                        See if all the const are there, if not set the flag to 3
                        ICHECK=1
                        DO I=31,32
                                IF (ITSTFLG(RCONST(I)).GT.2) THEN
                                        ICHECK=ICHECK+1
                                END IF
                        END DO

C                        ICHECK will be more than one if any of the constants
C                        are missing.
                        IF (ICHECK.EQ.1) THEN
C                                IFLAG=ITSTFLG(IRAW(IS, 77))
                                IFLAG=0
                        ELSE
                                IFLAG=3
                        END IF

C                        If the flag of the raw data is less than three, then
C                        convert the raw data into derived data. This is done
C                        using a polynomial fit.
                               IF (IFLAG.LT.3) THEN
                                RAW=FLOAT(IRAW(IS,77).AND.'FFF'X)
                                      RDER(IS, 671)=(RCONST(31)+RCONST(32)*
     1                                                 RAW)
                             ELSE

C                        If the flag is three or above, set the derived data
C                        to -9999.0.
```

(continues on next page)

(continued from previous page)

```
                              RDER(IS, 671)=-9999.0
                        END IF

C                      If the derived data is outside the bounds but not
C                      -9999.0, then set the flag to two.
                         IF (((RDER(IS, 671).GT.-0.4E-03).OR.
     1                                    (RDER(IS, 671).LT.-1.1E-03)).AND.
     2                                   (RDER(IS, 671).GT.-9000.0)) THEN
                              CALL ISETFLG(RDER(IS, 671), 2)
                        ELSE

C                              The derived data is within the limits then
C                              set the flag to that of the raw data. If the
C                              data is -9999.0 the flag will be three.
                               CALL ISETFLG(RDER(IS, 671), IFLAG)
                             ENDIF

                END DO
        ELSE

C                The data has not got the right frequency.
                RDER(1, 671)=-9999.0
                CALL ISETFLG(RDER(1,671),3)
        ENDIF

C        Calibrate the status word - DRS parameter 78, sample rate 1 Hz. This
C        will be in raw data.
        IF (IFRQ(78).EQ.1) THEN ! check the frequency.
                DO IS=1, IFRQ(78) ! Do for each sample.

C                      If the raw data is inside the bounds, process it.
                     IF (((IRAW(IS, 78).AND.'FFF'X).GT.0).OR.
     1                                   ((IRAW(IS,78).AND.'FFF'X).LT.4094))
     2                                                          THEN
C                              IFLAG=ITSTFLG(IRAW(IS, 78))
                              IFLAG=0

C                              If the flag of the raw data is less than
C                              three, then convert the raw data into derived
C                              data.
                               IF (IFLAG.LT.3) THEN
                                      RDER(IS, 672)=FLOAT(IRAW(IS,78)
     1                                                    .AND.'FFF'X)
                                   ELSE

C                                        If the flag is three or above, set the
C                                      derived data to -9999.0.
                                      RDER(IS, 672)=-9999.0
                                  END IF
                          ENDIF

C                      If the derived data is outside the bounds of 0 and
C                      4095, set the flag to two.
                             IF (((RDER(IS, 672).LT.0.0).OR.
     1                                    (RDER(IS, 672).GT.4095)).AND.
     2                                   (RDER(IS, 672).GT.-9000.0)) THEN
                                CALL ISETFLG(RDER(IS, 672), 2)
```

(continues on next page)

(continued from previous page)

```
                           ELSE

C                                 If the derived data is within the bounds of 0
C                                 and 4095, then set the flag to that of the raw
C                                 data's.
                                CALL ISETFLG(RDER(IS, 672), IFLAG)
                          END IF

                    END DO
        ELSE

C                 If the wrong frequency is there for the status, then set all
C                the samples for this second to -9999.0, with their flags set
C                to 3.
                RDER(1, 672)=-9999.0
                CALL ISETFLG(RDER(1,672),3)
        ENDIF

C        Check the rate of change for parametrs 665 to 671
        TIME=ABS(RDER(1, 515)-OLD_TIME)

C        If the time has been incremented by more than one, store the
C        parameters, and return.
        IF ((TIME.gt.1.1).or.(ITSTFLG(RDER(1,515)).GT.2)) THEN
                DO INC=665, 671
                        OLD_PARAS(INC-664)=RDER(1, INC)
                END DO

                OLD_TIME=RDER(1, 515)
                RETURN
        END IF

        DO INC=665, 671
C                Only bother with a parameter that is inside its bounds.
                IF (ITSTFLG(RDER(1,INC)).LT.2) THEN
                        CHANGE=OLD_PARAS(INC-664)-RDER(1, INC)

C                        Check the differnce of the old and new value against
C                        the stored value in the array RATE_CHANGE.
                        IF (ABS(CHANGE).GT.RATE_CHANGE(INC-664)) THEN
                                CALL ISETFLG(RDER(1, INC), 2)
                            END IF
                END IF
        END DO

C        Store away the parameters
        DO INC=665, 671
                OLD_PARAS(INC-664)=RDER(1, INC)
        END DO

C        Store away the time.
        OLD_TIME=RDER(1, 515)

        RETURN

        END
```

8.28 c_winds.for

```
C
C ROUTINE          C_WINDS SUBROUTINE FORTVAX
C
C PURPOSE          Computes raw winds from TAS, vanes, and INS data
C
C DESCRIPTION      Computes values of the three wind components, using true
C                  airspeed, angle of attack and sideslip, and INS velocity,
C                  attitude, and attitude rate information. Note that at this
C                  stage the INS data have not been corrected for drift, so
C                  these are 'raw' winds, which will normally be corrected
C                  later as part of the interactive renavigation processing.
C                  Once errors have been evaluated for the three INS velocity
C                  components, they can be applied directly to the three wind
C                  components; the wind components do not need to be recomputed
C                  from scratch.  To show that the winds are 'raw' all values
C                  of U, V and W are increased by 1000 m/s by this routine.
C                  This makes it easy to see that normal (flagged 0 or 1) data
C                  are 'raw', but it may not be enough to say unabiguously
C                  whether data that are already bad (flagged 2 or 3) are 'raw'
C                  or 'corrected'.
C
C                  The processing will handle the case that the INS is mounted
C                  off the boom axis, provided its position is specified in
C                  the flight constants file, using the INSPOSN keyword.  If
C                  the INS position is not specified then it is assumed to be
C                  in the nose bay, 7.06m behind the vanes, but on the axis of
C                  the boom.  All data is assumed to be at 32 Hz.
C
C                  This routine will not be called if there is no True
C                  Airspeed, or no INS information (with the exception of roll
C                  rate).  If there is no information from the angle of attack
C                  and sideslip vanes, winds will be computed using values of
C                  zero for these angles flagged with
C                  1's.  If there is no roll rate available (this wasn't
C                  recorded for the Ferranti 1012 INS), a value of 0 is used.
C                  This doesn't matter if the INS is located on the boom axis,
C                  since in this case roll rate has no effect on winds.
C
C                  The output vertical wind takes the worst flag present on the
C                  AOA, VZ, TAS and pitch data.  The output horizontal wind
C                  components take the worst flag present on the AOSS, VN, VE,
C                  TAS, and heading data.  This is suitable when the
C                  aircraft is not banking and reflects the fact that good
C                  horizontal winds can be found even when the vertical
C                  velocity is bad.  However this flagging scheme fails to
C                  reflect coupling between the vertical and horizontal
C                  measurement when the aircraft is banking.
C                  In addition horizontal wind components greater
C                  than 100 m/s and vertical components greater than 25 m/s
C                  are flagged with 2's, and if the change between adjacent
C                  samples (at 32 Hz) is greater than 1 m/s a flag of 2 is
C                  also applied.
C
C                  Input parameters (all at 32 Hz except 515):
C
```

(continues on next page)

(continued from previous page)

```
C                  Para 515   Time, secs
C                  Para 779   Turb.probe dry true airspeed, m s-1
C                  Para 548   Angle of attack, deg
C                  Para 549   Angle of side slip, deg
C                  Para 558   INS velocity north, m s-1
C                  Para 559   INS velocity east, m s-1
C                  Para 557   INS vertical velocity, m s-1
C                  Para 560   INS roll, deg
C                  Para 561   INS pitch, deg
C                  Para 562   INS heading, deg
C                  Para 567   INS roll rate, deg s-1 (optional)
C                  Para 565   INS pitch rate, deg s-1
C                  Para 566   INS yaw rate, deg s-1
C
C                  Constants:
C
C                  RCONST(1)  Distance of vanes ahead of INS, m (optional)
C                  RCONST(2)  Distance of vanes to port of INS, m (optional)
C                  RCONST(3)  Distance of vanes above INS, m (optional)
C
C                  Output parameters (all at 32 Hz):
C
C                  Para 714   Northward wind component + 1000, m s-1
C                  Para 715   Eastward wind component + 1000, m s-1
C                  Para 716   Vertical wind component + 1000, m s-1
C
C VERSION          1.00  10-5-93  W.D.N.JACKSON
C
C ARGUMENTS        IRAW(64,512) I*4 IN  Up to 64 samples for up to 512 DRS pars
C                  IFRQ(512)    I*4 IN  Sample rate of each DRS par (0-64)
C                  RCONST(64)   R*4 IN  Inputs constants
C                  RDER(64,1024)R*4 OUT Output array of up to 64 samples for
C                                       each of 1024 parameters
C
C CHANGES          1.01  20-04-98 W.D.N.JACKSON
C                  Error in computation of airspeed corrected.
C                  1.02  14-06-2004 Phil Brown
C                  AoA and AoSS now compulsory input parameters to ensure
C                  this routine gets called after C_TURB
C                  1.03  09/07/04 Phil Brown
C                  Input TAS parameter is now 779 (Turb.probe dry TAS)
C                  1.04  25/08/04 Phil Brown
C                  Temporary. Suspend rate-of-change checking on winds.
C                  1.05  29/11/04 Phil Brown
C                  Temporary. Check flagging of RU,RV,RW when returned to try
C                  to suppress FLTINV errors.
C
********************************************************************************
      SUBROUTINE C_WINDS(IRAW,IFRQ,RCONST,RDER)
CDEC$ IDENT 'V1.04'
      INTEGER*4 IRAW(64,512)      !Raw data array
      INTEGER*4 IFRQ(512)         !Raw data frequency
      REAL*4    RCONST(64)        !Constants array
      REAL*4    RDER(64,1024)     !Derived data array
C
C This routine uses the following parameters (note that the absence of AOA,
C AOSS or roll rate will not stop C_WINDS from being called).  All parameters,
```

(continues on next page)

(continued from previous page)

```
C except time, are at 32 Hz:
C
      PARAMETER GMT=515                 !Time, secs
      PARAMETER TAS=779                 !True airspeed, m s-1
      PARAMETER AOA=548                 !Angle of attack, deg
      PARAMETER AOS=549                 !Angle of side slip, deg
      PARAMETER VN=558                  !INS velocity north, m s-1
      PARAMETER VE=559                  !INS velocity east, m s-1
      PARAMETER VZ=557                  !INS vertical velocity, m s-1
      PARAMETER ROL=560                 !INS roll, deg
      PARAMETER PIT=561                 !INS pitch, deg
      PARAMETER HDG=562                 !INS heading, deg
      PARAMETER ROLR=567                !INS roll rate, deg s-1 (optional)
      PARAMETER PITR=565                !INS pitch rate, deg s-1
      PARAMETER YAWR=566                !INS yaw rate, deg s-1
C
C This routine takes three constants from the RCONST array.  They are
C all optional and if not specified will be defaulted to the position of the
C H423 INU on the 146 Core Console (16.002,-0.8128,-0.4390 m).
C
      PARAMETER PL=1                    !Const dist of vanes ahead of INS
      PARAMETER PM=2                    !Const dist of vanes to port of INS
      PARAMETER PN=3                    !Const dist of vanes above INS
C
C This routine computes the following parameters, all at 32 Hz:
C Note that TARDIS conventially labels parameter 714, Northerly component, as V
C and parameter 715, Easterly component, as U.
C
      PARAMETER U=714                   !Northward wind component, m s-1
      PARAMETER V=715                   !Eastward wind component, m s-1
      PARAMETER W=716                   !Vertical wind component, m s-1
C
C Set LFLAG to false if you want to treat all data as unflagged.
C
      DATA LFLAG   /.TRUE./            !Set false if want to ignore flagging

      DATA RLSTSEC /-2.0/               !Initial dummy value for last sec processed

      RDEFAOA=0.0                       !If not specified AOA is 0.0 flagged 1
      CALL ISETFLG(RDEFAOA,1)
      RDEFAOS=RDEFAOA                   !If not specified AOSS is 0.0 flagged 1

      IF(.NOT.LFLAG) THEN               !Ignore flagging
        DO I=1,32                       !For each sample in second
          CALL C_WINDS_UVW(RDER(I,TAS),RDER(I,AOA),RDER(I,AOS),
     -        RDER(I,VN),RDER(I,VE),RDER(I,VZ),
     -        RDER(I,HDG),RDER(I,PIT),RDER(I,ROL),
     -        RCONST(PL),RCONST(PM),RCONST(PN),
     -        RDER(I,YAWR),RDER(I,PITR),RDER(I,ROLR),
     -        RDER(I,U),RDER(I,V),RDER(I,W))
        END DO
      ELSE                              !Apply flags
        RL=RCONST(PL)                   !Get the INS position offsets
        RM=RCONST(PM)
        RN=RCONST(PN)
        IF(ITSTFLG(RL).GE.2) RL=16.002 !Use default values if not available
        IF(ITSTFLG(RM).GE.2) RM=-0.8128
```

(continues on next page)

(continued from previous page)

```
        IF(ITSTFLG(RN).GE.2) RN=-0.4390
        LCONSEQ=.FALSE.               !Will set true if this is next second
        IF(RDER(1,GMT).EQ.RLSTSEC+1.0) LCONSEQ=.TRUE.
        RLSTSEC=RDER(1,GMT)           !Save current time
        DO I=1,32                     !For each sample in second
          RTAS=RDER(I,TAS)            !Get the input values
          RAOA=RDER(I,AOA)
          RAOS=RDER(I,AOS)
          RVN=RDER(I,VN)
          RVE=RDER(I,VE)
          RVZ=RDER(I,VZ)
          RHDG=RDER(I,HDG)
          RPIT=RDER(I,PIT)
          RROL=RDER(I,ROL)
          RYAWR=RDER(I,YAWR)
          RPITR=RDER(I,PITR)
          RROLR=RDER(I,ROLR)
          IF(ITSTFLG(RAOA).GE.2) RAOA=RDEFAOA !Set AOA to 0 if missing
          IF(ITSTFLG(RAOS).GE.2) RAOS=RDEFAOS !Set AOSS to 0 if missing
          IF(ITSTFLG(RROLR).GE.2) RROLR=0.0   !Set roll rate to 0 if missing
          IHFLAG=MAX(ITSTFLG(RTAS),ITSTFLG(RAOS), !Compute worst horiz flag
     -        ITSTFLG(RVN),ITSTFLG(RVE),ITSTFLG(RHDG))
          IWFLAG=MAX(ITSTFLG(RTAS),ITSTFLG(RAOA), !Compute worst vert flag
     -        ITSTFLG(RVZ),ITSTFLG(RPIT))
          CALL ISETFLG(RTAS,0)         !Clear any flags before computation
          CALL ISETFLG(RAOA,0)
          CALL ISETFLG(RAOS,0)
          CALL ISETFLG(RVN,0)
          CALL ISETFLG(RVE,0)
          CALL ISETFLG(RVZ,0)
          CALL ISETFLG(RHDG,0)
          CALL ISETFLG(RPIT,0)
          CALL ISETFLG(RROL,0)
          CALL ISETFLG(RYAWR,0)
          CALL ISETFLG(RPITR,0)
          CALL ISETFLG(RROLR,0)
          CALL C_WINDS_UVW(RTAS,RAOA,RAOS,RVN,RVE,RVZ,RHDG,RPIT,RROL,
     -        RL,RM,RN,RYAWR,RPITR,RROLR,RU,RV,RW) !Compute wind components
          IUFLAG=IHFLAG                !Propagate worst case flag for each comp
          IVFLAG=IHFLAG
          IF(ABS(RU).GT.100.0) IUFLAG=MAX(IUFLAG,2) !Flag if out of range
          IF(ABS(RV).GT.100.0) IVFLAG=MAX(IVFLAG,2)
          IF(ABS(RW).GT.25.0) IWFLAG=MAX(IWFLAG,2)
          CALL ISETFLG(RU, 0)          ! ensure winds have zero flag
          CALL ISETFLG(RV, 0)
          CALL ISETFLG(RW, 0)
          RU=RU+1000.                  !Add offset to show winds are 'raw'
          RV=RV+1000.
          RW=RW+1000.

C suspend rate-of-change checks.
C          IF(ITSTFLG(RLSTU).EQ.0.AND.LCONSEQ.AND.ABS(RLSTU-RU).GT.1.0)
C     -        IUFLAG=MAX(IUFLAG,2)     !Flag if rate of change too high
C          IF(ITSTFLG(RLSTV).EQ.0.AND.LCONSEQ.AND.ABS(RLSTV-RV).GT.1.0)
C     -        IVFLAG=MAX(IVFLAG,2)
C          IF(ITSTFLG(RLSTW).EQ.0.AND.LCONSEQ.AND.ABS(RLSTW-RW).GT.1.0)
C     -        IWFLAG=MAX(IWFLAG,2)
```

(continues on next page)

(continued from previous page)

```
          CALL ISETFLG(RU,IUFLAG)       !Apply flags to result
          CALL ISETFLG(RV,IVFLAG)
          CALL ISETFLG(RW,IWFLAG)
          RDER(I,U)=RU                  !Transfer results to output array
          RDER(I,V)=RV
          RDER(I,W)=RW
          RLSTU=RU                      !Save latest values
          RLSTV=RV
          RLSTW=RW
          LCONSEQ=.TRUE.                !Further samples in second are consequetve
        END DO
      END IF
      RETURN
      END
********************************************************************************
C
C SUBROUTINE C_WINDS_UVW
C
C Computes the three wind components, using INS velocities, attitudes,
C attitude rates, vanes location, true airspeed, and angles of attack and
C sideslip.  All data are treated as unflagged real by 4 numbers.
C
C Arguments (all R*4) - no input arguments are changed:
C
C  RTAS   In  True airspeed (m/s)
C  RAOA   In  Angle of attack (deg, +ve when vane points down)
C  RAOS   In  Angle of attack (deg, +ve when vane points to left)
C  RVN    In  INS aircraft velocity component northwards (m/s, +ve when to N)
C  RVE    In  INS aircraft velocity component eastwards (m/s, +ve when to E)
C  RVZ    In  INS aircraft verical velocity component (m/s, +ve when up)
C  RHDG   In  INS aircraft heading (deg, +ve when left wing forward)
C  RPIT   In  INS aircraft pitch (deg, +ve when nose is up)
C  RROL   In  INS aircraft roll (deg, +ve left wing up)
C  RL     In  Distance of vanes/nose from INS (m, +ve when nose ahead)
C  RM     In  Distance of the vanes/nose from the INS (m, +ve when nose to port)
C  RN     In  Distance of the vanes/nose from the INS (m, +ve when nose above)
C  RYAWR  In  Yaw rate (deg/s, +ve when left wind moving ahead)
C  RPITR  In  Pitch rate (deg/s, +ve when nose moving up)
C  RROLR  In  Roll rate (deg/s, +ve when left wing moving up)
C  RU     Out Wind component Northwards (m/s)
C  RV     Out Wind component Eastwards (m/s)
C  RW     Out Wind component Upwards (m/s)
C
C Derives winds using the following matrix formulation of the wind equations:
C
C (U)               (-U'      ) (VN) [( 0 )    (0  )         (r')]            (l)
C (V) = (A3.A2).A1.(-U'tan(b))+(VW)+[( 0 )+A3.(-t')+(A3.A2).(0 )]x(A3.A2).A1.(m)
C (W)               ( U'tan(a)) (VZ) [(-p')    (0  )         (0 )]            (n)
C
C where U' = TAS / (1+ tan(b)^2 + tan(a)^2)^(1/2), b is angle of sideslip, a is
C angle of attack, p' is heading rate, t' is pitch rate, r' is roll rate, l m
C and n are the position of the vanes and nose with respect to the INS (l +ve
C forwards, m +ve to port, n +ve up), U is wind component northwards, V is wind
C component westwards, W is wind component upwards, VN is aircraft velocity
C northwards, VW is aircraft velocity westwards, VZ is aircraft velocity
Cupwards, and
```

(continues on next page)

(continued from previous page)

```
C
C    (1  0       0    )      (cos(t) 0 -sin(t))     ( cos(p) sin(p) 0)
C A1=(0 cos(r) -sin(r))  A2=( 0     1   0    )  A3=(-sin(p) cos(p) 0)
C    (0 sin(r)  cos(r))      (sin(t) 0  cos(t))     (  0      0     1)
C
C where r is roll, t is pitch, and p is heading.
C
C This is simpler to use than explicit wind component equations when the INS
C is off the aircraft axis.  Comparisons of the wind components derived by this
C subroutine with those derived by the normal wind equations, as used in
C C_INS_WINDS, show no differences greater than 2E-5 m/s.  However direct use
C of the wind equations is about 30% faster.
C
C Ref: MRF Internal Note No 8 - 'The measurement of flight level winds and
C aircraft position by the MRF Hercules' by S. Nicholls, together with
C additional notes by W.D.N.Jackson, February 1993, which extend the
C analysis to cases where the INS is off axis and derives the matrix equation
C used here.
C
C V1.00  15/03/93  W.D.N.JACKSON
C V1.01  20/04/98  W.D.N.JACKSON
C        Error in formulation of Axford/Nicholls/Jackson equations when
C        computing airspeed corrected.  (See note by R Wood and G W Inverarity)
C
      SUBROUTINE C_WINDS_UVW(RTAS,RAOA,RAOS,RVN,RVE,RVZ,RHDG,RPIT,RROL,
     -    RL,RM,RN,RYAWR,RPITR,RROLR,RU,RV,RW)
CDEC$ IDENT 'V1.01'
      REAL*4 RP(3)                    !Vanes posn wrt to INS, fore, port & up
      REAL*4 RWIND(3)                 !The 3 computed wind comps Un, Vw and Wu
      REAL*4 RVG(3)                   !INS VN, VW and VZ
      REAL*4 RA1(3,3)                 !Transformation matrix about roll axis
      REAL*4 RA2(3,3)                 !Transformation matrix about pitch axis
      REAL*4 RA3(3,3)                 !Transformation matrix about heading axis
      REAL*4 RT(3,3)                  !Full transformation matrix
      REAL*4 RTMP(3,3)                !Temporary matrix store
      REAL*4 RUA(3)                   !Airflow vector in a/c frame
      REAL*4 RYR(3)                   !Yaw rate vector
      REAL*4 RPR(3)                   !Pitch rate vector
      REAL*4 RRR(3)                   !Roll rate vector
      REAL*4 RTEMP(3)                 !Temporary vector store

      RA1(1,1)=1.                     !Define roll transformation matrix
      RA1(1,2)=0.
      RA1(1,3)=0.
      RA1(2,1)=0.
      RA1(2,2)=COSD(RROL)
      RA1(2,3)=-SIND(RROL)
      RA1(3,1)=0.
      RA1(3,2)=-RA1(2,3)
      RA1(3,3)=RA1(2,2)

      RA2(1,1)=COSD(RPIT)             !Define pitch transformation matrix
      RA2(1,2)=0.
      RA2(1,3)=-SIND(RPIT)
      RA2(2,1)=0.
      RA2(2,2)=1.
      RA2(2,3)=0.
```

(continues on next page)

(continued from previous page)

```
      RA2(3,1)=-RA2(1,3)
      RA2(3,2)=0.
      RA2(3,3)=RA2(1,1)

      RA3(1,1)=COSD(RHDG)              !Define heading transformation matrix
      RA3(1,2)=SIND(RHDG)
      RA3(1,3)=0.
      RA3(2,1)=-RA3(1,2)
      RA3(2,2)=RA3(1,1)
      RA3(2,3)=0.
      RA3(3,1)=0.
      RA3(3,2)=0.
      RA3(3,3)=1.

!       PRINT *,'RTAS/AOA/AOSS =',RTAS,RAOA,RAOS

      TANAOS=TAND(RAOS)                !Define airspeed vector
      TANAOA=TAND(RAOA)
      D=SQRT(1.0+TANAOS*TANAOS+TANAOA*TANAOA)
      RUA(1)=-RTAS/D
      RUA(2)=-RTAS*TANAOS/D
      RUA(3)=RTAS*TANAOA/D

      RP(1)=RL                         !Define INS offset vector
      RP(2)=RM
      RP(3)=RN

      RVG(1)=RVN                       !Define INS velocity vector
      RVG(2)=-RVE                      !Matrix eqn requires VW
      RVG(3)=RVZ

      RYR(1)=0.                        !Define yaw rate vector
      RYR(2)=0.
      RYR(3)=-RYAWR*3.14159/180.       !Convert to rad/s

      RPR(1)=0.                        !Define pitch rate vector
      RPR(2)=-RPITR*3.14159/180.       !Convert to rad/s
      RPR(3)=0.

      RRR(1)=RROLR*3.14159/180.        !Define roll rate vector in rad/s
      RRR(2)=0.
      RRR(3)=0.

      RWIND(1)=0.                      !Clear wind vector
      RWIND(2)=0.
      RWIND(3)=0.

      CALL C_WINDS_MULM(RA3,RA2,RTMP)  !Compute full transformation vector
      CALL C_WINDS_MULM(RTMP,RA1,RT)
      CALL C_WINDS_MATV(RT,RUA,RUA)    !Transform airspeed to ground frame
      CALL C_WINDS_VADD(RUA,RWIND,RWIND) !This is first wind component
      CALL C_WINDS_VADD(RVG,RWIND,RWIND) !Add ground speed component
      CALL C_WINDS_MATV(RT,RP,RP)      !Transform INS offset to ground frame
      CALL C_WINDS_MULM(RA3,RA2,RTMP)  !Transfm roll rate effects to ground fram
      CALL C_WINDS_MATV(RTMP,RRR,RRR)  !Compute roll rate effects
      CALL C_WINDS_MATV(RA3,RPR,RTEMP) !Transfm pitch rate effects to ground frm
      CALL C_WINDS_VADD(RRR,RTEMP,RTEMP) !Add pitch rate effects
```

(continues on next page)

(continued from previous page)

```
      CALL C_WINDS_VADD(RYR,RTEMP,RTEMP) !Add yaw rate effects to get full effect
      CALL C_WINDS_VMUL(RTEMP,RP,RTEMP) !Apply rate effects to INS offset vector
      CALL C_WINDS_VADD(RTEMP,RWIND,RWIND) !This is the last wind component

      RU=RWIND(1)                       !Transfer result to output arguments
      RV=-RWIND(2)                      !Convert westwards to eastwards
      RW=RWIND(3)

      RETURN
      END
*******************************************************************************
      SUBROUTINE C_WINDS_MULM(A,B,C)
C
C Applies the 3x3 matrix A to the 3x3 matrix B, and leaves the result in C
C which may be the same as A or B.
C
C V1.00  15/03/93  W.D.N.JACKSON
C
CDEC$ IDENT 'V1.00'
      REAL*4 A(3,3),B(3,3),C(3,3),T(3,3)
      T(1,1)=A(1,1)*B(1,1)+A(1,2)*B(2,1)+A(1,3)*B(3,1)
      T(1,2)=A(1,1)*B(1,2)+A(1,2)*B(2,2)+A(1,3)*B(3,2)
      T(1,3)=A(1,1)*B(1,3)+A(1,2)*B(2,3)+A(1,3)*B(3,3)
      T(2,1)=A(2,1)*B(1,1)+A(2,2)*B(2,1)+A(2,3)*B(3,1)
      T(2,2)=A(2,1)*B(1,2)+A(2,2)*B(2,2)+A(2,3)*B(3,2)
      T(2,3)=A(2,1)*B(1,3)+A(2,2)*B(2,3)+A(2,3)*B(3,3)
      T(3,1)=A(3,1)*B(1,1)+A(3,2)*B(2,1)+A(3,3)*B(3,1)
      T(3,2)=A(3,1)*B(1,2)+A(3,2)*B(2,2)+A(3,3)*B(3,2)
      T(3,3)=A(3,1)*B(1,3)+A(3,2)*B(2,3)+A(3,3)*B(3,3)
      DO I=1,3
        DO J=1,3
          C(I,J)=T(I,J)
        END DO
      END DO
      RETURN
      END
*******************************************************************************
      SUBROUTINE C_WINDS_MATV(A,B,C)
C
C Applies the 3x3 matrix A to the column vector B, and returns with the result
C in C, which may be the same as B.
C
C V1.00  15/03/93  W.D.N.JACKSON
C
CDEC$ IDENT 'V1.00'
      REAL*4 A(3,3),B(3),C(3),T(3)
      T(1)=A(1,1)*B(1)+A(1,2)*B(2)+A(1,3)*B(3)
      T(2)=A(2,1)*B(1)+A(2,2)*B(2)+A(2,3)*B(3)
      T(3)=A(3,1)*B(1)+A(3,2)*B(2)+A(3,3)*B(3)
      C(1)=T(1)
      C(2)=T(2)
      C(3)=T(3)
      RETURN
      END
*******************************************************************************
      SUBROUTINE C_WINDS_VADD(A,B,C)
C
```

(continues on next page)

(continued from previous page)

```
C Adds the 3 element column vector A to column vector B and returns the result
C in C, which can be the same as A or B.
C
C V1.00  15/03/93  W.D.N.JACKSON
C
CDEC$ IDENT 'V1.00'
      REAL*4 A(3),B(3),C(3)
      C(1)=A(1)+B(1)
      C(2)=A(2)+B(2)
      C(3)=A(3)+B(3)
      RETURN
      END
********************************************************************************
      SUBROUTINE C_WINDS_VMUL(A,B,C)
C
C Multiplies the 3 element column vector A with the column vector B and returns
C the result in C, which can be the same as A or B.
C
C V1.00  15/03/93  W.D.N.JACKSON
C
CDEC$ IDENT 'V1.00'
      REAL*4 A(3),B(3),C(3),T(3)
      T(1)=A(2)*B(3)-A(3)*B(2)
      T(2)=A(3)*B(1)-A(1)*B(3)
      T(3)=A(1)*B(2)-A(2)*B(1)
      C(1)=T(1)
      C(2)=T(2)
      C(3)=T(3)
      RETURN
      END
```

8.29 g_mach.for

```
C--------------------------------------------------------------------------------
C
C ROUTINE              G_MACH      SUBROUTINE FORTVAX                   [G_MACH.FOR]
C
C PURPOSE             COMPUTE MACH  NO. FROM STATIC PRESSURE AND PITOT STATIC.
C
C DESCRIPTION      The two input arguments are Static and Pitot static pressure.
C                  The value of Static pressure is checked to make sure it is
C                  not zero, to avoid a 'divide-by-zero' error. The division of
C                   Pitot staic pressure by Static pressure is check to make
C                   sure that it is not negative, is so the Mach number is set
C                   to 0.0.
C                  Computation of the Mach no. 'RMACH' proceeds using the
C                  formula below.
C
C                  RMACH = SQRT( 5.* ((1.+ PITOT/RSTAT)**(2./7.) - 1.))
C
C VERSION           1.00    26-02-92        M.J.Glover
C
C ARGUMENTS        RSTAT - R*4 IN    Static pressure       (100 - 1050 mb.)
C                  PITOT - R*4 IN    Pitot static pressure (0   - 125  mb.)
```

(continues on next page)

(continued from previous page)

```
C                  RMACH - R*4 OUT   MACH NO.                 [none ]
C
C SUBPROGRAMS             None.
C
C REFERENCES            Code adapted from SCILIB:S_MACH
C
C CHANGES          None.
C
C------------------------------------------------------------------------------

        SUBROUTINE G_MACH (RSTAT,PITOT,RMACH)
CDEC$ IDENT 'V1.00'
C
        IMPLICIT NONE

        INTEGER*4  IFLAG

        REAL*4     RSTAT,PITOT,RMACH ,SRSTAT,SPITOT

C------------------------------------------------------------------------------

              SPITOT = PITOT                                !Put input arguments
              SRSTAT = RSTAT                                !into placeholders.
        IFLAG = 0

        IF (SRSTAT .EQ. 0 ) THEN                      !Divide-by-zero err?
                RMACH  = 0.                           !Zero return value
                RETURN                                !.. cannot proceed.
              ENDIF

        IF (SPITOT/SRSTAT .LT.0)IFLAG=3               !Must be +ve or eqn
                                                      !below will fail.

C If flag not fatal, compute Mach no.

          IF (IFLAG.NE.3 ) THEN
                RMACH = SQRT( 5.*  ((1.+ SPITOT/SRSTAT)**(2./7.) - 1.))
        ELSE
                RMACH = 0.0                           !Return flagged zero
        ENDIF                                         !if input is invalid

        RETURN

        END
```

8.30 isetflg.for

```
C
C ROUTINE          ISETFLG SUBROUTINE FORTVAX
C
C PURPOSE          Sets the flag bits in a variable
C
C DESCRIPTION      Flagged values are stored in bits 0 and 1 of a 32 bit
C                  word.  This routine simply sets these bits.  It will work
```

(continues on next page)

(continued from previous page)

```
C                    with either REAL*4 or INTEGER*4 values.
C
C VERSION            1.00  21-12-89  N.JACKSON
C
C ARGUMENTS          IVALUE    R*4 or I*4  IN/OUT   Variable with flag bits
C                    IFLAG     I*4         IN       Flag value (0-3)
C
C CHANGES            1.01  16-06-05  N.JACKSON
C                    Now sets uses bits 0 and 1 instead of 16 and 17 to match
C                    IEEE S_Floating mantissa.
C
      SUBROUTINE ISETFLG(IVALUE,IFLAG)
CDEC$ IDENT 'V1.01'
C
C The routine masks off bits 0 and 1 (the lowest 2 bits of IVALUE(1)) and
C then sets them with the lowest 2 bits in IFLAG.  IFLAG is masked to 2 bits
C in case an invalid number is sent.
C
      INTEGER*2 IVALUE(2),IFLAG(2)
      IVALUE(1)=(IVALUE(1).AND.'FFFC'X).OR.(IFLAG(1).AND.'0003'X)
      RETURN
      END
```

8.31 itstflg.for

```
C
C ROUTINE            ITSTFLG FUNCTION FORTVAX
C
C PURPOSE            Returns the flag value of a variable
C
C DESCRIPTION        Flagged values are stored in bits 0 and 1 of a 32 bit
C                    word.  This routine simply extracts these bits and returns
C                    their value.  It will work with either REAL*4 or INTEGER*4
C                    values.
C
C VERSION            1.00  21-12-89  N.JACKSON
C
C ARGUMENTS          IVALUE    R*4 or I*4  IN   Variable with flag bits
C                    ITSTFLG   I*4         OUT  Flag value (0-3)
C
C CHANGES            1.01  16-06-05  N.JACKSON
C                    Now uses bits 0 and 1 rather than 16 and 17 to match the
C                    IEEE S_Floating mantissa.
C
      INTEGER FUNCTION ITSTFLG(IVALUE)
CDEC$ IDENT 'V1.01'
C
      INTEGER*2 IVALUE(2)
      ITSTFLG=IVALUE(1).AND.'0003'X
      RETURN
      END
```

8.32 s_mach.for

```
C-------------------------------------------------------------------------
C
C ROUTINE             S_MACH      SUBROUTINE FORTVAX
C
C PURPOSE             COMPUTE MACH  NO. FROM STATIC PRESSURE AND PITOT STATIC.
C
C DESCRIPTION       The two input arguments Static and Pitot static pressure
C                   (normally taken from samples of parameters 576 and 577)
C                   have their flag values noted. The input arguments are
C                   ranged-checked, with an out-of-range condition giving
C                   a flag value of two. The input flag values, together
C                   with results of range-checking give a 'worst' flag value.
C                   If the worst value is <2 computation of Mach number proceeds.
C                   Otherwise the value of Mach no will be set to zero later.
C                   The value of Static pressure is also checked that it is
C                   not zero, to avoid a 'divide-by-zero' error.
C                   Computation of the Mach no. 'RMACH' proceeds using the
C                   formula below.
C                   The return value of Mach no. has its flag area set to the
C                   'worst' flag value found.
C
C                   RMACH = SQRT( 5.* ((1.+ PITOT/RSTAT)**(2./7.) - 1.))
C
C VERSION            1.00    070290   A.D.HENNINGS
C
C ARGUMENTS         RSTAT - R*4 IN    Static pressure       (100 - 1050 mb.)
C                   PITOT - R*4 IN    Pitot static pressure (0    - 125  mb.)
C                   RMACH - R*4 OUT   MACH NO.                 [none ]
C
C SUBPROGRAMS               ISETFLG, ITSTFLG
C
C REFERENCES              Code adapted from MRF1/HORACE
C
C CHANGES           V1.01  020490  Include check for divide-by-zero error
C                                 Return 0.0 flagged '3' if Static pressure
C                                 input is zero.
C----------------------------------------------------------------------------
         SUBROUTINE S_MACH (RSTAT,PITOT,RMACH)
CDEC$ IDENT 'V1.01'
C
      IMPLICIT NONE
      INTEGER*4  ITSTFLG, IFLAG, IFLAG2
      REAL*4     RSTAT,PITOT,RMACH ,SRSTAT,SPITOT
      REAL*4     STMAX,STMIN,PIMAX,PIMIN
      PARAMETER (STMAX=1050. ,STMIN= 100. ,PIMAX=125. ,PIMIN=0.) !Static,Pitot
                                                                 !range limits.
C----------------------------------------------------------------------------

      SPITOT = PITOT                                          !Put input arguments
      SRSTAT = RSTAT                                          !into placeholders.

      IFLAG  =ITSTFLG (SRSTAT)                                !Test validity of
      IF (SRSTAT .GT. STMAX .OR. SRSTAT .LT.STMIN) IFLAG=2
      IFLAG2 =ITSTFLG (SPITOT)                                !input arguments
      IF (SPITOT .GT. PIMAX .OR. SPITOT .LT.PIMIN) IFLAG2=2
```

(continues on next page)

(continued from previous page)

```
      IF (IFLAG.LT.IFLAG2 ) IFLAG=IFLAG2                   !Choose worst flag.

      IF (SRSTAT .EQ. 0    ) THEN                          !Divide-by-zero err?
        RMACH   = 0.                                       !Zero return value
        IFLAG   = 3                                        !Set flag
        CALL ISETFLG (RMACH,IFLAG)                         !Flag result invalid
        RETURN                                             !.. cannot proceed.
      ENDIF

                                                           !Exponentiation err?
      IF (SPITOT/SRSTAT .LT.0)IFLAG=3                      !Must be +ve or eqn
                                                           !below will fail.

      CALL ISETFLG(SPITOT,0)                               !Clear flag bits
      CALL ISETFLG(SRSTAT,0)                               !before computation

C  If flag not fatal, compute Mach no.

      IF (IFLAG.LT.3 ) THEN
        RMACH = SQRT( 5.* ((1.+ SPITOT/SRSTAT)**(2./7.) - 1.))
      ELSE
        RMACH = 0.0                                        !Return flagged zero
      ENDIF                                                !if input is invalid

      CALL ISETFLG (RMACH,IFLAG)                           !Re-flag result
      RETURN

      END
```

8.33 s_qcpt.for

```
C-----------------------------------------------------------------------------
C ROUTINE              S_QCPT      SUBROUTINE FORTVAX
C
C PURPOSE               PERFORM RANGE CHECK AND RATE OF CHANGE CHECK ON DATA POINT.
C
C DESCRIPTION       This routine is only valid for data which varies in a linear
C                   manner. It will not Q/C data which changes in a cyclic manner
C                   such as direction/angles going through 0/360.
C                   It quality-controls a data point with respect to range limits
C                   and check rate-of-change between it, and previous/good points
C                   Elementary checks performed to discrimimate between isolated
C                   'spikes' and a new trend departing from the previous data
C                   values.
C                   Data passing checks causes flag value: 0 to be returned
C                   Data failing checks causes flag value: 2 to be returned
C
C METHOD            1. Check for a break of time between samples (whole seconds
C                      only).
C                      If true, initialise 'last time through' variables to
C                      current values of data and time.
C                   2. Attempt to eliminate spikes; after the error count RERCNT
C                      rises above RERRMX bad points, accept next point that is
```

(continues on next page)

(continued from previous page)

```
C                     within valid range.
C                     n.b  You must initialise RERRMX > 0, suggested value: 3
C                  3. Check rate-of-change and range limits:
C                     If either fails: set the return flag to 'failed' value: 2
C                                      increment the error counter.
C                     If both valid: Accept point, set flag to 'good'  value: 0
C                                    reset error count
C                                    retain this value as 'last good point'
C                  4. Retain last-seconds time, for comparison next time through
C
C VERSION           1.00  290190   A.D.HENNINGS
C                  1.01  17-01-96 D Lauchlan
C
C ARGUMENTS        RSEC   REAL*4 IN     Current time; seconds from midnight
C                  RLASTM REAL*4 IN/OUT Previous time; seconds from midnight
C                  RVAL   REAL*4 IN     Data value; current sample
C                  RLASTV REAL*4 IN/OUT Previous 'good' data value
C                  RMAX   REAL*4 IN     Q/C Max limit before rejecting
C                  RMIN   REAL*4 IN      "  Min   "     "       "
C                  RCHG   REAL*4 IN      "  Rate-of-change between succ samples.
C                  RERRMX REAL*4 IN      "  No.of succ bad pts before reset.
C                  RERCNT REAL*4 IN/OUT  "  No.of succ bad pts found so far.
C                  IFLAG  INT*4 OUT     Return value. 0: Good  2: Failed.
C
C CHANGES          1.01 Unused variables removed.
C
C------------------------------------------------------------------------------
         SUBROUTINE S_QCPT (rsec, rlastm, rval,rlastv,
     +                     rmax, rmin, rchg, rerrmx,rercnt,iflag)
CDEC$ IDENT 'V1.01'
C
        real*4  rsec                                !Current second-from-mid.
        real*4  rlastm                              !Last time (secs) checked
        real*4  rval                                !current value
        real*4  rlastv                              !Last acceptable value
        real*4  rmax                                !Max acceptable value
        real*4  rmin                                !Min acceptable value
        real*4  rchg                                !acctble diff bet. samps
        real*4  rerrmx                              !no.bad pts before reset
        real*4  rercnt                              !No. succ. bad pts found.
        integer*4 iflag                             !Value of flag on return
C-------------------------------------------------------------------------------



C       AFTER A BREAK, OR FIRST TIME THROUGH; INITIALISE 'PREVIOUS/LAST' VALUES

        if (rsec-rlastm .gt. 1. )then
           rlastv= rval                              !Last 'good' value; this
           rlastm= rsec-1.                           !Last second processed.
        endif

C       DISCRIMINATE BETWEEN 'SPIKES' AND A NEW TREND.

        if (rercnt .gt.rerrmx)then                 !max error pts, accept next
          if (rval .le. rmax .and. rval .ge. rmin) then  !within range limits.
            rlastv= rval                               !Last good val is this
```

(continues on next page)

(continued from previous page)

```
            rercnt   = 1.                                 !reset error counter
          endif
        endif

C       CHECK NEW VALUE IS VALID FOR RATE-OF-CHANGE, AND RANGE LIMITS.

        if ( abs(rlastv-rval) .gt. rercnt*rchg .or.         !UNACCEPTABLE
     _       rval .gt. rmax .or. rval .lt. rmin) then     !------------
             rercnt = rercnt + 1.                         !Inc count of bad pts
             iflag = 2                                    !Set ret. flag invalid
        else                                            !ACCEPTABLE
                                                        !------------
             iflag = 0                                  !Set ret. flag valid
             rlastv = rval                              !only save if its good
             rercnt = 1.0                               !reset acc err marg count
        endif

C        PRESERVE VALUES FOR NEXT  TIME THROUGH.

        rlastm = rsec                                     !Preserve last second
        return
        end
```

8.34 s_sun.for

```
C
C ROUTINE           S_SUN         SUBROUTINE FORTVAX  S_SUN.FOR
C
C PURPOSE           COMPUTE SOLAR ZENITH AND AZIMUTH ANGLES
C
C DESCRIPTION       Given date, time and location on the earth's
C                   surface this routine computes a solar zenith
C                   and azimuth angle. This routine was adapted from
C                   the one used in MRF1.
C
C VERSION           1.01   301090 R.W. SAUNDERS
C
C ARGUMENTS         IDAY  I*4 IN Day in month (1-31)
C                   IMON  I*4 IN Month in year (1-12)
C                   IYR   I*4 IN Year (eg 1984)
C                   RSECS R*4 IN Time GMT (seconds from midnight)
C                   RLAT  R*4 IN Latitude degrees (north +ve)
C                   RLON  R*4 IN Longitude degrees (east +ve)
C                   AZIM  R*4 OUT Solar azimuth in degrees
C                   ZEN   R*4 OUT Solar zenith in degrees
C
C SUBPROGRAMS       DAT_CONV
C
C REFERENCES        Air Almanac useful for checking GHA and DECL
C                   Norton's Star Atlas for equation of time
C                   Robinson N. Solar Radiation Ch 2 for useful
C                   introduction to theory/terminology
C
C CHANGES           01 Documentation improved, range checks now done on
```

(continues on next page)

(continued from previous page)

```
C                     inputs RWS 30/10/90
C
C##########################################################################
      SUBROUTINE S_SUN(IDAY,IMON,IYR,RSECS,RLAT,RLON,AZIM,ZEN)
CDEC$ IDENT 'V1.01'
C
!      IMPLICIT NONE
      INTEGER*4 LASTDAY/0/
      INTEGER*4 DAYM(12)/31,29,31,30,31,30,31,31,30,31,30,31/
!      REAL*8 RSECS,RLAT,RLON,TWOPI,HALFPI,D2R,R2D,RCD,RCD2
!      REAL*8 Y,Y2,EQNT,SINALP,TANEQN,DECL,RSINDC,RCOSDC,RSINLT,RCOSLT
!      REAL*8 RGMT,TIME,HRA,RSINEV,RCOSEV,ZEN,COSAZ,AZIM
!      INTEGER*4 ICD,IDAY,IMON,IYR
      DATA TWOPI/6.283185/,HALFPI/1.570796/,D2R/0.017453/
      DATA R2D/57.29578/

      SAVE
C
      AZIM = -99.                     ! Initialise azimuth
      ZEN = -99.                      ! Initialise zenith
C
C   Perform range checking for inputs
C
      IF (IMON .GT. 12)THEN
        RETURN
      ENDIF
      IF (( RSECS .GE. 0. AND. RSECS .LE. 86400.) .AND.
     1   ( IYR .GT. 1950 .AND. IYR .LT. 2200 ) .AND.
     2   ( IMON .GE.1 .AND. IMON .LE. 12) .AND.
     3   ( IDAY .GE. 1 .AND. IDAY .LE. DAYM(IMON)) .AND.
     4   ( RLAT .GE. -90. .AND. RLAT .LE. 90. ) .AND.
     5   ( RLON .GE. -180. .AND. RLON .LE. 180. ) )THEN

C
C   Only call this section once per day
C________________________________________________________________________
C
      IF ( LASTDAY .NE. IDAY)THEN
C
C   First get century day (ie no. of days since 0-JAN-1900)
C
      CALL DAT_CONV(IDAY,IMON,IYR,ICD)
!      print *,IYR,IMON,IDAY,RLAT,RLON,ICD
C
!      print *,sizeof(ICD)
      RCD = FLOAT(ICD) / 36525.0             ! Fraction of days elapsed this
      RCD2 = RCD*RCD                         ! century
      Y = (RCD * 36000.769 + 279.697) / 360.0

!      print *,ICD
!      print *,sizeof(ICD)
!      print 10,FLOAT(ICD)
!      print 10,RCD
!      print 10,RCD2
!      print 10,Y
!10    format(f17.10)
!      print *,sizeof(Y)
```

(continues on next page)

(continued from previous page)

```
C      Y = AMOD( Y , 1.0 ) * 360.0
      Y = AMOD( Y, 1.E0) * 360.E0
      Y2 = Y * D2R
!      print *,RCD,RCD2,Y,Y2
!      print 20,Y
!20    format(f17.10)
!      print *,sizeof(RCD),sizeof(RCD2),sizeof(Y),sizeof(Y2)

C
C       Compute equation of time (in seconds) for this day
C       (No reference for this but it gives the correct answers
C       when compared with table in Norton's star Atlas)
C
      EQNT=-((93.0+14.23*RCD-0.0144*RCD2)*SIN(Y2))-((432.5-3.71*RCD
     + -0.2063
     + *RCD2)*COS(Y2))+((596.9-0.81*RCD-0.0096*RCD2)*SIN(2.0*Y2))-((1.4
     + +0.28*RCD)*COS(2.0*Y2))+((3.8+0.6*RCD)*SIN(3.0*Y2))+((19.5-0.21
     + *RCD-0.0103*RCD2)*COS(3.0*Y2))-((12.8-0.03*RCD)*SIN(4.0*Y2))
C
C     Get solar declination for given day (radians)
C
      SINALP = SIN((Y-EQNT/240.0) * D2R)
      TANEQN = 0.43382 - 0.00027*RCD
      DECL = ATAN(TANEQN*SINALP)
      EQNT = EQNT / 3600.0                   ! Convert to hours
!      print *,SINALP,TANEQN,DECL,EQNT
C
C     Sine and cosine of declination
C
      RSINDC = SIN(DECL)
      RCOSDC = COS(DECL)
!      print *,RSINDC,RCOSDC
C
      ENDIF
C_________________________________________________________________________
C
      LASTDAY = IDAY
C
      RSINLT = SIN(RLAT*D2R)                      ! Sine of lat
      RCOSLT = COS(RLAT*D2R)                      ! Cos of lat
      RGMT = RSECS / 3600.                        ! Convert secs elapsed to
C                                                 ! gmt (eg 12:30 = 12.5)
C     Calculate solar zenith (degrees)
C
      TIME = ( RLON / 15.0 ) + EQNT + RGMT         ! Local solar time (hours)
      HRA = ( TIME*15.0 + 180.0 ) * D2R            ! Local hour angle (note
      RSINEV = RSINDC*RSINLT + RCOSDC*RCOSLT*COS(HRA) ! when longitude is zero
      RCOSEV = SQRT(1.0 - RSINEV*RSINEV)           ! this equals the GHA given
      ZEN = (HALFPI - ASIN(RSINEV)) * R2D          ! in the Air Almanac)
C
C     Calculate solar azimuth (degrees)
C
      COSAZ = ( RSINDC - RSINEV*RSINLT) / (RCOSLT*RCOSEV)
      IF (COSAZ .LT. -1.0) COSAZ = -1.0
      IF (COSAZ .GT. 1.0) COSAZ = 1.0
      AZIM = ACOS(COSAZ)
      IF (AMOD(TIME+72.0,24.0) .GE. 12.0) AZIM = TWOPI-AZIM
```

(continues on next page)

(continued from previous page)

```
      AZIM = AZIM*R2D
C
      ENDIF
C

!       Following lines are for debugging purpose
!       ========================================
!       print *,IDAY,IMON,IYR,RSECS,RLAT,RLON,AZIM,ZEN
!       print *,RSECS,RLAT,RLON,TWOPI,HALFPI,D2R,R2D,RCD,RCD2
!       print *,Y,Y2,EQNT,SINALP,TANEQN,DECL,RSINDC,RCOSDC,RSINLT,RCOSLT
!       print *,RGMT,TIME,HRA,RSINEV,RCOSEV,ZEN,COSAZ,AZIM
!       print *,ICD,IDAY,IMON,IYR


      RETURN
      END
C
C ROUTINE           DAT_CONV  SUBROUTINE FORTVAX
C
C PURPOSE           To convert day,mon,yr to days since 0/1/1900
C
C DESCRIPTION       Given the day, month and year this routine
C                   computed the no. of days elpased since
C                   Jan 0 1900, the so-called century day.
C
C VERSION           1.00   130290  R.W. SAUNDERS
C
C ARGUMENTS         IDAY  I*4 IN   Day in month (1-31)
C                   IMON  I*4 IN   Month in year (1-12)
C                   IYR   I*4 IN   Year (eg 1984)
C                   ICD   I*4 OUT  Century day
C
C CHANGES           None
C
C#######################################################################
CDEC$ IDENT 'V1.00'
C
      SUBROUTINE DAT_CONV(IDAY,IMON,IYR,ICD)
C
      DIMENSION IMON2(12)
      DATA IMON2/0,31,59,90,120,151,181,212,243,273,304,334/
C
      IYD = IMON2(IMON) + IDAY !IYD is the number of days so far this year
      IF(MOD(IYR,4).EQ.0.AND.IMON.GT.2)IYD=IYD+1  !Leap year adjustment
      ILEAP = (IYR-1901) / 4 !Number of leap years since 1900 excluding this one
      ICD = (IYR-1900)*365 + ILEAP + IYD
C
      RETURN
      END
```

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index